
FEATURE EXTRACTION, CONSTRUCTION AND SELECTION:

A Data Mining Perspective

**THE KLUWER INTERNATIONAL SERIES
IN ENGINEERING AND COMPUTER SCIENCE**

FEATURE EXTRACTION, CONSTRUCTION AND SELECTION:

A Data Mining Perspective

edited by

Huan Liu

*National University of Singapore
SINGAPORE*

and

Hiroshi Motoda

*Osaka University
Osaka, JAPAN*

SPRINGER-SCIENCE+BUSINESS MEDIA, LLC

Library of Congress Cataloging-in-Publication Data

Feature extraction, construction and selection : a data mining perspective / edited by Huan Liu and Hiroshi Motoda.

p. cm. -- (The Kluwer international series in engineering and computer science ; SECS 453)

Includes bibliographical references and index.

ISBN 978-1-4613-7622-4 ISBN 978-1-4615-5725-8 (eBook)

DOI 10.1007/978-1-4615-5725-8

1. Database management. 2. Data mining. I. Liu, Huan.

II. Motoda, Hiroshi. III. Series.

QA76.9.D3F43 1998

006.3--dc21

98-24450

CIP

Copyright © 1998 by Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers in 1998

Softcover reprint of the hardcover 1st edition 1998

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher,
Springer Science+Business Media, LLC.

Printed on acid-free paper. This printing is a digital duplication of the original edition.

Contents

Preface	xiii
Acknowledgments	xv
Contributing Authors	xvii
Part I Background and Foundation	
1	
Less is More	3
<i>Huan Liu and Hiroshi Motoda</i>	
1.1 Background	3
1.2 Current Status of Feature Transformation and Subset Selection	5
1.3 Future Work	9
References	11
2	
Feature Weighting for Lazy Learning Algorithms	13
<i>David W. Aha</i>	
2.1 Introduction	13
2.2 Feature Weighting in A Lazy Learning Algorithm	15
2.3 A Categorization of Feature Weighting in Lazy Learners	16
2.4 Additional Contributions	27
2.5 Summary	28
References	29
3	
The Wrapper Approach	33
<i>Ron Kohavi and George H. John</i>	
3.1 Introduction	33
3.2 Relevance of Features	35
3.3 The Filter Approach	36

3.4	The Wrapper Approach	38
3.5	Related Work	45
3.6	Future Work	46
3.7	Conclusion	47
References		48
4	Data-driven Constructive Induction: Methodology and Applications	
<i>Eric Bloedorn, and Ryszard S. Michalski</i>		51
4.1	Introduction	52
4.2	An Illustration of the Importance of the Representation Space	53
4.3	The Need for Constructive Induction	54
4.4	A General Schema for Constructive Induction	56
4.5	Experimental Applications	59
4.6	Conclusion	66
References		67
Part II Subset Selection		
5	Selecting Features by Vertical Compactness of Data	
<i>Ke Wang and Suman Sundaresan</i>		71
5.1	Introduction	71
5.2	The Vertical Compactness Criterion	73
5.3	Search Methods	75
5.4	Hybrid Search Algorithm	77
5.5	Experiments	80
5.6	Conclusions	83
References		84
6	Relevance Approach to Feature Subset Selection	
<i>Hui Wang, David Bell, and Fionn Murtagh</i>		85
6.1	Introduction	85
6.2	Characterisation of Feature Subset Selection	87
6.3	A Relevance-based Algorithm for Feature Selection	90
6.4	Experiment and Evaluation	91
6.5	Comparison with Related Work	92
6.6	Conclusion	95
Appendix: A unified framework for relevance		96
References		97
7	Novel Methods for Feature Subset Selection with Respect to Problem Knowledge	
<i>Pavel Pudil and Jana Novovičová</i>		101
7.1	Introduction	101

7.2	Feature Subset Selection Problem in Statistical Pattern Recognition	103
7.3	Basic Situation with Respect to Problem Knowledge	105
7.4	Floating Search Methods	106
7.5	Feature Selection by Modified Gaussian Mixtures	108
7.6	Experimental Results	112
7.7	Subset Selection Guide	114
References		115
8	Feature Subset Selection Using A Genetic Algorithm	117
<i>Jihoon Yang and Vasant Honavar</i>		
8.1	Introduction	117
8.2	Related Work	118
8.3	Feature Selection Using a Genetic Algorithm for Neural Network Pattern Classifiers	122
8.4	Implementation Details	125
8.5	Experiments	127
8.6	Summary and Discussion	131
References		133
9	A Relevancy Filter for Constructive Induction	137
<i>Nada Lavrač, Dragan Gamberger, and Peter Turney</i>		
9.1	Introduction	137
9.2	Relevance of Literals and Features	139
9.3	Utility Study: The East-West Challenge	144
9.4	Handling Noisy Data	150
9.5	Noise and Relevance	151
9.6	Related Work	151
9.7	Conclusion	152
References		153
Part III Feature Extraction		
10	Lexical Contextual Relations for the Unsupervised Discovery of Texts Features	157
<i>Patrick Perrin and Fred Petry</i>		
10.1	Introduction	157
10.2	Collocational Expressions as Text Features	160
10.3	Discovery of Contextual Text Features	163
10.4	Results and Discussion	164
10.5	Conclusion	170
References		172
11	Integrated Feature Extraction Using Adaptive Wavelets	175

<i>Yvette Mallet, Olivier de Vel, and Danny Coomans</i>	
11.1 Introduction	175
11.2 Wavelet Analysis of Spectra	179
11.3 Multivariate Prediction Models	183
11.4 The Data Set	185
11.5 Classification Applications	185
11.6 Regression Applications	187
11.7 Future Directions	188
References	189
12 Feature Extraction via Neural Networks	191
<i>Rudy Setiono and Huan Liu</i>	
12.1 Introduction	191
12.2 Feature Extraction via Neural Networks	193
12.3 Illustrative Examples	195
12.4 Empirical Study and Analysis	199
12.5 CNF9a Revisited for Decision Tree Induction	200
12.6 Summary	202
Appendix: Definitions for artificial problems	202
References	202
13 Using Lattice-based Framework as a Tool for Feature Extraction	205
<i>E. Mephu Nguifo, P. Njiwoua</i>	
13.1 Introduction	205
13.2 Galois Lattice	207
13.3 Feature Extraction Algorithm	209
13.4 The IGLUE System	212
13.5 Experimental Results	213
13.6 Conclusion	216
References	217
14 Constructive Function Approximation	219
<i>Paul E. Utgoff and Doina Precup</i>	
14.1 Introduction	219
14.2 The Need for Features	220
14.3 Feature Spaces	221
14.4 Comparison of Three Function Approximation Algorithms	223
14.5 Related Work	233
References	235
Part IV Feature Construction	
15	

A Comparison of Constructing Different Types of New Feature for Decision Tree Learning <i>Zijian Zheng</i>	239
15.1 Introduction	239
15.2 Four Types of New Feature	241
15.3 A Single Algorithm for Creating Four Different Types of New Feature	243
15.4 Experiments	244
15.5 Discussion	251
15.6 Related Work	252
15.7 Conclusions and Future Work	253
References	254
16 Constructive Induction: Covering Attribute Spectrum <i>Yuh-Jyh Hu</i>	257
16.1 Introduction	257
16.2 Important Issues for Constructive Induction	258
16.3 GALA2.0: Combining Neural Network with Iterative Attribute Construction	261
16.4 Discussion	269
References	271
17 Feature Construction Using Fragmentary Knowledge <i>Steve Donoho and Larry Rendell</i>	273
17.1 Introduction	273
17.2 Types of Fragmentary Knowledge	274
17.3 Incorporating Fragmentary Knowledge into Search	278
17.4 Bankruptcy Experiments	280
17.5 Turfgrass Management Experiments	283
17.6 Discussion	286
References	287
18 Constructive Induction on Continuous Spaces <i>João Gama and Pavel Brazdil</i>	289
18.1 Constructive Induction	289
18.2 An Illustrative Example	291
18.3 Growing the Tree	294
18.4 Related Work	298
18.5 Experiments	299
18.6 Conclusions	301
References	302

Part V Combined Approaches

Evolutionary Feature Space Transformation	307
<i>Haleh Vafaei and Kenneth De Jong</i>	
19.1 Introduction	307
19.2 Approaches to Feature Selection and Construction	308
19.3 General System Architecture	310
19.4 Experimental Setup	315
19.5 Comparative studies	320
19.6 Summary and Conclusions	321
References	322
20	
Feature Transformation by Function Decomposition	325
<i>Blaž Zupan, Marko Bohanec, Janez Demšar, and Ivan Bratko</i>	
20.1 Introduction	325
20.2 Single-step Function Decomposition	327
20.3 Finding the Best Feature Partition	329
20.4 Redundancy Discovery and Removal	330
20.5 Discovering Feature Hierarchies	331
20.6 Feature Construction	334
20.7 Related Work	336
20.8 Summary	338
References	338
21	
Constructive Induction of Cartesian Product Attributes	341
<i>Michael J. Pazzani</i>	
21.1 Introduction	341
21.2 A Wrapper Approach for Creating Cartesian Product Attributes	342
21.3 Experiments	344
21.4 Attribute deletion	349
21.5 Related Work	350
21.6 Conclusions	351
References	352
Part VI Applications of Feature Transformation	
22	
Towards Automatic Fractal Feature Extraction for Image Recognition	357
<i>Matteo Baldoni, Cristina Baroglio, Davide Cavagnino and Lorenza Saitta</i>	
22.1 Introduction	357
22.2 Fractals and IFS	359
22.3 Extraction of Fractal Features	361
22.4 Experiments	363
22.5 Discussion and future work	370
References	372

23	
Feature Transformation Strategies for a Robot Learning Problem <i>Luis Seabra Lopes, Luis M. Camarinha-Matos</i>	375
23.1 Introduction	375
23.2 Robot Programming by Demonstration	376
23.3 Learning Structured Concepts	377
23.4 Constructive Induction	379
23.5 Experimental Situation: the Pick and Place Task	380
23.6 Feature Transformation Strategies	381
23.7 Experimental Results	386
23.8 Conclusions	390
References	391
24	
Interactive Genetic Algorithm Based Feature Selection and Its Application to Marketing Data Analysis <i>Takao Terano and Yoko Ishino</i>	393
24.1 Introduction	394
24.2 Interactive GAs, Feature Selection, and Knowledge Extraction	395
24.3 Algorithm of SIBILE	396
24.4 Experiments	399
24.5 Concluding Remarks	404
References	405
Index	407

Preface

There is broad interest in feature extraction, construction, and selection among practitioners from statistics, pattern recognition, and data mining to machine learning. Data preprocessing is an essential step in the knowledge discovery process for real-world applications. This book compiles contributions from many leading and active researchers in this growing field and paints a picture of the state-of-art techniques that can boost the capabilities of many existing data mining tools.

The objective of this collection is to increase the awareness of the data mining community about the research of feature extraction, construction and selection, which are currently conducted mainly in isolation. This book is part of our endeavor to produce a contemporary overview of modern solutions, to create synergy among these seemingly different branches, and to pave the way for developing meta-systems and novel approaches.

Even with today's advanced computer technologies, discovering knowledge from data can still be fiendishly hard due to the characteristics of the computer generated data. Feature extraction, construction and selection are a set of techniques that transform and simplify data so as to make data mining tasks easier. Feature construction and selection can be viewed as two sides of the representation problem. In some cases, the representation language contains more features than necessary, feature selection can be employed to simplify the language; in other cases, the language may be insufficient to describe the problem for some learning algorithms, feature construction can be engaged to enrich the language. It is often the case that some constructed features are not useful at all. Feature selection can then remove these useless features. It is also common to see the combined use of feature extraction and selection. Some chapters in this book skillfully combine various methods. These techniques aim at preserving the topological structure of the data or at enhancing the discriminatory power.

This book consists of five parts each of which is made up of three to five chapters, a total of twenty four chapters. Except for Part 1 which is about background, foundation, and general approaches to feature construction, ex-

traction and selection, each part revolves around a theme and can be read independently of other parts. Each chapter is a contribution of respective researchers. Part 2, Part 3 and Part 4 respectively cover subset selection, feature extraction and feature construction. Throughout this book, we distinguish these three topics as follows. Subset selection does not generate any new features but only selects a subset of original features and, thus, the feature space is reduced. Feature extraction is a process that extracts a set of new features from the original ones through some functional mapping and usually reduces the feature space. Feature construction, on the other hand, is a process that discovers missing relations among features and augments the space of features by inferring or creating additional compound features, and thus could expand the feature space. As a result, some original or constructed features may not be necessary. Part 4 describes the combined use of feature extraction, construction and selection. Part 5 introduces some real-world applications using these feature manipulation techniques. As a chapter may cover multiple aspects to different extents, the allocation of a chapter to a part is not absolute. Many chapters are indeed cross-linked to one another.

The book can be used by researchers and graduate students in machine learning, data mining, and knowledge discovery, who wish to understand techniques of feature extraction, construction and selection for data pre-processing and to solve large size, real-world problems. The book can also be served as a reference book for those who are conducting the research about feature extraction, construction and selection, and are ready to meet the exciting challenges ahead of us.

HUAN LIU AND HIROSHI MOTODA

Acknowledgments

First and foremost we want to acknowledge the contributors - 48 authors from 12 countries. Without their broad experience in developing theories and practicing with data, this book would not have been possible. In addition, we thank them for their forbearance when the process of putting this book together took more time than anyone anticipated.

We are grateful to the editorial staff of Kluwer Academic Publishers, especially Sharon Fletcher and Scott Delman for their patience, interest, and helpfulness in bringing this project to a successful conclusion.

All during the editing process we were generously supported by our colleagues and friends, our special thanks to Kaitat Foong and Kiansing Ng for their innovative cover design for this book.

Contributing Authors

David W. Aha (NCARAI, NRL) conducts research on machine learning and case-based reasoning. He leads two projects, serves on the editorial boards for *ML*, *JAIR*, and *Applied Intelligence*, and edited the 1997 special issue/book on *Lazy Learning* for *AI Review*. David also serves on the steering committee for AIRIES and the Technical Advisory Board for Inference Corporation.

Matteo Baldoni is currently a Ph.D. student at the Department of Computer Science in the University of Turin, where he took his “Laurea” degree “summa cum laude” in Computer Science in 1993. His main research interests include automatic reasoning, knowledge representation, modal and non-monotonic extensions of logic programming.

Cristina Baroglio was born in 1967, she graduated in Computer Science in 1991 at the University of Turin, where she also got a Ph.D. in Cognitive Sciences in 1996. She investigated different aspects of Machine Learning: from symbolic induction to reinforcement learning.

David Bell has been Professor of Computing at the University of Ulster in Northern Ireland since 1986. He is now Head of School of Information and Software Engineering. He was until recently Editor of the “Database Technology” Journal and is currently on the Editorial Board of “Information Systems”. He was the 1993 Co-chair of the Programme Committee of the VLDB Conference and serves on many international Programme Committees.

Eric Bloedorn is a senior staff member of the Artificial Intelligence Technical Center at the MITRE Corporation. His interests include machine learning and its application to data and text mining. Dr. Bloedorn received his B.A. degree in Physics from Lawrence University in 1989, and his M.Sc. and Ph.D from George Mason University in 1992 and 1996 respectively.

Marko Bohanec is Research Associate at the Department of Intelligent Systems, J. Stefan Institute, Ljubljana, Slovenia. His research and development interests are in decision support systems and machine learning. He has published in journals such as *Machine Learning*, *Acta Psychologica*, and *Information & Management*.

Ivan Bratko is Professor of computer science at the Faculty of Computer and Information Science, Ljubljana University, Slovenia. He heads the AI laboratories at J. Stefan Institute and the University. He has conducted research in machine learning, knowledge-based systems, qualitative modeling, intelligent robotics, heuristic programming and computer chess.

Pavel Brazdil has completed his Ph.D. degree at the University of Edinburgh in 1981. Currently he holds the post of a Professor at the University of Porto, and co-directs the Laboratory of Artificial Intelligence and Computer Science (LIACC). His research main interests include machine learning, meta-learning, inductive logic programming, knowledge discovery in databases and data mining.

Luis M. Camarinha-Matos obtained his computer science degree from Universidade Nova de Lisboa, Portugal, in 1979, and the Ph.D degree in the topic of robotics and CIM from the same university in 1989. His research interests are CIM systems integration and intelligent manufacturing systems.

Davide Cavagnino received his “Laurea” degree “summa cum laude” in Computer Science at the University of Turin, Italy, in 1992. He is currently a Ph.D student at the Department of Computer Science in the University of Turin. His main research interests include signal and image processing, and fuzzy logic.

Danny Coomans received his Ph.D in pharmaceutical sciences from the Free University of Brussels, Belgium. He is Associate Professor in the Department of Mathematics and Statistics, James Cook University, Townsville. His present research interests lie in the areas of pattern recognition and dynamic models.

Kenneth De Jong is a faculty member of the Computer Science at George Mason University in Fairfax, Virginia. He is an active member of the Genetic Algorithms research community with a variety of papers, Ph.D students, and presentations in this area. He is a member of the executive council of the International Society for Genetic Algorithms and the founding editor of the journal *Evolutionary Computation*.

Janez Demšar is Research Assistant at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. He has graduated in computer science at the same faculty. His research interest include machine learning and intelligent data analysis.

Olivier de Vel obtained a M.Sc.(Hons) from the University of Waikato (New Zealand) in 1974 and a Docteur de 3ième Cycle from the Institute National Polytechnique of Grenoble (France) in 1978. He is currently Associate Professor in the Department of Computer Science, James Cook University, Townsville. His current research interests include machine learning, wavelets and web mining.

Steve Donoho received his Ph.D from the University of Illinois in 1996. He is currently a research scientist at SRA International, Inc. His research interests include fraud detection and change of representation.

João Gama teaches at the Faculty of Economics, University of Porto. Currently he is completing his Ph.D at the Laboratory of Artificial Intelligence and Computer Science of the some University. His main interests include Characterization of learning algorithms and Combining Classifiers

Dragan Gamberger is Research Associate at the Department of Physics, R. Bošković Institute, Zagreb, Croatia. His research interests include machine learning and applications of machine learning in medicine and ecology. He developed the inductive learning system ILLM (Inductive Learning by Logic Minimization).

Vasant Honavar is Assoc. Prof. of Computer Science/Neuroscience at Iowa State Univ. His research interests include AI, intelligent agents, machine learning, data mining and knowledge discovery, neural and evolutionary computing, and bioinformatics. Dr. Honavar holds a B.E. in Electronics Engg. from Bangalore Univ. (India), an M.S in Electrical/Computer Engg. from Drexel Univ., and M.S and Ph.D degrees in Computer Science from the Univ. of Wisconsin.

Yuh-Jyh Hu is a Ph.D student in Information and Computer Science, University of California, Irvine. His research interest includes machine learning, data mining and biopattern discovery. He got his Master degrees in Computer Science from University of Southern California in 1992, and from University of California, Irvine in 1994.

Yoko Ishino received her B.A. degree in 1987 from the University of Tokyo, and M.A degree in 1978 from GSSM, The University of Tsukuba, Tokyo. Dur-

ing 1987 and 1996, she was a member of a marketing intelligence division of a company. Currently, she is a Ph.D student of the University of Tokyo, Japan. Her research interests include marketing sciences and artificial intelligence.

George H. John is the Data Mining Guru at Epiphany Marketing Software, where he is developing third-generation data mining technology and applications for marketing. He earned a Ph.D with Distinction from the Computer Science Department at Stanford University, where his research was supported by a National Science Foundation fellowship. His thesis, *Enhancements to the Data Mining Process*, is available at <http://xenon.stanford.edu/~gjohn>.

Ron Kohavi is the engineering manager for MineSet, Silicon Graphics' award-winning product for data mining and visualization. Dr. Kohavi received his Ph.D from Stanford University, where he led the *MLC++* project, now used in MineSet and at several universities. He serves on the editorial board for the Data Mining and Knowledge Discovery journal and the journal Machine Learning. He co-edited the ML special issue on applications of machine learning.

Nada Lavrač is Research Associate at the Department of Intelligent Systems, J. Stefan Institute, Ljubljana, Slovenia. Her research interests include machine learning, inductive logic programming and intelligent data analysis in medicine. She is coauthor of numerous publications, including *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood 1994.

Huan Liu is Senior Lecturer with Department of Information Systems and Computer Science, National University of Singapore. He was with Telecom Australia Research Labs from Nov. 89 to Jan. 94. He obtained his Ph.D from University of Southern California. His interests include feature transformation, subset selection, and rule extraction.

Luis Seabra Lopes obtained his computer science degree from Universidade Nova de Lisboa, Portugal in 1990. He stayed at this university as a researcher until 1996 and is now an assistant professor at Universidade de Aveiro, Portugal. His research interests are robotics, machine learning, artificial intelligence and flexible manufacturing systems.

Yvette Mallet obtained the Degree Bachelor of Science (Hons) from James Cook University of North Queensland in 1993 and has recently submitted her Ph.D thesis. Her interests are in the area of wavelet based feature extraction methods for the discrimination and regression of spectral data.

Ryszard S. Michalski is Planning Research Corporation Chaired Professor of Computer Science and Systems Engineering, Affiliate Professor at the Institute of Computer Science, Polish Academy of Sciences in Warsaw, and Director of Machine Learning and Inference Laboratory. He has authored more than 350 publications in machine learning and inference, data mining cognitive modeling, applications of machine learning to computer vision, and intelligent agents.

Hiroshi Motoda is Professor of the Division of Intelligent Systems Science at the Institute of Sicutific and Industrial Research of Osaka University. He was with Hitachi, Ltd. from 1967 to 1995. He obtained his Ph.D from University of Tokyo. His research interests include machine learning, knowledge acquisition, scientific knowledge discovery, data mining, heterogeneous reasoning, qualitative reasoning, information filtering, and knowledge based systems.

Fionn Murtagh (Ph.D Université P.&M. Curie, Paris; Habilitation Université Louis Pasteur, Strasbourg) is Full Professor of Computing Science at the University of Ulster. He was with the European Space Agency's Space Science Department for 12 years in Germany. He is a member of the editorial boards of Journal of Classification, Neurocomputing, Artificial Intelligence Review, and other journals.

Patrick Njiwoua is a Ph.D student at the University of Artois at Lens (France). He earned its master degree from University of Lyon (France) working on Lattices and Parallelism. His Ph.D thesis focuses on the design and use of lattice-based machine learning systems. He also teaches Programming Language and Operating Systems at the Institute of Technology of Lens.

Jana Novovičová received RNDr. in Mathematics from Charles University, Faculty of Mathematics and Physics, Prague, and Ph.D in Theoretical Cybernetics from the Inst. of Information Theory and Automation, Czechoslovak Academy of Sciences, Prague, where she is currently a Senior Research Fellow. Her primary research interests include statistical approach to pattern recognition and robust statistical estimation methods.

Engelbert Mephu Nguifo is Lecturer and Chairman of the Department of Computer Science, Institute of Technology (IUT), University of Artois at Lens (France), where he has taught since 1994. He earned his Ph.D from University of Montpellier II (France). His research main interests are in the areas of AI, Machine Learning and Concept Lattices.

Michael J. Pazzani is Professor and Department Chair of Information and Computer Science at the University of California, Irvine. Dr. Pazzani has

been active in Machine Learning research for the past decade with numerous publications in KDD, IJCAI, AAAI, and ICML. He has served on the editorial board of Machine Learning and the Journal of Artificial Intelligence Research, and is currently the moderator of the machine learning list.

Patrick Perrin is Visiting Research Assistant Professor at Tulane University. He received a B.S. in Computer Engineering from Belle-Beille Institute of Technology, Angers, France; a M.S and Ph.D in Computer Science from Tulane University, New Orleans, LA. His interests are in large-scale unsupervised information extraction, knowledge discovery, machine learning, discourse analysis, and computational linguistics. He is a member of AAAI and ACM.

Fred Petry is Full Professor at Tulane University. He received BS and MS in physics and a Ph.D in computer science from Ohio State University in 1975. His interests include representation of imprecision in databases and other information systems and has over 175 scientific publications and 5 books.

Doina Precup is a third year Ph.D student in the Department of Computer Science at the University of Massachusetts at Amherst. In 1994, she earned her BSc from the Technical University Cluj-Napoca in Romania, and in the following year she received a Fulbright scholarship. Her current Ph.D research, under the direction of Rich Sutton, focuses on using spatial and temporal abstraction to enhance reinforcement learning algorithms.

Pavel Pudil received a Ph.D in Technical Cybernetics from the Czechoslovak Academy of Sciences, Prague, in 1970. He is currently the Head of Pattern Recognition Dept. in the Inst. of Information Theory and Automation, Academy of Sciences of the Czech Republic, where his research interests include statistical approach to pattern recognition, particularly the problem of dimensionality reduction.

Larry Rendell completed his Ph.D at the University of Waterloo in 1981. His early work integrated pattern recognition methods with artificial intelligence approaches. He is currently an Associate Professor of Computer Science at the University of Illinois where his interests include constructive induction and change of representation.

Lorenza Saitta is Full Professor of Computer Science at the University of Turin. In 1984 she started working in Machine Learning, initiating thus the research in the field in Italy. Her interests moved from inductive symbolic approaches to integrated learning strategies, based on more complex reasoning schemes, and also to Genetic Algorithms.

Rudy Setiono is Senior Lecturer with Department of Information Systems and Computer Science, National University of Singapore. He earned his Ph.D from the University of Wisconsin-Madison in 1990. His research interests include neural network construction and pruning, and rule extraction from neural networks.

Suman Sundaresh is Research Assistant in the Medical Computing Laboratory at the National University of Singapore. Her research interests include data mining, machine learning and dynamic decision modelling.

Takao Terano received his B. A. in 1976 and M.A degree in 1978, both from the University of Tokyo, and had a Ph.D in 1991 from Tokyo Institute of Technology. He is currently Professor at the Graduate School of Systems Management (GSSM), The University of Tsukuba, Tokyo. His current research includes GA-based Machine Learning, Agents, and Computational Organization Theory.

Peter Turney is Research Officer in the Interactive Information Group of the National Research Council of Canada. His areas of expertise are machine learning and information retrieval. He is the author or co-author of more than fifty publications, a past editor of Canadian AI magazine, and a member of the editorial board of the Journal of Artificial Intelligence Research.

Paul Utgoff is Associate Professor in the Department of Computer Science at the University of Massachusetts at Amherst, where he has taught since 1985. He is a former Action Editor of Machine Learning, and in 1993 served as Conference Chair for the Tenth International Conference on Machine Learning. His principal research interest is in developing methods for automatic construction of representations that facilitate machine learning.

Haleh Vafaie is currently a senior scientist at Datamat Research, Inc. Her research interests include artificial intelligence, genetic algorithms, machine learning, and computer vision. She has published several conference and journal papers in these areas. She received her Ph.D in Information Technology from George Mason University, USA.

Hui Wang is Lecturer in Faculty of Informatics, University of Ulster. He earned his BSc and MSc in Jilin University, P. R. China, and his Ph.D in University of Ulster, N. Ireland. His research interests include statistical and algebraic machine learning, data mining, neural networks, and information retrieval and filtering.

Ke Wang is Senior Lecturer at National University of Singapore. Dr. Wang's recent interests of research are Web data mining, interestingness of knowledge, data mining for large databases, data mining for less structured data.

Jihoon Yang is a graduate student of Computer Science at Iowa State Univ. His research interests include intelligent agents, data mining and knowledge discovery, machine learning, neural networks, pattern recognition, and evolutionary computing. He holds a B.S. in Computer Science from Sogang Univ. (Seoul, Korea), an M.S in Computer Science from Iowa State Univ., and is currently working towards his Ph.D in Computer Science at Iowa State Univ.

Zijian Zheng is Research Fellow in School of Computing and Mathematics, Deakin University, Australia where he has done research on machine learning and data mining. He earned his Ph.D in Computer Science from the University of Sydney in 1996.

Blaž Zupan is Researcher at the Department of Intelligent Systems, J. Stefan Institute, Ljubljana, Slovenia. His research interests include machine learning, data mining, medical decision making, and medical informatics.

| Background and Foundation

1 LESS IS MORE

Huan Liu¹ and Hiroshi Motoda²

¹Department of Information Systems & Computer Science
National University of Singapore
Singapore 119260
liuh@iscs.nus.edu.sg

²Institute of Scientific & Industrial Research
Osaka University
Ibaraki, Osaka 567-0047, Japan
motoda@sanken.osaka-u.ac.jp

1.1 BACKGROUND

As computer and database technologies rapidly advance, human beings rely more and more on computers to accumulate data, process data, and make use of data. Machine learning, knowledge discovery, and data mining are some intelligent tools that help mankind accomplish those tasks. Researchers and practitioners realize that in order to use these tools effectively, an important part is pre-processing in which data is processed before it is presented to any learning, discovering, or visualizing algorithm. In many discovery applications (for example, marketing data analysis), a key operation is to find subsets of the population that behave enough alike to be worthy of focused analysis (Brackman and Anand, 1996). Although many learning methods attempt to select, extract, or construct features, both theoretical analyses and experimental studies indicate that many algorithms scale poorly in domains with large numbers of irrelevant and/or redundant features (Langley, 1996). All the evidence suggests the need for additional methods to overcome the difficulties.

Feature transformation and subset selection are some frequently used techniques in data pre-processing. Feature transformation is a process through

Portions reprinted, with permission, from IEEE Intelligent Systems; Volume 13, Issue 2, Pages 26-28, March/April 1998. ©1998 IEEE.

H. Liu et al. (eds.), *Feature Extraction, Construction and Selection*
© Kluwer Academic Publishers 1998

which a new set of features is created. The variants of feature transformation are feature construction and feature extraction. Both are sometimes called feature discovery. Assuming the original set consists of A_1, A_2, \dots, A_n features, these variants can be defined below.

Feature construction is a process that discovers missing information about the relationships between features and augments the space of features by inferring or creating additional features (Matheus, 1991; Wnek and Michalski, 1994). After feature construction, we may have additional m features $A_{n+1}, A_{n+2}, \dots, A_{n+m}$. For example, a new feature A_k ($n < k \leq n + m$) could be constructed by performing a logical operation on A_i and A_j from the original set. Another example is: a two-dimensional problem (say, $A_1=\text{width}$ and $A_2=\text{length}$) may be transformed to a one-dimensional problem ($B_1=\text{area}$) after B_1 is discovered.

Feature extraction is a process that extracts a set of new features from the original features through some functional mapping (Wyse et. al., 1980). After feature extraction, we have B_1, B_2, \dots, B_m ($m < n$), $B_i = F_i(A_1, A_2, \dots, A_n)$, and F_i is a mapping function. For instance, $B_1 = c_1A_1 + c_2A_2$ where c_1 and c_2 are coefficients.

Subset selection is different from feature transformation in that no new features will be generated, but only a subset of original features is selected and the feature space is reduced (Blum and Langley, 1997; Dash and Liu, 1997).

As to feature transformation, feature construction often expands the feature space, whereas feature extraction usually reduces the feature space. Feature transformation and subset selection are not two totally independent issues. They can be viewed as two sides of the representation problem. We can consider features as a representation language. In some cases where this language contains more features than necessary, subset selection helps simplify the language; in other cases where this language is not sufficient to describe the problem, feature construction helps enrich the language by constructing compound features. It is common that some constructed features are not useful at all. Subset selection can then remove these useless features. It is also common to see the combined use of feature extraction and subset selection. Some chapters in this issue skillfully combine various feature transformation and subset selection methods. The use of feature transformation and subset selection depends on the purpose - for simpler concept description or for better classification. The former aims at preserving the topological structure of the data whereas the latter targets enhancing the predictive power.

Learning algorithms can be categorized as being **eager** or **lazy**, depending on the degree to which they process their inputs prior to their use in performing tasks. Many algorithms eagerly compile input samples and use only compiled results to make decisions. Lazy learning does not generalize from the data when the data is available; instead, it delays the learning until it is necessary. (Aha, 1998) discussed lazy learning algorithms and feature selection. Feature weighting is often considered for the task of selection. Feature selection algorithm can be classified as a **filter** or a **wrapper**, depending on whether it is treated as

a pre-process or intertwined with the learning task. (Kohavi and John, 1998) discussed in detail a wrapper approach. A wrapper approach generally outperforms a filter one because it directly optimizes the evaluation measure of the learning task while removing features, but the time needed to complete feature selection is much longer than a filter approach.

There is broad interest in feature transformation and subset selection among practitioners from statistics, pattern recognition, data mining, and knowledge discovery to machine learning since data pre-processing is an essential step in the knowledge discovery process for real-world applications. The objective of this book is to report on the recent studies in feature transformation and subset selection, and to increase the awareness of various research communities about the state-of-art work on feature transformation and subset selection, mainly conducted in isolation so far. Through this collection, we hope to present an updated overview of modern solutions, to create synergy among these seemingly different branches but with a similar goal - facilitating data processing and knowledge discovery, and to point to future research directions.

The 24 chapters in this book cover various aspects of feature transformation and subset selection:

- General description of feature transformation and subset selection;
- New methods of feature extraction, construction, and selection;
- Combinations of different methods like machine learning, statistics, neural networks, and genetic algorithms in feature transformation and subset selection; and
- Applications of feature transformation and subset selection to real-world problems.

1.2 CURRENT STATUS OF FEATURE TRANSFORMATION AND SUBSET SELECTION

This collection includes contemporary descriptions of representative research work and applications, with comprehensive references. The self-complete chapters will help readers get a grasp of techniques for feature transformation and subset selection and the use of them in data mining. Starting from chapters of particular interest and following the references, the reader will find an extensive and active field of feature transformation and subset selection. Before we begin the journey with twenty some exciting “stops”, let’s have a preview of what is ahead of us, which is presented in the following eight sub-sections. In this collection, features and attributes are used interchangeably if not explicitly stated.

1.2.1 Characterization of Subset Selection

Subset selection is defined as finding a minimum subset that satisfies a criterion, be it a filter or a wrapper. The criterion can be error rate, inconsistency rate, information measure, distance measure, or dependence measure. Another way of formulating the problem of subset selection, which is sort of dual, is to find a subset that minimizes the number of projected instances within the allowable

bound of a criterion (consistency rate in this case) as proposed by (Wang and Sundaresh, 1998). Both definitions are related to the notion of relevance (Blum and Langley, 1997). (Wang et. al., 1998) attempted to characterize subset selection in the relevance framework that has a mathematical foundation. Two axioms: the sufficiency axiom (preserving learning information) and necessity axiom (minimizing encoding length) are shown to be equivalent to maximizing the relevance in both ways (i.e., relevance of a feature subset to a class and relevance of a class to a feature subset). (Lavrač et. al., 1998) proposed a filter selector that selects the potentially relevant features from the set of constructed features. The degree of relevance is measured by p/n (positive pairs/negative ones) which can be used to identify irrelevant features by a set-inclusion relation. Negation is explicitly treated in their approach (i.e., a feature is irrelevant only if both of its corresponding positive and negative literals are irrelevant).

1.2.2 New Search Methods

Search plays an important role in subset selection (Blum and Langley, 1997; Liu and Motoda, 1998). **Search** can be characterized by search direction (forward, backward, bidirectional and random), search strategy (complete/exhaustive, heuristic and non-deterministic), and evaluation measure (accuracy, consistency and classic). (Kohavi and John, 1998) proposed a new way of changing the search space topology by creating dynamic operators that directly connect a node to other nodes considered. These compound operators make a backward search, starting from the full set of features, practical. (Wang and Sundaresh, 1998) proposed a simple combination of breadth first and depth first search - another hybrid search method to quickly prune the search space for the monotonic evaluation function. (Pudil and Novovičová, 1998)'s "floating" algorithm is another innovative idea that is a variation of bidirectional search in which it allows a variable (thus floating) number of features to be included or excluded.

1.2.3 Constructive Operators

Conjunction, disjunction, and negation are commonly used constructive operators for nominal feature values. Another two constructive operators are *M-of-N* (*M* out of *N*) and *X-of-N* (how many *X* is true of *N*). (Zheng, 1998) compared the performances of these operators. Yet another operator is the Cartesian product which was investigated in depth by (Pazzani, 1998). All these operators work on and produce binary features. For numerical features, simple algebraical operators such as equivalence, inequality, addition, subtraction, multiplication, division, maximum, minimum, average are often used to construct compound features. The skillful application of these operands can be seen in (Vafaie and De Jong, 1998; Bloedorn and Michalski, 1998).

1.2.4 Extracting Features

Inducing an oblique decision tree is becoming popular, where the decision surface of each test node is oblique to the axes defined by the original features. Linear discriminant analysis is often used to extract a feature for continuous

variables. (Gama and Brazdil, 1998)'s probabilistic decision tree propagates the new features, extracted in this way, downwards through the tree, thus allowing non-linear fittings.

Another common method of feature extraction is to use a feed forward neural network. The hidden unit activations are interpreted as new features extracted from the original dataset. The work by (Hu, 1998; Setiono and Liu, 1998; Utgoff and Precup, 1998) follows this idea. Utgoff and Precup tried to solve the problem of incremental constructive function approximation and they proposed two other extraction methods of covering techniques with repeated local error minimization.

Four other interesting approaches introduced in this book are fractal encoding (Baldoni et. al., 1998), use of wavelet (Mallet et. al. 1998), use of mutual information (Perrin and Petry, 1998) and conversion of binary features to continuous ones (Mephu Nguifo and Njiwoua, 1998). One of the advantages of fractal is that it is invariant with respect to the size of the objects. Wavelets are most effective in extracting the important local features in spectra of very high dimensionality. Mutual information can be used to successfully extract a set of informative collocational expressions as text features without the need for any extra sources of knowledge. A numerical taxonomy relies on a similarity measure of numerical value and the use of a distance measure implicitly assumes that objects can be represented naturally in terms of continuously valued variables. When a numerical taxonomy is needed, it is necessary to transform nominal features into continuous ones.

1.2.5 Using Knowledge

Although the majority of the chapters have a data-driven flavor, the importance of knowledge is recognized in several systems. This is because (a) different problems may require different approaches to solving them, and (b) domain knowledge helps effective search for relevant features. Whereas strong knowledge provides straightforward means to improve feature transformation and subset selection, weak knowledge should also be used. One such example is exhibited in (Donoho and Rendell, 1998). Fragmentary knowledge is employed to narrow the search space. They identified types of fragmentary knowledge (i.e., relevance, support, correlation, contingency, normalization, proximity, inheritance hierarchies, dimensional analysis) and showed how these different types of knowledge constrain a search starting point, search boundaries and connectivity of search. (Pudil and Novovičová, 1998) also addressed the importance of knowledge about the prior probability distribution. (Yang and Honavar, 1998) pointed out that their multi-criteria fitness function can be improved by using domain knowledge.

1.2.6 Combination of Feature Transformation and Subset Selection

This collection demonstrates that researchers are starting to explore the advantages of both feature transformation and subset selection instead of individually considering feature extraction, construction and selection. (Bloedorn and

Michalski, 1998) investigated feature selection, quantization and construction as operators in their data-driven constructive induction. The order of applying these operators can be either specified by a user or predefined, although it is still premature to draw general conclusions about the best ordering. (Lavrač et. al., 1998) first expanded the initial features and then pruned the irrelevant or unimportant ones, both operations are performed systematically in an inductive logic programming setting. (Vafaie and De Jong, 1998) applied genetic algorithms to both feature selection and construction which are employed in turn to search for adequate but necessary features. (Pazzani, 1998)'s algorithm constructs new features via Cartesian product, but it also allows deletion of both original and constructed features. (Zupan et. al., 1998) suggested to use function decomposition to identify appropriate subsets of existing features, to eliminate redundant features, and discover a hierarchy of new features that can be added into the existing feature set. The work by (Gama and Brazdil, 1998) and (Zheng, 1998) can also be considered a combined approach because they both use a decision tree induction algorithm as a feature selector.

1.2.7 Complimentary Coverage

The various techniques of feature transformation and subset selection discussed in this collection have been successfully applied with the techniques of pattern recognition, statistics, machine learning, neural networks, and genetic algorithms. Such good examples can be seen in the three papers using genetic algorithm. (Vafaie and De Jong, 1998) explained in detail how feature construction and selection are done in a genetic algorithm; in order to speed up the calculation of the fitness function, they used C4.5's accuracy as the fitness measure, although it may render the selected and constructed features improper for non-linear classifiers to use, e.g., neural networks. (Yang and Honavar, 1998) suggested, in a complimentary way, to use a relatively fast inter-pattern distance-based neural network learning algorithm in the fitness function. They also discussed the issue of multi-criteria optimization which often occurs in practice where both accuracy and cost of classification should be considered. On the other hand, (Terano and Ishino, 1998) handled a problem in which an expert's judgement is crucial and proposed an interactive genetic algorithm using simulated breeding. Classical techniques, such as principal component analysis, linear discrimination (Gama and Brazdil, 1998), Fourier transformation, wavelet transformation (Mallet et. al. 1998), and so on, have often been applied to feature transformation. (Zupan et. al., 1998), however, approached feature transformation from another angle - via function decomposition.

1.2.8 Real-world Applications

(Zupan et. al., 1998) applied feature transformation to allocating housing loans. Their program discovered feature structures meaningful to a domain expert. (Vafaie and De Jong, 1998) described in detail their experiments of feature transformation with the face data. The number of features and error rates are

significantly reduced. (Bloedorn and Michalski, 1998) showed positive effects of feature selection, discretization and construction to two real-world problems: text categorization and natural scene interpretation. (Pudil and Novovičová, 1998) applied their method to a texture discrimination problem and a speech recognition problem and showed that the method works effectively even without the knowledge of underlying probability distribution. (Terano and Ishino, 1998) successfully applied a genetic algorithm to extract knowledge useful to deciding on promotion strategies of new oral care product by analyzing noisy questionnaire. (Seabra Lopes and Camarinha-Matos, 1998) extracted features that capture qualitative states of a time varying system from numerical sensor data over a period of time and applied them to robot diagnosis.

1.3 FUTURE WORK

Several chapters have made initial attempts to combine different learning algorithms or statistical methods, and to combine feature transformation and selection. Nevertheless, much work is needed to unify this currently still diversified field.

1.3.1 Data Categorization

- Many types of data exist in practice. Some types, such as boolean, nominal, and numerical, can be found in the UC Irvine data repository (Merz and Murphy, 1996). Others like structural, relational, temporal should also receive our equal attention in knowledge-discovery applications to real-world problems. Some data have dimensions (such as time, mass and pressure), but there is no notion of scale (such as nominal, ratio and interval) (Washio and Motoda, 1997). So far analyses made using the UCI data have not addressed these issues yet. We should use this type of knowledge to deepen our understanding of the underlying phenomena and exclude irrational combination when constructing new features. The attempt made by (Donoho and Rendell, 1998) shows a good starting point.
- We need to investigate relations between different data types and create compatible data technologies. By doing so, we will be able to apply the existing feature transformation and selection methods and handle various applications effectively and efficiently.
- We need to study the links between various techniques and data types. Pudil and Novovičová's work in this book presented their effort toward establishing such links. The issue is also studied in (Liu and Motoda, 1998). This work is necessary especially when we face an ever increasing number of choices of transformation and selection methods. A contingency theory will allow a naive user to make best use of the techniques available.

1.3.2 Perspectives from Other Disciplines

The use of feature transformation and subset selection is not confined only to data mining and machine learning. Other fields do have a similar problem

which may not be phrased as such. For example, the term of dimensionality reduction is more often used in statistics, pattern recognition (Wyse et. al., 1980), databases (Barbara et al., 1997). Switching circuit design in electrical engineering also implicitly addresses the issue of feature transformation. This collection has brought together different approaches and made some cross-paradigm connections. This is a good beginning. We should strive to continue our efforts in avoiding repeated research and in making maximum use of the already available techniques to help us solve real-world problems. By studying how other fields approach similar problems, we can expand our data mining repertoire of algorithms by equipping them with tools of feature transformation and subset selection.

1.3.3 Comprehensibility of Extracted/Constructed Features

Most of the efforts have been directed to improve performance such as estimated classification accuracy. In the case of data mining, however, we also need to pay attention to issues like comprehensibility of newly extracted or constructed features. People want to know not only whether the data contains valuable information, but also what the information is through discovered rules and features.

1.3.4 Better Use of Knowledge

Different kinds of knowledge, from complete to fragmentary knowledge, can all contribute to feature transformation and subset selection. In order to make best use of knowledge, we need to handle conflicts arising from using knowledge of different sources and to balance domain-specific knowledge and domain-independent bias. Many unsolved problems still await intelligent solutions.

1.3.5 Better Understanding of Human Behavior

Little work has been devoted to studying how human experts come up with good representations. We know little about how human experts relate primitive features to higher-level abstract intermediate concepts in real-world domains. It would be very useful if we could know more about involving human efforts in the process of developing good representations.

We are facing a grim reality of ever growing amounts of data and yet constrained by limitations of tools and representations. Nevertheless, we are determined to continue our pursuit of mining regularities from disordered data and of discovering knowledge hidden in massive data. In order to get more useful findings from the data, we need to first make the data less by removing its irrelevant parts. That is, less is more.

With less dimensionality, we can overcome some limitations imposed by best technologies: we can handle data that could not be dealt with before dimensionality reduction; we can discover knowledge that would have been lost in the massive data. With fewer features, we can focus better on the relevant data,

learn more sensible knowledge from the data; with simpler learned representations, we can understand more what has been learned from the data.

Feature transformation and subset selection are key techniques in answering the pressing needs for data mining. These techniques can help reduce data for mining or learning tasks and enable those mining algorithms, which were unable to mine, to mine.

References

- Aha, D.W. (1998). Feature Weighting for Lazy Learning Algorithms. *This volume*, pages 13–32.
- Baldoni, M., Baroglio, C., Cavagnino, D., and Saitta. L. (1998). Towards Automatic Fractal Feature Extraction for Image Recognition. *This volume*, pages 357–373.
- Barbara, D., DuMouchel, W., Faloutsos, C., Haas, P., Hellerstein, J., Ioannidis, Y., Jagadish, H., Johnson, T., Ng, R., Poosala, V., Ross, K., and Sevcik, K. (1997). The New Jersey data reduction report. *Bulletin of the Technical Committee on Data Engineering*, 20(4).
- Bloedorn, E. and Michalski, R.S. (1998). Data-driven Constructive Induction: Methodology and Applications. *This volume*, pages 51–68.
- Brachman, R.J., and Anand, T. (1996). The Process of Knowledge Discovery in Databases: A Human-centered Approach. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 37–57. AAAI Press / The MIT Press.
- Dash, M. and Liu, H. (1997). Feature Selection Methods for Classifications. *Intelligent Data Analysis: An International Journal*, 1(3).
- Donoho, S. and Rendell, L. (1998). Feature Construction Using Fragmentary Knowledge. *This volume*, pages 273–288.
- Gama, J. and Brazdil, P. (1998). Constructive Induction on Continuous Spaces. *This volume*, pages 289–303.
- Hu, Y. (1998). Constructive Induction: Covering Attribute Spectrum. *This volume*, pages 257–272.
- Kohavi, R. and John, G.H. (1998). The Wrapper Approach. *This volume*, pages 33–50.
- Blum, A.L. and Langley, P. (1997). Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, 97:246–271.
- Langley, P. (1996). *Elements of Machine Learning*. Morgan Kaufmann Publishers, Inc.
- Lavrač, N., Gamberger, D., and Turney, P. (1998). A Relevancy Filter for Constructive Induction. *This volume*, pages 137–154.
- Liu, H. and Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers.
- Mallet, Y., de Vel, O., and Coomans, D. (1998). Integrated Feature Extraction Using Adaptive Wavelets. *This volume*, pages 175–189.

- Matheus, C. (1991). The Need for Constructive Induction. In L.A. Birnbaum and Collins G.C., editors, *Machine Learning - Proceedings of the Eighth International Workshop*, pages 173–177, June.
- Mephu Nguifo, E. and Njiwoua, P. (1998). Using Lattice-based Framework as a Tool for Feature Extraction. *This volume*, pages 205–218.
- Merz, C. and Murphy, P. (1996). UCI repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>. Irvine, CA: University of California, Department of Information and Computer Science.
- Pazzani, M.J. (1998). Constructive Induction of Cartesian Product Attributes. *This volume*, pages 341–354.
- Perrin, P. and Petry, P. (1998). Lexical Contextual Relations for the Unsupervised Discovery of Texts Features. *This volume*, pages 157–173.
- Pudil, P. and Novovičová, J. (1998). Novel Methods for Subset Selection with Respect to Problem Knowledge. *This volume*, pages 101–116.
- Seabra Lopes, L. and Camarinha-Matos, L.M. (1998). Feature Transformation Strategies for a Robot Learning Problem. *This volume*, pages 375–391.
- Setiono, R. and Liu, H. (1998). Feature Extraction via Neural Networks. *This volume*, pages 191–204.
- Terano, T. and Ishino, Y. (1998). Interactive Genetic Algorithm Based Feature Selection and Its Application to Marketing Data Analysis. *This volume*, pages 393–406.
- Thornton, C.J. (1992). *Techniques of Computational Learning: an Introduction*. Chapman and Hall.
- Utgoff, P.E. and Precup, D. (1998). Constructive Function Approximation. *This volume*, pages 219–235.
- Vafaie, H. and De Jong, K. (1998). Evolutionary Feature Space Transformation. *This volume*, pages 307–323.
- Wang, H., Bell, D., and Murtagh, F. (1998). Relevance Approach to Feature Subset Selection. *This volume*, pages 85–99.
- Wang, K. and Sundaresan, S. (1998). Selecting Features by Vertical Compactness of Data. *This volume*, pages 71–84.
- Washio, T and Motoda, H. (1997). Discovering Admissible Models of Complex Systems Based on Scale-Types and Identity Constraints. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, pages 810–817.
- Wnek, J. and Michalski, R.S. (1994). Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, 14:139–168.
- Wyse, N., Dubes, R., and Jain, A.K. (1980). A Critical Evaluation of Intrinsic Dimensionality Algorithms. In Gelsema, E.S. and Kanal, L.N., editors, *Pattern Recognition in Practice*, pages 415–425. Morgan Kaufmann Publishers, Inc.
- Yang, J. and Honavar, V. (1998). Feature Subset Selection Using A Genetic Algorithm. *This volume*, pages 117–136.
- Zheng, Z. (1998) A Comparison of Constructing Different Types of New Feature for Decision Tree Learning. *This volume*, pages 239–255.
- Zupan, B., Bohanec, M., Demšar, J., and Bratko, I. (1998). Feature Transformation by Function Decomposition. *This volume*, pages 325–340.

2 FEATURE WEIGHTING FOR LAZY LEARNING ALGORITHMS

David W. Aha

Navy Center for Applied Research in Artificial Intelligence

Naval Research Laboratory, Code 5510

4555 Overlook Ave, SW

Washington, D.C. 20375-5337

www.aic.nrl.navy.mil/~aha

Abstract: Learning algorithms differ in the degree to which they process their inputs prior to their use in performance tasks. Many algorithms eagerly compile input samples and use only the compilations to make decisions. Others are lazy: they perform less precompilation and use the input samples to guide decision making. The performance of many lazy learners significantly degrades when samples are defined by features containing little or misleading information. Distinguishing feature relevance is a critical issue for these algorithms, and many solutions have been developed that assign weights to features. This chapter introduces a categorization framework for feature weighting approaches used in lazy similarity learners and briefly surveys some examples in each category.

2.1 INTRODUCTION

Lazy learning algorithms are machine learning algorithms (Mitchell, 1997) that are welcome members of procrastinators anonymous. Purely lazy learners typically display the following characteristics (Aha, 1997):

1. *Defer*: They delay the processing of their inputs until they receive requests for information; they simply store their inputs for future use.
2. *Demand-Driven*: They reply to information queries by combining information from their stored (e.g., training) samples.
3. *Discard*: They delete the constructed query and any intermediate results.

In contrast, *eager* algorithms greedily replace their inputs with an abstraction (e.g., a rule set, decision tree, or neural network) and use it to process queries.

Lazy learners have been designed to induce decision trees (e.g., (Smyth and Cunningham, 1995; Friedman et al., 1996)), rule sets (e.g., (Lachiche and Marquis, 1998)), Bayes classifiers (Kontkanen et al., 1998)), and improve speedup learning algorithms (e.g., (Borrado and Veloso, 1997)). However, the most frequently studied group of lazy learners are those that use similarity functions to answer queries. These include k -nearest neighbor classifiers and algorithms identified by names satisfying the following grammar:

{case,exemplar,instance,memory}-based {learning,reasoning}.

These *lazy similarity algorithms* are the focus of this chapter.

Eager learning algorithms assume their learning bias is appropriate for the performance task. When this assumption is correct, this can yield performance benefits (e.g., increased query response speed). However, there is a risk that this assumption is wrong, and that information lost during eager abstraction is crucial for generating accurate responses to queries. Thus, a key advantage of lazy algorithms is that they can respond to unanticipated queries in ways not available to all eager learners (Barsalou, 1983).

Lazy algorithms have three other computational advantages. First, they have small training costs (e.g., store a sample and update some indices), although this is frequently balanced by higher costs for generating predictions unless a fast indexing scheme is implemented. Small training costs make lazy algorithms particularly well suited for incremental learning tasks, where a data stream continually updates the set of input samples. Second, lazy problem solvers provide efficiency gains through solution reuse; they can store and adapt solutions for subsequent problems, which can greatly reduce problem solving effort. Finally, lazy algorithms can generate precedent explanations, which are preferable to abstract explanations for many tasks.

Only purists and teachers remain interested in purely lazy, standalone learners. Most research and practice with lazy learners involves some form of *caching*, which can be used to “fine tune” a lazy learner (e.g., by storing information on prediction quality) so as to improve its performance. In this chapter, we focus on methods for caching binary or continuous feature weights in the context of feature-value representations for samples. Feature weights are used to denote the relevance of features in similarity computations, allowing similarity functions to emphasize features according to their relevance, assuming accurate weight settings. When some features are irrelevant to the prediction task, learning accurate weight settings can greatly increase learning rates in lazy learners; both theoretical (Langley and Iba, 1993) and empirical analyses (Wettschereck, 1994) suggest that the complexity of lazy learning is exponential in the number of irrelevant features.

This chapter introduces a categorization framework for weighting features in lazy similarity learners, and summarizes some example algorithms. Research on this topic spans several decades and disciplines (e.g., data mining, cognitive psychology, statistics). We focus primarily on contributions to machine learning and case-based reasoning. Our framework complements rather than repeats the

framework presented in (Wetschereck et al., 1997), which is organized around a set of dimensions rather than a structured hierarchy.

2.2 FEATURE WEIGHTING IN A LAZY LEARNING ALGORITHM

This section defines the lazy similarity learning algorithm named *k*-nearest neighbor (*k*-NN) (Fix and Hodges, 1951) to exemplify the use of feature weights. We chose *k*-NN because it is well known, and many other lazy similarity learners can be described according to how they differ from *k*-NN. Some of *k*-NN's less relevant details are omitted. Although this section focuses on classification, we will later discuss *k*-NN variants for other types of tasks (e.g., retrieval, regression).

A *classifier* inputs a query \mathbf{q} and outputs a class prediction. The query and each *training sample* $\mathbf{x} = \{x_1, x_2, \dots, x_{|\mathbf{F}|}\}$ are points in a multidimensional space defined by a feature set \mathbf{F} . The class (or cluster) of a sample is $x_c \in \mathbf{J}$. We assume each feature is either continuous or discrete, although only for this presentation.

A classifier's performance objective is to minimize *expected loss*, or *misclassification risk* R_j , for each class $c_j \in \mathbf{J}$:

$$R_j = \sum_{c_{j'} \in \mathbf{J}} L_{c_j c_{j'}} p(c_{j'} | \mathbf{q}) \quad (2.1)$$

where $L_{c_j c_{j'}}$ is the loss (cost) associated with mistakenly classifying a sample of class c_j as in class $c_{j'}$ ($j \neq j'$), and $p(c_{j'} | \mathbf{q})$ is the probability of classifying \mathbf{q} in class $c_{j'}$. *k*-NN generally assumes that all misclassifications have equal cost:

$$L_{c_j c_{j'}} = \begin{cases} 0 & j = j' \\ 1 & j \neq j' \end{cases} \quad (2.2)$$

Since *k*-NN is not given \mathbf{q} 's class, it instead outputs the most probable class:

$$\text{k-NN}(\mathbf{q}) = \arg \max_{c_j \in \mathbf{J}} p(c_j | \mathbf{q}) \quad (2.3)$$

k-NN differs from other classifiers in how it defines posterior class probabilities:

$$p(c_j | \mathbf{q}) = \frac{\sum_{\mathbf{x} \in \mathbf{K}_{\mathbf{q}}} \begin{cases} K(d(\mathbf{x}, \mathbf{q})) & x_c = c_j \\ 0 & \text{otherwise} \end{cases}}{\sum_{\mathbf{x} \in \mathbf{K}_{\mathbf{q}}} K(d(\mathbf{x}, \mathbf{q}))} \quad (2.4)$$

where $K()$ is a *kernel* function, here defined as

$$K(d(\mathbf{x}, \mathbf{q})) = \frac{1}{d(\mathbf{x}, \mathbf{q})}, \quad (2.5)$$

and where $\mathbf{K}_{\mathbf{q}}$ is the set of \mathbf{q} 's nearest neighbors among a set \mathbf{X} of (previously supplied) *training samples* as determined by the distance function $d()$. That is,

k -NN computes the distance $d(\mathbf{x}, \mathbf{q})$ of \mathbf{q} to each $\mathbf{x} \in \mathbf{X}$ using:

$$d(\mathbf{x}, \mathbf{q}) = \left(\sum_{f \in \mathbf{F}} w(f) \cdot \delta(x_f, q_f)^r \right)^{\frac{1}{r}} \quad (2.6)$$

where k -NN defines $r = 2$ (i.e., Euclidean distance), function $\delta()$ defines how values of a given feature differ:

$$\delta(x_f, q_f) = \begin{cases} |x_f - q_f| & f \text{ is continuous} \\ 0 & f \text{ is discrete and } x_f = q_f \\ 1 & f \text{ is discrete and } x_f \neq q_f \end{cases} \quad (2.7)$$

and $w(f)$ defines the *feature weighting* function in k -NN, which is

$$w(f) = w_f \quad (2.8)$$

$$\forall_{f \in \mathbf{F}} w_f = s \quad (2.9)$$

for some scalar constant s .

Several variants of k -NN exist (Dasarathy, 1991; Aha, 1997). We will focus on alternatives to Equations 2.8 and 2.9 that assign unequal weights to features.

2.3 A CATEGORIZATION OF FEATURE WEIGHTING IN LAZY LEARNERS

2.3.1 Framework

Figure 2.1 displays our categorization framework for feature weighting methods in lazy similarity learners. Highlighting whether an oracle is available reinforces a lesson: exploit expert domain knowledge when it is available. Although this does not guarantee performance improvements, available experts should always be consulted. Furthermore, knowledge-poor approaches will sometimes fail without exploiting domain-specific knowledge (Cain et al., 1991).

Our emphasis is on automated, knowledge-poor approaches to feature weighting and selection (i.e., binary weights). This task can be viewed as a search process through a space of weight settings for a given sample space representation. Figure 2.2 summarizes this view; it corresponds to the *inferred* category shown in Figure 2.1, where less domain knowledge is assumed to be available. This framework emphasizes that search is guided by an evaluation function, which can be categorized by its task-dependent performance measures (e.g., classification accuracy, retrieval efficiency, problem solution quality, or a cost function) and the evaluation method.

Evaluation methods can be dichotomized by whether they use feedback from the performance task. Methods that do not use feedback are called *filters*, whereas methods that use the performance algorithm itself as the evaluation function are *wrappers* (John et al., 1994). Because most of the algorithms mentioned here use either a filter or wrapper model to set feature weights, we tend to ignore feedback methods that are not wrappers.

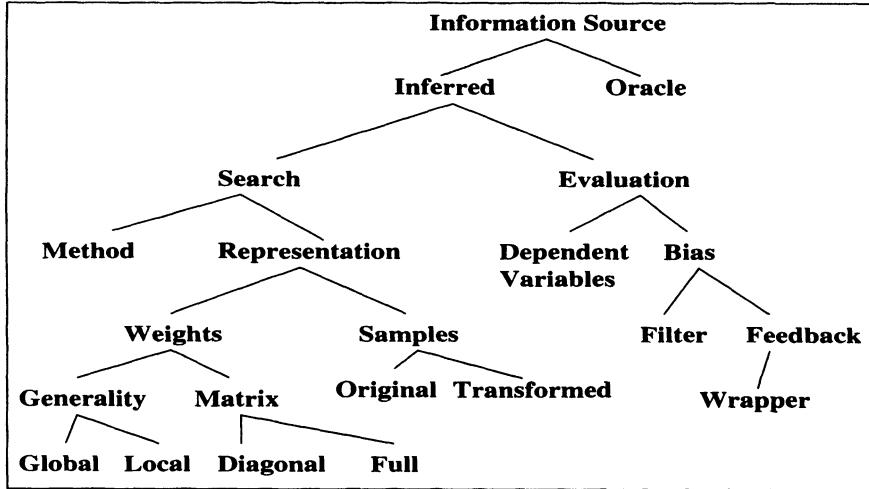


Figure 2.1 A Categorization Framework of Feature Weighting Methods for use in Lazy Similarity Learning Algorithms

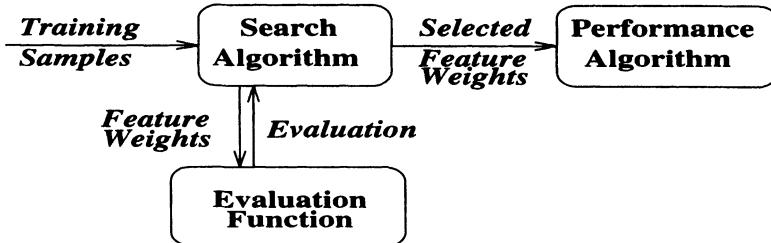


Figure 2.2 Automated Feature Weighting as a Search Task

2.3.2 Inferring Feature Weights

Automated feature weighting methods can be categorized by their search algorithm and evaluation function (Figure 2.1). We focus primarily on search functions, often using the evaluation function category to group algorithms with similar search functions.

Searching for Feature Weight Settings.

Search Algorithms

Some filter methods for weighting features do not perform search, including those that assign weights using simple probabilities (Creecy et al., 1992), mutual information gain (Daelemans and van den Bosch, 1992; Wettschereck et al., 1997), or value-specific differences in concept distributions (Stanfill and Waltz, 1986). However, the vast majority of feature weighting methods do perform search. This search is rarely exhaustive because the size of the weight space is

exponential in the number of features (i.e., $O(y^{|F|})$, where y is the number of different weight settings a feature can have and F is the feature set).

Search Algorithms used with Filter Models: Many filter methods for feature weighting use a hill-climbing search. For example, some induce a decision tree, where the features selected for use in similarity computations are those that remain in the tree after pruning. Cardie (Cardie, 1993) reported that this approach significantly increased classification accuracies for three natural language learning tasks, while Kibler and Aha (Kibler and Aha, 1987) reported more mixed results on two medical classification tasks.

Trott and Leng (Trott and Leng, 1997; Leng and Trott, 1997) instead used an *integer programming* (IP) approach to set feature weights for an interactive sample retrieval task, where users iteratively supply query \mathbf{q} 's feature values. In this task, the ordering of features in a stored sample influences the query input process. Leng and Trott's algorithm imposes a fixed ratio, for all samples, between the weight of the last feature and the sum of the weights of all other features in a sample. This ratio can be used to generate a set of linear equations, one per training sample, which can be solved using IP to generate feature weights. For highly engineered applications, this method could be used to remove the need to manually set feature weights for interactive k -NN retrieval algorithms. However, its evaluation has not yet been published.

Search Algorithms used with Wrapper Models: Many wrapper search methods use a variant of hill climbing. This includes the online weighting method in EACH (Salzberg, 1991), which updates feature weights by a fixed amount Δ after classifying each training sample. Correct classifications cause the weights of matching (mismatching) features to be incremented (decremented). Incorrect classifications cause the weights of mismatching (matching) features to be incremented (decremented). Thus, weights are continuously updated so as to locally maximize classification accuracy. By selecting good values for Δ , EACH (with $k = 1$) consistently improved classification performance in comparison with using Equation 2.9, although performance differences are also due to EACH's lazy replacement of input samples with hyperrectangles.

Salzberg's algorithm influenced the design of IB4 (Aha, 1992), which uses another online, hill-climbing wrapper to set feature weights:

$$w(f) = \max \left(\frac{\text{CumulativeWeight}_f}{\text{WeightNormalizer}_f} - 0.5, 0 \right), \quad (2.10)$$

where $\text{CumulativeWeight}_f$ is assumed to converge to half of $\text{WeightNormalizer}_f$ for irrelevant features. IB4 was designed for skewed concept distributions and to avoid the need for a fixed weight adjustment parameter. Let $\Lambda(\mathbf{x}, \mathbf{y}) = \max(p(x_c), p(y_c))$ (i.e., the higher a priori probability of \mathbf{x} and \mathbf{y} 's classes). Then $\text{CumulativeWeight}_f$ is incremented by

$$\text{CumulativeWeight}_f \doteq \begin{cases} 1 - \delta(x_f, y_f) \cdot (1 - \Lambda(\mathbf{x}, \mathbf{y})) & \text{if } (x_c = y_c) \\ \delta(x_f, y_f) \cdot (1 - \Lambda(\mathbf{x}, \mathbf{y})) & \text{Otherwise} \end{cases} \quad (2.11)$$

and WeightNormalizer is always incremented by $(1 - \Lambda(\mathbf{x}, \mathbf{y}))$. With $k = 1$, IB4 recorded good results vs. using Equation 2.9 on some tasks involving irrelevant features.

IB4 assumes a uniform distribution for irrelevant feature values. Kira and Rendell (Kira and Rendell, 1992) introduced the binary weighting algorithm RELIEF to remove this constraint. It iterates through a weight-updating procedure m times that (1) selects a random training sample \mathbf{x} , (2) locates \mathbf{x} 's most similar positive (\mathbf{p}) and negative (\mathbf{n}) training samples, and (3) updates each feature's weight using

$$w(f) = w(f) - \delta(x_f, p_f) + \delta(x_f, n_f) \quad (2.12)$$

During classification, weights that exceed a user-specified *relevance* threshold are mapped to 1, while those that do not are mapped to 0. Kira and Rendell reported good results for RELIEF on parity tasks. Kononenko (Kononenko, 1994) extended RELIEF for $k > 1$ and to work on multiclass tasks. Robnik-Šikonja and Kononenko extended it to work on regression tasks. Wettschereck et al. (Wettschereck et al., 1997) did not map RELIEF's continuous weights to binary values. Domingos (Domingos, 1997) introduced RC, an algorithm reminiscent of RELIEF. RC hill-climbs only if feature selection increases predictive accuracy, as guided by leave-one-out cross validation error (LOOCE) on the training set. In contrast, RELIEF climbs after classifying each sample.

Conjugate gradient, combined with LOOCE, has also been used to guide search for feature weights in lazy similarity learners. Lowe (Lowe, 1995) used this approach in the variable kernel similarity metric (VSM). It uses knowledge of the function's gradient to direct search, which can substantially increase learning rates when the target function is reasonably smooth. Error derivatives are used to guide the conjugate gradient procedure for each feature weight. The VSM performed as well as or better than several other algorithms on two data sets yet required far less training time than some other algorithms. Wettschereck (Wettschereck, 1994) investigated a simplification of the VSM named $k\text{-NN}_{VSM}$. After optimizing k , it uses conjugate gradient to optimize feature weights so as to minimize LOOCE training error. $k\text{-NN}_{VSM}$ can learn good feature weights in a variety of domains although its relatively large number of design decisions can heavily influence its performance. Wilke and Bergmann (Wilke and Bergmann, 1996) also used conjugate gradient with LOOCE error to search for feature weights. They tested this approach in two algorithms, KNNacc and KNN_{cost}, and reported substantial decision cost decreases for a credit scoring task.

Several parallel hill climbing algorithms for finding feature weight settings have been investigated. Kelly and Davis (Kelly and Davis, 1991) reported that a genetic algorithm (GA) search outperformed Equation 2.9 on three classification tasks. Wilson and Martinez (Wilson and Martinez, 1996) found that their GA outperformed unweighted k -NN on only half of 16 tasks, but significant accuracy increases often occurred for tasks with irrelevant features. Chang and Lippman's (Chang and Lippman, 1991) GA performed feature selection rather

than feature weighting. On one task, they compared their algorithm vs. sequential search algorithms (i.e., forward sequential search (FSS) and backwards sequential elimination (BSE)). Their GA approach reduced error rates using fewer than half the features selected by the other algorithms. Skalak (Skalak, 1994), using mutation operations on a single bit string, reported increased classification accuracies on three of four tasks while selecting approximately half the features per task. His algorithm simultaneously deleted at least 96% of the samples per task.

Aha and Bankert (Aha and Bankert, 1996) explored beam search variants of hill-climbing in sequential feature selection algorithms when $k = 1$. Their algorithm, which stochastically selects feature subsets to evaluate, significantly increased accuracies for some cloud classification data sets. Extensions of this research lead to using a distributed output representation with error-correcting capability, where feature selection is performed independently for each output bit (Ricci and Aha, 1998). This computationally expensive approach significantly increased accuracy on some data sets without significant decreases on any others.

Maron and Moore (Maron and Moore, 1997) described an innovative parallel feature selection approach named *schemata racing*. They noted that FSS and BSE are heavily biased towards selecting few or many features, respectively, and are slow when these biases are not met. In contrast, schemata racing hill climbs using *races*, simultaneously run for all features, to perform feature selection. Schemata racing iteratively selects a random training sample, randomly selects a subset of features, and computes the selected sample's LOOCE error. Error statistics are maintained for each feature. If the error when selecting (dropping) a feature is significantly higher than when dropping (selecting) it, that feature will be selected (dropped) in all future races. Racing continues until feature selections are finalized. On several synthetic tasks and four data sets, schemata racing reliably yields accuracies similar to those obtained by FSS and BSS, but often greatly reduces the computational costs of feature selection. Ricci and Aha (Ricci and Aha, 1998) reported that a variant of schemata racing performed well vs. no feature selection when using an error-correcting output representation with $k = 1$.

Some algorithms use best-first search to guide feature weighting. For example, CS-IBL (Tan, 1993) uses a novel approach for incremental feature selection. At each step, it evaluates the lowest-cost feature of the nearest neighbor, terminating when the nearest neighbor's distance is guaranteed to be smaller than any training sample from a competing class. On two tasks, CS-IBL reduced the number of feature evaluations by an order of magnitude, although it increased the total number of classification errors prior to convergence.

Representations of Weight and Sample Space

Most artificial intelligence research on feature weighting for lazy learners imposes constraining assumptions on the representations used for both weights

and samples. The following sections describe these constraints and mention example algorithms that violate them.

Weight Space: Figure 2.1 lists two categories for distinguishing the feature weight space: *generality* and *matrix*. We discuss these characteristics in turn.

The *generality* of the weight space denotes a weight's scope. Most algorithms that infer feature weight settings are *global*: their weights apply across the entire space. In contrast, *local* algorithms permit feature weights to vary in different parts of the sample space, and thus model a wider range of tasks. Local feature weighting methods include methods that weight features by *class*¹, *feature value*, and/or *sample space*. Feature value and sample space weighting methods can depend either on the training data or the query itself, which Atkeson et al. (Atkeson et al., 1997) refer to as *point-based* and *query-based* methods, respectively.

Creecy et al. (Creecy et al., 1992) introduced *per category feature importance* (PCF), which assigns feature weights using

$$w(f, c_j) = p(c_j | f) \quad (2.13)$$

Thus, the weight of feature f for class c_j is defined as the conditional probability that a sample \mathbf{x} is in c_j given x_f , averaged across all values for f . PCF assigns high weights to features that have high correlations with the given class. IB4 (Aha, 1992), mentioned previously, also infers class-specific weights. It uses a different weight-updating rate depending on class frequency. More recently, Howe and Cardie (Howe and Cardie, 1997) introduced CDW, a class-specific weighting algorithm for symbolic features that computes weights using

$$w(f, c_j) = \sum_{v \in f} |r(f, v, \{\mathbf{x} | \mathbf{x} \in c_j\}) - r(f, v, \{\mathbf{x} | \mathbf{x} \notin c_j\})| \quad (2.14)$$

where $r(f, v, X')$ is, for a training subset \mathbf{X}' , $P(x'_f = v | x' \in \mathbf{X}')$. CDW sets a feature's weight for a class c_j according to the degree to which its distributions of values for c_j differ from its distributions for other classes. CDW outperformed k -NN (i.e., for both $k = 1$ and $k = 10$) on some data sets where feature relevance varied per class, but significantly lowered accuracies for some other data sets. This lead Howe and Cardie to investigate EF-CDW, which transforms the sample representation to include one feature for every feature-value pair. This sample and value-specific algorithm generally outperformed CDW and compared well with unweighted k -NN.

The value-difference metric (Stanfill and Waltz, 1986), one of the first similarity algorithms designed for symbolic features, assigns feature weights according to a feature's value:

$$w(f, v) = \sqrt{\sum_{c_j \in \mathbf{J}} p(\mathbf{x} \in c_j | \mathbf{x} \in \mathbf{X}, x_f = v)^2} \quad (2.15)$$

This point-based algorithm performed well on a word pronunciation task, although it was not compared with unweighted k -NN. Ricci and Avesani's (Ricci and Avesani, 1995) AASM, designed for continuous features, assigns a feature f 's weight depending on whether $q_f > x_f$:

$$w(f, \mathbf{x}, \mathbf{q}) = \begin{cases} w_{x_f >} & \text{if } x_f \geq q_f \\ w_{x_f <} & \text{if } x_f < q_f \end{cases} \quad (2.16)$$

where $w_{x_f >}$ and $w_{x_f <}$ are the weights of sample \mathbf{x} at feature f on the respective "sides" of x_f . AASM is point-based because these weight settings are determined prior to being given a query. Ricci and Avesani reported favorable results for AASM as compared to k -NN ($k = 1$) and EACH (Salzberg, 1991) on four data sets.

Value-specific feature weighting approaches constrain weights to be identical for all samples with the same feature value. In contrast, sample-specific algorithms remove this constraint. Several sample-specific algorithms have been investigated. For example, Fukunaga and his colleagues (Short and Fukunaga, 1981; Fukunaga and Flick, 1982) described a query-based algorithm that computes its k nearest neighbors, and then computes each neighbor's local distance function to find the query's nearest neighbor. Local distance functions were computed by estimating the finite sample risk from the local neighborhood of a given sample, and then minimizing the difference between the finite sample risk and the asymptotic risk. Myles and Hand (Myles and Hand, 1990) extended this approach for multiclass tasks.

This local weighting approach is sensitive to noisy samples, and its many distinct local distance functions can obscure useful feature information. Therefore, Fukunaga and Flick (Fukunaga and Flick, 1984) proposed a global distance function that combines the information from all local functions into one global set of weights for a weighted Euclidean distance function. Similarly, Aha and Goldstone (Aha and Goldstone, 1992) combined local with global distance functions to compute sample-specific weights for GCM-ISW. It defines distance for numeric features using

$$d(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_{f \in \mathbf{F}} w(f, \mathbf{x}, \mathbf{q}) \cdot \delta(x_f, q_f)^2} \quad (2.17)$$

where sample-specific weights are dynamically assigned using

$$w(f, \mathbf{x}, \mathbf{q}) = w(f, \mathbf{x}) \cdot \sqrt{1 - |x_f - q_f|} + w(f) \cdot \left(1 - \sqrt{1 - |x_f - q_f|}\right) \quad (2.18)$$

Equation 2.18 adds the contributions of \mathbf{x} 's sample-specific weight $w(f, \mathbf{x})$ to the global weight $w(f)$. This distance function depends more on sample-specific weights when \mathbf{q} is similar to \mathbf{x} , and depends more on the global weights otherwise. Likewise, the updating algorithm for sample-specific weights has more influence when \mathbf{q} and \mathbf{x} are similar. GCM-ISW correlated significantly better with subject data than did its non-weighting and global weighting variants for

a categorization task where feature relevance varied in different parts of the sample space.

Domingos's (Domingos, 1997) RC algorithm uses another sample-specific feature weighting algorithm. It drops features from \mathbf{x} if they (1) differ from \mathbf{x} 's nearest sample and (2) removing them does not decrease overall LOOCE error. Using k -NN with $k = 1$, RC easily outperformed FSS and BSE on 24 data sets, and its benefits increased with increasing context dependency of feature relevance.

Bonzano et al. (Bonzano et al., 1997) describe a sample-specific weighting algorithm ISAC for problem solving tasks. Like RELIEF (Kira and Rendell, 1992; Kononenko, 1994) and RC (Domingos, 1997), ISAC randomly selects a training sample \mathbf{x} and updates feature weights based on \mathbf{x} 's differences with similar samples. However, RELIEF learns only global weights, RC is limited to binary weights, and both use only the nearest positive and negative samples to update feature weights for classification tasks. In contrast, ISAC learns continuous weight settings using the k -nearest neighbors. Weight updates are based on whether a neighbor's solution satisfies the selected sample's problem. Positive updates are made for \mathbf{x} using

$$w_i(\mathbf{x}) = w_i(\mathbf{x}) + \text{Correct}_{\mathbf{x}} / \text{Incorrect}_{\mathbf{x}} \quad (2.19)$$

where $\text{Correct}_{\mathbf{x}}$ ($\text{Incorrect}_{\mathbf{x}}$) is the number of times that \mathbf{x} has been correctly (incorrectly) retrieved. Negative updates decrement weights by the same ratio. Failure-driven weight updating was more valuable than success-driven updating for their air traffic control task, and weighting was more appropriate for training sets that contain redundant information.

Friedman (Friedman, 1994) described an alternative approach for local feature weighting that integrates strategies of lazy similarity learning with decision tree induction. His SCYTHE algorithm recursively zooms in on query \mathbf{q} along the most relevant feature, where local feature relevance is estimated from the estimated reduction in prediction error given that feature's value. The most relevant feature is scaled at each recursive step such that a fixed fraction of the training samples fall outside of a predetermined range around \mathbf{q} . After discarding these samples, the new "most relevant" feature is determined, and the process is repeated until only k training samples remain. SCYTHE performed well in comparison to k -NN on several classification tasks.

Cardie and Howe's (Cardie and Howe, 1997) sample-specific weighting approach induces a pruned decision tree from the training set to perform feature selection; features in the path that classify \mathbf{q} are selected. Continuous weights are then assigned to these features according to their entropy on the entire training set. Their algorithm performed well, in the context of minority class predictions, on a set of natural language learning tasks in comparison with unweighted k -NN.

The weight *matrix* of a feature weighting algorithm characterizes the types of weight settings that it can infer, where the matrix's two dimensions are the

sample space's features. Using no weights corresponds to a *diagonal* matrix whose values are all the same positive constant. Simple weighting methods also use a diagonal matrix, but allow weights to vary (i.e., feature selection approaches assign binary values while feature weighting approaches assign continuous values on the diagonal), where larger weights denote greater relevance.

Searching a continuous weight space does not always yield settings superior to feature selection. Kohavi et al. (Kohavi et al., 1997) showed that, for some data sets, increasing the number of feature weight settings beyond two yields few accuracy benefits, and can sometimes degrade performance. Their DIET algorithm uses a best-first search, guided by 10-fold cross validation, in a weight space where each feature has a weight in $\{0, 1/v, 2/v, \dots, (v-1)/v, 1\}$. Starting with "middle" weight settings, DIET explores all variants that increase or decrease each weight by $1/v$ (i.e., minding the boundaries). The best-performing weight set is used as the next starting point. DIET stops when continued search does not seem promising. On tests with synthesized tasks and several data sets, DIET often performed better using smaller values for v , which is evidence that over fitting can occur when using larger values for v .

Using diagonal matrices assumes that the performance task can be best modelled by stretching and shrinking the sample space along the individual feature dimensions. This prevents the representation of feature interactions, where some combination might prove particularly relevant. In contrast, a *full* matrix, where any matrix element can have a non-zero entry, allows arbitrary scaling of the sample space (Atkeson et al., 1997; Friedman, 1994). Mohri and Tanaka (Mohri and Tanaka, 1994) describe how to use full matrices with symbolic features, while Hastie and Tibshirani (Hastie and Tibshirani, 1996) describe local weighting methods that use full matrices. Weights need not be constants, but could be a polynomial or any arbitrarily complex function (e.g., a connectionist network).

Sample Space: Using full matrices relates to transforming the sample space. Algorithms that simply assign feature weights are usually insensitive to interacting or correlated features. In contrast, *principal components analysis* (PCA) can transform a sample space, re-representing it with a list of polynomial equations of the original features, ordered by nonincreasing variance. However, PCA's bias is not always appropriate; features with low variance might actually have high predictive relevance.

QM2Y (Mohri and Tanaka, 1994) is a lazy similarity learner that attempts to solve this deficiency in PCA. It assigns the absolute values computed by *Quantification Method II* (QM2) (Hayashi, 1952) to set its feature weights in the following variant of Equation 2.6, where symbolic features with V_f values are replaced with $|V_f|$ binary features before transformation:

$$d(\mathbf{x}, \mathbf{q}) = \sum_{f' \in \mathbf{F}'} \left(\sum_{f \in \mathbf{F}} w(f', f) x_f \cdot \sum_{f \in \mathbf{F}} w(f', f) q_f \right)^2 \quad (2.20)$$

where \mathbf{F}' is the set of transformed (i.e., new) features and $w(f', f)$ is the weight of the given feature f for the transformed feature f' . QM2 locates a new set of features such that the sums of the squared distances of each sample to its projections on the successive feature dimensions are minimized. After transformation, data dimensionality is reduced by removing the transformed features with lowest variation. New feature values are computed using

$$x'_f = \sum_{f \in \mathbf{F}} w(f', f) x_f \quad (2.21)$$

where these weights are calculated to maximize the ratio of the variance between each class's samples to the variance of all samples.

QM2Y performed best among the set of lazy similarity algorithms tested by Mohri and Tanaka, but hypotheses explaining why have not yet been published. We briefly discuss knowledge-intensive similarity learners that transform the sample space in Section 2.3.3.

Evaluation Function.

Dependent Variables

Most feature weighting methods for lazy similarity learners have been developed for classification tasks, and where classification accuracy was the dominant dependent variable. Other variables exist, including those concerning memory requirements and speed. Also, other contexts exist, such as retrieval, where measures including precision and recall are more relevant.

An important and frequently overlooked dependent variable is *cost* (Turney, 1995). For some tasks, some types of prediction errors are more costly than others, which violates Equation 2.2. Wilke and Bergmann's (Wilke and Bergmann, 1996) algorithm KNN_{COST} defines the cost of a decision as

$$\sum_{c_i \in J} \sum_{c_j \in J} P_{ij} C_{ij} \quad (2.22)$$

where P_{ij} and C_{ij} are entries in a confusion matrix and a cost matrix, respectively. Using a conjugate gradient search procedure to assign weight settings for a weighted-distance variant of k -NN (with $k = 11$) on a classification task, they reported substantial reductions in cost for KNN_{COST} in comparison to an accuracy-optimizing weighting approach and a non-weighting variant.

Features can have evaluation costs. Tan (Tan, 1993) defined feature costs for a mobile robot's actions as an increasing function of their duration. In CS-IBL, the first stage selects the sample that maximizes the ratio of expected match success to feature cost:

$$\frac{\prod_{f \in \mathbf{F}'} P_{if}}{\sum_{f \in \mathbf{F}'} C(f) \prod_{k=1}^{f-1} (1 - P_{ik})} \quad (2.23)$$

where \mathbf{F}' is the set of features whose values are known for a stored sample \mathbf{x} but not the query \mathbf{q} , P_{if} is the probability that feature f has the value x_i among training samples, and $C(f)$ is the cost incurred for evaluating f .

Smyth and Cunningham (Smyth and Cunningham, 1995) investigated the situation where features were free or had a fixed non-zero cost. Their NODAL_{cbr} algorithm integrates a lazy decision tree approach (Friedman et al., 1996) with a lazy similarity learner. Given \mathbf{q} , NODAL_{cbr} computes a set of similar samples using \mathbf{q} 's free features. An information theoretic criterion then determines the most informative non-free feature whose value, if known, would maximally reduce the remaining samples. Reminiscent of SCYTHE (Friedman, 1994), this step recurses until a most similar (rather than k) sample remains. NODAL_{cbr} consistently reduced the number of feature evaluations needed to retrieve correct solutions for two tasks.

NODAL_{cbr}, Trott and Leng's (Trott and Leng, 1997) IP algorithm, and ISAC (Bonzano et al., 1997) perform case retrieval, but none address the context of planning. Muñoz-Avila and Hüllen (Muñoz Avila and Hüllen, 1996) describe one weighting method for planning tasks, where weights, based on an algorithm similar to the one used in EACH (Salzberg, 1991), denote sample-specific relative feature relevance. Over a sequence of problem solving episodes, their weighting model increased the reliability that the retrieved sample solves the query problem (for two tasks). In particular, using their weighting method greatly reduced both the number of retrieved samples and solution failures in comparison with the same algorithm when not learning weights.

Evaluation Bias

Feature weighting algorithms that use an evaluation function other than the performance function are susceptible to bias mismatches between these functions, which can degrade performance (John et al., 1994). Thus John et al. advocate using wrapper rather than filter models (e.g., to set feature weights). Some evidence suggests that wrapper models are superior to filter models in this context (e.g., (Wettschereck et al., 1997)) when the dependent variable is classification accuracy. However, wrapper models are often more computationally expensive (i.e., when they evaluate a large number of feature weight settings). Filter models and efficient variants of wrapper models should be considered for tasks where computational expense is a concern.

2.3.3 Using Domain-Specific Information to Weight Features

Several researchers have reported benefits from using domain-specific information to assign feature weights (i.e., when an *oracle* is available). This is commonly done in case-based reasoning applications (e.g., (Shimazu et al., 1994)).

Domain-specific knowledge can usefully guide transformations of the sample space. For example, IB3-CI (Aha, 1991) induces feature combinations using knowledge of the target domain to bias its search. It significantly increased k -NN classification accuracy on one task. Turney instead examined the intro-

duction of a separate set of *contextual* features \mathbf{F}' , disjoint from the initial set \mathbf{F} . He used these to transform each sample \mathbf{x} into a sample \mathbf{y} :

$$y_i = (x_i - \mu_i(\mathbf{F}'))/\sigma_i(\mathbf{F}') \quad (2.24)$$

where $\mu_i(\mathbf{F}')$ is the expected value and $\sigma_i(\mathbf{F}')$ is the expected variation of x_i . Turney reported impressive accuracy increases when using variants of this weighting procedure on two tasks.

Domain-specific knowledge can also be used to select features directly. ROBBIE (Fox and Leake, 1995), which modifies the set of features used to retrieve samples using a k -NN variant, improved retrieval efficiency for a navigation task. Barletta and Mark (Barletta and Mark, 1988) described how an explanation-based learning (EBL) method can be used to select features for a lazy learner, while Cain et al. (Cain et al., 1991) used EBL to select the relevant features for each sample. Their lazy similarity algorithm uses these features to bias classification predictions.

2.4 ADDITIONAL CONTRIBUTIONS

2.4.1 Formal Contributions

Formal analyses that address feature weighting in lazy similarity learners include Satoh and Okamoto's (Satoh and Okamoto, 1994) PAC-learning analysis for weighting features, and Ling and Wang's (Ling and Wang, 1997) algorithm for computing optimal weights for a restricted set of classification tasks, given knowledge of the concept boundary's location. Their approach can be used to evaluate the performance of heuristic methods for learning feature weight settings on some synthesized classification tasks. Also, Griffiths and Bridge (Griffiths and Bridge, 1997) describe PAC analyses for VS-CBR, a feature selecting variant of k -NN that selectively retains samples. They stress the importance of defining VS-CBR's hypothesis space, and of analysing its constituent algorithms.

2.4.2 Other Roles for Weights in Lazy Learners

A “weighted” lazy learner does not necessarily refer to feature weighting. For example, distance-weighted variants of k -NN, which assign relevance weights to samples as a function of their similarity to a query, have been studied for several years (Dudani, 1975). Related algorithms (e.g., EACH (Salzberg, 1991) assign weights to samples based on their recorded performance in prediction tasks. See (Atkeson et al., 1997; Wettschereck et al., 1997) for more discussion on this topic.

Hastings et al. (Hastings et al., 1995) show how weights can help adapt solutions from those retrieved by CARMA, a lazy similarity algorithm for predicting rangeland pest consumption. It uses a hill-climbing algorithm that varies weights to minimize root mean squared error. CARMA can learn these weights in one of two modes: one set of weights for the entire training set or one set per sample. These weights have helped to improve CARMA's performance.

2.4.3 Avoiding Feature Selection

Several methods essentially remove the need for assigning feature weights. For example, Güvenir and his colleagues (e.g., (Güvenir and Sirin, 1996)) recommend voting among the individual features' matches. This avoids the need to combine the contributions of different features in a similarity function. Alternatively, feature transformation can obviate the need for feature weighting. For example, Chang and Lippman's (Chang and Lippman, 1991) GA recursively searches for constrained polynomial combinations of features. It reduced k -NN's error from 19% to zero on one task by replacing the original representation with a single feature. Finally, Datta and Kibler (Datta and Kibler, 1997) argued that, for some similarity functions, a feature's range of values determines a feature's impact on similarity computations.

2.5 SUMMARY

This chapter introduced a categorization framework for feature weighting algorithms in lazy similarity learners. Our focus was on the search algorithms used in automated weighting methods, although algorithms exemplifying other categories of this framework were discussed. This chapter contrasts with the framework presented in (Wettschereck et al., 1997), which focused on a set of unrelated dimensions rather than a categorization hierarchy. We did not discuss relations to feature weighting approaches used in other types of learning algorithms. Additional information on feature weighting in lazy learners can be found in (Dasarathy, 1991) and (Aha, 1997).

Although many more contributions on feature weighting for lazy similarity learners have been published (e.g., with similarity functions other than those based on Minkowski metrics), space constraints prevented their discussion. Also, our categorization framework is incomplete, and several dimensions for distinguishing these algorithms were not flushed out (e.g., offline vs. online algorithms, sensitivity to solution cluster distributions), but instead were briefly mentioned in the context of specific algorithms.

Several issues are good targets for further research. These include using full weight matrices, performance tasks other than classification, average case analyses, and examining how cognitively plausible categorization algorithms can impact the design of computationally effective lazy learners.

Acknowledgments

This research was supported by the Office of Naval Research.

Notes

1. Or training cluster, for tasks other than classification.

References

- Aha, D. W. (1991). Incremental constructive induction: An instance-based approach. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 117–121, Evanston, IL. Morgan Kaufmann.
- Aha, D. W. (1992). Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36:267–287.
- Aha, D. W., editor (1997). *Lazy Learning*. Kluwer, Norwell, MA.
- Aha, D. W. and Bankert, R. L. (1996). A comparative evaluation of sequential feature selection algorithms. In Fisher, D. and Lenz, J.-H., editors, *Artificial Intelligence and Statistics V*. Springer, New York.
- Aha, D. W. and Goldstone, R. L. (1992). Concept learning and flexible weighting. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 534–539, Bloomington, IN. Lawrence Erlbaum.
- Atkeson, C., Moore, A., and Schaal, S. (1997). Locally weighted learning. *AI Review*, 11:11–73.
- Barletta, R. and Mark, W. (1988). Explanation-based indexing of cases. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 541–546, St. Paul, MN. Morgan Kaufmann.
- Barsalou, L. W. (1983). Ad hoc categories. *Memory & Cognition*, 11:211–227.
- Bonzano, A., Cunningham, P., and Smyth, B. (1997). Using introspective learning to improve retrieval in CBR: A case study in air traffic control. In *Proceedings of the Second ICCBR Conference*, pages 291–302, Providence, RI. Springer.
- Borrajo, D. and Veloso, M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review*, 11:371–405.
- Cain, T., Pazzani, M. J., and Silverstein, G. (1991). Using domain knowledge to influence similarity judgement. In *Proceedings of a Case-Based Reasoning Workshop*, pages 191–202, Washington, DC. Morgan Kaufmann.
- Cardie, C. (1993). Using decision trees to improve case-based learning. In *Proceedings of the Tenth ICML*, pages 25–32, Amherst, MA. Morgan Kaufmann.
- Cardie, C. and Howe, N. (1997). Improving minority-class prediction using case-specific feature weights. In *Proceedings of the Fourteenth ICML*, pages 57–65, Nashville, TN. Morgan Kaufmann.
- Chang, E. I. and Lippman, R. P. (1991). Using genetic algorithms to improve pattern classification performance. In Lippman, R., Moody, J., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, Denver, CO.
- Creecy, R. H., Masand, B. M., Smith, S. J., and Waltz, D. L. (1992). Trading mips and memory for knowledge engineering. *CACM*, 35:48–64.
- Daelemans, W. and van den Bosch, A. (1992). Generalization performance of backpropagation learning on a syllabification task. In *Proceedings of TWLT3: Connectionism and Natural Language Processing*, pages 27–37, Enschede, The Netherlands. Unpublished.

- Dasarathy, B. V. (1991). *Nearest neighbor(NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, Los Alamitos, CA.
- Datta, P. and Kibler, D. (1997). Learning symbolic prototypes. In *Proceedings of the Fourteenth ICML*, pages 158–166, Tahoe City, CA. Morgan Kaufmann.
- Domingos, P. (1997). Context-sensitive feature selection for lazy learners. *AI Review*, 11:227–253.
- Dudani, S. (1975). The distance-weighted k-nearest neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:325–327.
- Fix, E. and J. L. Hodges, Jr. (1951). Discriminatory analysis, nonparametric discrimination, consistency properties. Technical Report 4, United States Air Force, School of Aviation Medicine.
- Fox, S. and Leake, D. L. (1995). Using introspective reasoning to refine indexing. In *Proceedings of the Fourteenth IJCAI*, pages 391–397, Montreal. Morgan Kaufmann.
- Friedman, J. H. (1994). Flexible metric nearest neighbor classification. Technical report, Stanford University, Department of Statistics.
- Friedman, J. H., Kohavi, R., and Yun, Y. (1996). Lazy decision trees. In *Proceeding of the Thirteenth National Conference on Artificial Intelligence*, pages 717–724, Portland, OR. AAAI Press.
- Fukunaga, K. and Flick, T. (1982). A parametrically-defined nearest neighbor distance measure. *Pattern Recognition Letters*, 1:3–5.
- Fukunaga, K. and Flick, T. (1984). An optimal global nearest neighbor metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:314–318.
- Griffiths, A. G. and Bridge, D. G. (1997). PAC analyses of a ‘similarity learning’ IBL algorithm. In *Proceedings of the Second ICCBR Conference*, pages 445–454, Providence, RI. Springer.
- Güvenir, H. A. and Sirin, I. (1996). Classification by feature partitioning. *Machine Learning*, 23:47–68.
- Hastie, T. and Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Pattern Analysis and Machine Intelligence*, 18:607–616.
- Hastings, J., Branting, L. K., and Lockwood, J. A. (1995). Case adaptation using an incomplete causal model. In *Proceedings of the First ICCBR Conference*, pages 181–192, Sesimbra, Portugal. Springer.
- Howe, N. and Cardie, C. (1997). Examining locally varying weights for nearest neighbor algorithms. In *Proceedings of the Second ICCBR Conference*, pages 455–466, Providence, RI. Springer.
- J. D. Kelly, Jr. and Davis, L. (1991). A hybrid genetic algorithm for classification. In *Proceedings of the Twelfth IJCAI*, pages 645–650, Sydney, Australia. Morgan Kaufmann.
- John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh ICML*, pages 121–129, New Brunswick, NJ. Morgan Kaufmann.

- Kibler, D. and Aha, D. W. (1987). Learning representative exemplars of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 24–30, Irvine, CA. Morgan Kaufmann.
- Kira, K. and Rendell, L. A. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth IJCAI*, pages 129–134, San Jose, CA. AAAI Press.
- Kohavi, R., Langley, P., and Yun, Y. (1997). The utility of feature weighting in nearest neighbor algorithms. In van Someren, M. and Widmer, G., editors, *Poster Papers: Ninth ECML*. Unpublished, Prague, Czech Republic.
- Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief. In *Proceedings of the Seventh ECML*, pages 171–182, Catania, Italy. Springer.
- Kontkanen, P., Myllymaki, P., Silander, T., and Tirri, H. (1998). Bayes optimal instance-based learning. In *Proceedings of the Tenth ECML*, Chemnitz, Germany. Springer.
- Lachiche, N. and Marquis, P. (1998). Scope classification: An instance-based learning algorithm with a rule-based characterization. In *Proceedings of the Tenth ECML*, Chemnitz, Germany. Springer.
- Langley, P. and Iba, W. (1993). Average-case analysis of a nearest neighbor algorithm. In *Proceedings of the Thirteenth IJCAI*, pages 889–894, Chambery, France. Morgan Kaufmann.
- Leng, B. and Trott, J. (1997). Automatically tuning question weights in CBR Express 2.x. Unpublished Manuscript.
- Ling, X. C. and Wang, H. (1997). Towards optimal weights setting for the 1-nearest neighbour learning algorithm. *AI Review*, 11:255–272.
- Lowe, D. (1995). Similarity metric learning for a variable-kernal classifier. *Neural Computation*, 7:72–85.
- Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *AI Review*, 11:192–225.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York.
- Mohri, T. and Tanaka, H. (1994). An optimal weighting criterion of case indexing for both numeric and symbolic attributes. In Aha, D. W., editor, *Case-Based Reasoning: Papers from the 1994 Workshop*. AAAI Press, Menlo Park, CA.
- Myles, J. and Hand, D. (1990). The multi-class metric problem in nearest neighbor discrimination rules. *Pattern Recognition*, 23:1291–1297.
- Muñoz Avila, H. M. and Hüllen, J. (1996). Feature weighting by explaining case-based planning episodes. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 280–294, Lausanne, Switzerland. Springer.
- Ricci, F. and Aha, D. W. (1998). Error-correcting output codes for local learners. In *Proceedings of the Tenth ECML*, Chemnitz, Germany. Springer.
- Ricci, F. and Avesani, P. (1995). Learning a local similarity metric for case-based reasoning. In *Proceedings of the First ICCBR Conference*, pages 301–312, Sesimbra, Portugal. Springer.

- Salzberg, S. L. (1991). A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276.
- Satoh, K. and Okamoto, S. (1994). Toward PAC-learning of weights from qualitative distance information. In Aha, D. W., editor, *Case-Based Reasoning: Papers from the 1994 Workshop*. AAAI Press, Menlo Park, CA.
- Shimazu, H., Shibata, A., and Nihei, K. (1994). Case-based retrieval interface adapted to customer-initiated dialogues in help desk operations. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 513–518, Seattle, WA. AAAI Press.
- Short, R. and Fukunaga, K. (1981). The optimal distance measure for nearest neighbor classification. *IEEE Transactions on Information Theory*, 27:622–627.
- Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh ICML*, pages 293–301, New Brunswick, NJ. Morgan Kaufmann.
- Smyth, B. and Cunningham, P. (1995). A comparison of incremental case-based reasoning and inductive learning. In Hatton, J. P. and Keane, M., editors, *Advances in Case-Based Reasoning*. Springer, Berlin.
- Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *CACM*, 29:1213–1228.
- Tan, M. (1993). Cost-sensitive learning of classification knowledge and its application in robotics. *Machine Learning*, 13:7–34.
- Trott, J. R. and Leng, B. (1997). An engineering approach for troubleshooting case bases. In *Proceedings of the Second ICCBR Conference*, pages 178–189, Providence, RI. Springer.
- Turney, P. D. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409.
- Wettschereck, D. (1994). A study of distance-based machine learning algorithms. Doctoral dissertation, Oregon State University, Department of Computer Science.
- Wettschereck, D., Aha, D. W., and Mohri, T. (1997). A review and empirical comparison of feature weighting methods for a class of lazy learning algorithms. *AI Review*, 11:273–314.
- Wilke, W. and Bergmann, R. (1996). Considering decision costs while learning of feature weights. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 460–472, Lausanne, Switzerland. Springer.
- Wilson, D. R. and Martinez, T. R. (1996). Instance-based learning with genetically derived attribute weights. In *Proceedings of the International Conference on Artificial Intelligence, Expert Systems, and Neural Networks*, pages 11–14. Unpublished.

3 THE WRAPPER APPROACH

Ron Kohavi¹ and George H. John²

¹Data Mining and Visualization
Silicon Graphics, Inc.
2011 N. Shoreline Boulevard,
Mountain View, CA 94043
ronnyk@sgi.com
robotics.stanford.edu/~ronnyk

²Data Mining,
Epiphany Marketing Software,
2141 Landings Drive
Mountain View, CA 94043
gjohn@cs.stanford.edu
robotics.stanford.edu/~gjohn

Abstract: In the feature subset selection problem, a learning algorithm is faced with the problem of selecting a relevant subset of features upon which to focus its attention, while ignoring the rest. To achieve the best possible performance with a particular learning algorithm on a particular training set, a feature subset selection method should consider how the algorithm and the training set interact. We explore the relation between optimal feature subset selection and relevance. The wrapper method searches for an optimal feature subset tailored to a particular algorithm and a domain. We compare the wrapper approach to induction without feature subset selection and to Relief, a filter approach to feature subset selection. Improvement in accuracy is achieved for some datasets for the two families of induction algorithms used: decision trees and Naive-Bayes. In addition, the feature subsets selected by the wrapper are significantly smaller than the original subsets used by the learning algorithms, thus producing more comprehensible models.

3.1 INTRODUCTION

In supervised machine learning, an induction algorithm is typically presented with a set of training instances, where each instance is described by a vector of

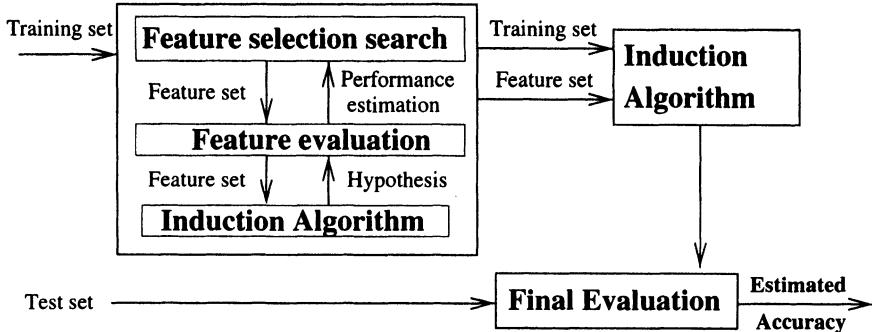
feature (or attribute) values and a class label. For example, in medical diagnosis problems the features might include the age, weight, and blood pressure of a patient, and the class label might indicate whether or not a physician determined that the patient was suffering from heart disease. The task of the induction algorithm, or the *inducer*, is to induce from training data a *classifier* that will be useful in classifying future cases. The classifier is a mapping from the space of feature values to the set of class values.

In the feature subset selection problem, a learning algorithm is faced with the problem of selecting some subset of features upon which to focus its attention, while ignoring the rest. The idea behind the wrapper approach (John, Kohavi and Pfleger, 1994; Kohavi and John, 1997), shown in Figure 3.1, is simple: the induction algorithm is used as a black box. It is repeatedly run on the dataset using various feature subsets. Some method is used to evaluate its performance on each subset, and the feature subset with the highest evaluation is chosen as the final set on which to run the induction algorithm. The resulting classifier is then evaluated on an independent test set that was *not* used during the search. Since the typical goal of supervised learning algorithms is to maximize classification accuracy on an unseen test set, we use accuracy as our metric in guiding the feature subset selection.

Practical machine learning algorithms—decision tree algorithms such as C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984), and instance-based algorithms such as IBL (Aha et al., 1991)—are known to degrade in prediction accuracy when trained on data containing superfluous features. Algorithms such as Naive-Bayes (Duda and Hart, 1973; Good, 1965; Domingos and Pazzani, 1997) are robust with respect to irrelevant features, but their performance degrades when correlated (even if relevant) features are added. John (1997) shows examples where adding a single irrelevant feature to a credit-approval or diabetes dataset degrades the performance of C4.5 by over 5%. The problem of feature subset selection is that of finding a subset of the original features of a dataset, such that an induction algorithm that is run on data containing only these features generates a classifier with the highest possible accuracy.

From a purely theoretical standpoint, the question of which features to use may not be of much interest. A Bayes rule predicts the most probable class for a given instance, based on the full joint probability distribution over the features and the class. The Bayes rule is monotonic, *i.e.*, adding features cannot decrease accuracy, so subset selection is useless.

In practical learning scenarios, however, we are faced with two problems. First, the learning algorithms are not given access to the underlying distribution; rather, they are usually given a relatively small training set. Second, even quite similar algorithms may incorporate different heuristics to aid in quickly building models of the training data—finding the smallest model consistent with the data is NP-hard in many cases (Hyafil and Rivest, 1976; Blum and Rivest, 1992).



The induction algorithm itself is used as a “black box” by the subset selection algorithm.

Figure 3.1 The wrapper approach to feature subset selection

We define an optimal feature subset with respect to a particular induction algorithm and dataset, taking into account the algorithm’s biases and their interaction with the training sample:

Definition 1

Given an inducer \mathcal{I} , and a training dataset \mathcal{D} with features X_1, X_2, \dots, X_n , an optimal feature subset, \mathbf{X}_{opt} , is the subset of the features that maximizes the accuracy of the induced classifier $\mathcal{C} = \mathcal{I}(\mathcal{D})$.

Thus, by definition, to get the highest possible accuracy, the best subset that a feature subset selection algorithm can select is an optimal feature subset.

This chapter is organized as follows. In Section 3.2, we present definitions of relevance and distinguish between relevance and optimality. In Section 3.3 we describe filter methods of feature subset selection. Section 3.4 describes an algorithm based on the wrapper approach, comparing the algorithm empirically with a filter algorithm. Related and future work are discussed in Sections 3.5 and 3.6, and we present our conclusions in Section 3.7.

3.2 RELEVANCE OF FEATURES

In this section, we define two degrees of relevance: weak and strong. These and several earlier definitions of relevance are reviewed in Kohavi and John (1997). We define relevance in terms of a Bayes rule. A feature X is **strongly relevant** if removal of X alone will result in performance deterioration of an optimal Bayes rule. A feature X is **weakly relevant** if it is not strongly relevant, but in some contexts may contribute to prediction accuracy of an optimal Bayes rule. A feature is **relevant** if it is either weakly relevant or strongly relevant; otherwise, it is **irrelevant**.

Definition 2 (Strong relevance)

*Let S be the set of all features, and let S_i be $S - \{X_i\}$. A feature X_i is **strongly relevant** iff there exist values x_i , y , and s_i for which $p(X_i = x_i, S_i = s_i) > 0$*



Figure 3.2 The feature filter approach, in which the features are filtered independently of the induction algorithm.

such that

$$p(Y = y \mid X_i = x_i, S_i = s_i) \neq p(Y = y \mid S_i = s_i) .$$

Definition 3 (Weak relevance)

A feature X_i is **weakly relevant** iff it is not strongly relevant, and there exists a subset of features S'_i of S_i for which there exists some x_i , y , and s'_i with $p(X_i = x_i, S'_i = s'_i) > 0$ such that

$$p(Y = y \mid X_i = x_i, S'_i = s'_i) \neq p(Y = y \mid S'_i = s'_i) .$$

For a Bayes rule, the optimal set of features must include all strongly relevant features and possibly some weakly relevant features. However, classifiers induced from real data do not have such nice theoretical properties. Relevance of a feature does not imply that it is in the optimal feature subset for a particular induction algorithm and, somewhat surprisingly, irrelevance does not imply that it should not be in the optimal feature subset (Kohavi and John, 1997).

Example 1 (Optimality does not imply relevance)

A feature that always takes the value one is irrelevant by any reasonable definition of relevance we can think of. But consider a limited Perceptron classifier (Rosenblatt, 1958). It has a weight associated with each feature, and it classifies instances based upon whether the linear combination of weights and feature values is greater than zero. By adding the feature that is always set to one, the limited Perceptron becomes equivalent in representational power to the regular Perceptron (which has an additional parameter allowing an arbitrary threshold, not just zero). However, removal of all irrelevant features would remove that crucial feature.

Although relevance and optimality are not equivalent concepts, the idea that they must be related empirically motivates a set of feature selection methods that measure the relevance of a feature from the data alone.

3.3 THE FILTER APPROACH

Whereas the wrapper approach attempts to identify the best feature subset to use with a particular algorithm, the *filter approach*, which is more common in statistics, attempts to assess the merits of features from the data alone. The filter approach, shown in Figure 3.2, selects features using a preprocessing step, based on the training data. In this section we describe three algorithms from the machine learning literature: FOCUS, Relief, and tree filters.

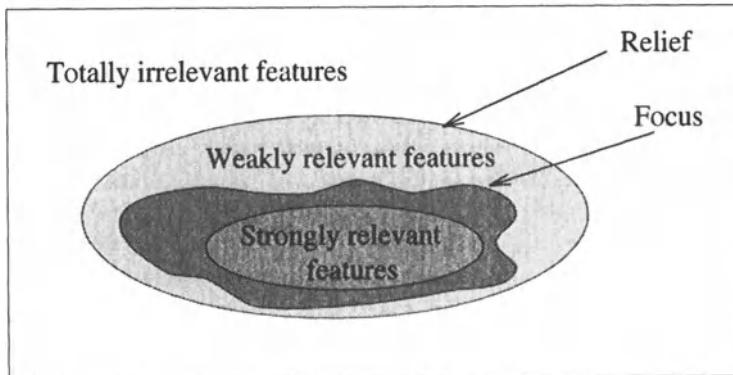


Figure 3.3 A view of feature relevance.

FOCUS (Almuallim and Dietterich, 1991; Almuallim and Dietterich, 1994), is a feature selection algorithm for noise-free Boolean domains. It exhaustively examines all subsets of features, selecting the minimal subset of features sufficient to determine the label value for all instances in the training set.

The Relief algorithm (Kira and Rendell, 1992; Kononenko, 1994) assigns a “relevance” weight to each feature, which is meant to denote the relevance of the feature to the target concept. The Relief algorithm attempts to find all relevant features. In real domains, many features have high correlations with the label, and thus many are weakly relevant, and will not be removed by Relief. Kohavi and John (1997) discuss early experiments with Relief and the variant we used, Relieved-F. Relief searches for all the relevant features (both weak and strong).

Tree filters (Cardie, 1993) use a decision tree algorithm to select a subset of features, typically for a nearest-neighbor algorithm. Although they work well for some datasets, they may select bad feature subsets because features that are good for decision trees are not necessarily useful for nearest-neighbor. Also, due to fragmentation, the tree may fail to include relevant features.

Filter approaches to feature subset selection do not take into account the biases of the induction algorithms—they select feature subsets that are independent of the induction algorithms. In some cases, measures can be devised that are algorithm-specific, and these may be computed efficiently. For example, measures such as Mallow’s C_p (Mallows, 1973) and Prediction Sum of Squares (Neter et al., 1990) have been devised specifically for linear regression. However, these tailored measures would not work well with other algorithms, such as Naive-Bayes.

Filter approaches can be easily fooled—the Corral artificial dataset from John et al., (1994) is one example. There are 32 instances with six Boolean features plus a Boolean class. The target concept is $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. A feature named “irrelevant” is uniformly random, and another feature “corre-

lated” matches the class label 75% of the time. Greedy strategies for building decision trees pick the “correlated” feature as it seems best by all known selection criteria. After the “wrong” root split, the instances are fragmented and there are not enough instances at each subtree to describe the correct concept. Although the “correlated” feature is weakly relevant, it is harmful to decision trees. When this feature is removed, the optimal tree is found.

Even a filter that perfectly selected strongly and weakly relevant features would not always work well with Naive-Bayes for example, because in some cases the performance of Naive-Bayes improves with the removal of relevant features. These examples and the discussion of relevance versus optimality above suggest that a feature selection method should take the induction algorithm into account.

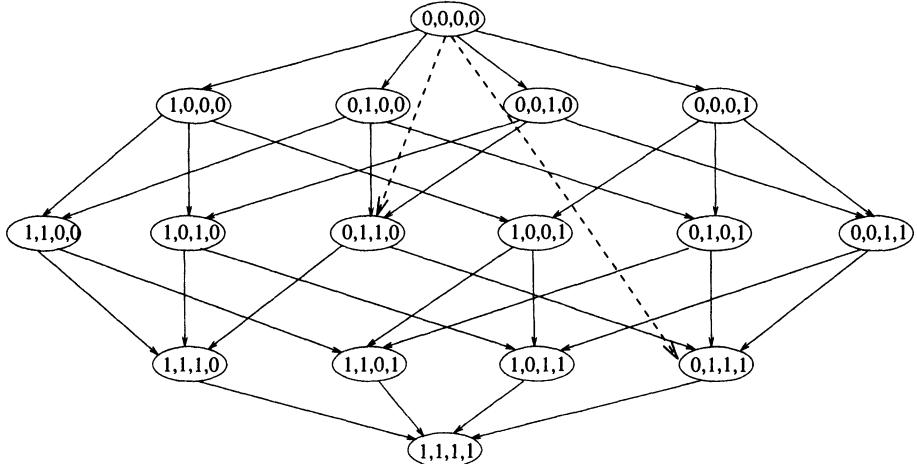
3.4 THE WRAPPER APPROACH

In the wrapper approach, shown in Figure 3.1, the feature subset selection is done using the induction algorithm as a black box (*i.e.*, no knowledge of the algorithm is needed, just the interface). The feature subset selection algorithm conducts a search for a good subset using the induction algorithm itself as part of the evaluation function. The accuracy of the induced classifiers is estimated using accuracy estimation techniques (Kohavi, 1995b) The problem we are investigating is that of state space search.

The wrapper approach conducts a search in the space of possible parameters. A search requires a state space, an initial state, a termination condition, and a search engine (Ginsberg, 1993). Below we describe a specific instantiation of the wrapper approach using best-first search.

The search space organization that we chose is such that each state represents a feature subset. For n features, there are n bits in each state, and each bit indicates whether a feature is present (1) or absent (0). Operators determine the connectivity between the states, and we have chosen to use operators that add or delete a single feature from a state, corresponding to the search space commonly used in stepwise methods in Statistics. Figure 3.4 shows such the state space and operators for a four-feature problem. The size of the search space for n features is $O(2^n)$, so it is impractical to search the whole space exhaustively, unless n is small.

The goal of the search is to find the state with the highest evaluation, using a heuristic function to guide it. Since we do not know the actual accuracy of the induced classifier, we use accuracy estimation as both the heuristic function and the evaluation function. The evaluation function we use is five-fold cross-validation, repeated multiple times. The number of repetitions is determined on the fly by looking at the standard deviation of the accuracy estimate, assuming they are independent. If the standard deviation of the accuracy estimate is above 1% and five cross-validations have not been executed, we execute another cross-validation run. While this is only a heuristic, it seems to work well in practice and avoids multiple cross-validation runs for large datasets.



Each node is connected to nodes that have one feature deleted or added. The dotted arrows are an enhancement that is explained in Section 3.4.2.

Figure 3.4 The state space search for feature subset selection

3.4.1 A Best-first Search Engine

Best-first search (Ginsberg, 1993) is a robust search method. The idea is to select the most promising node we have generated so far that has not already been expanded. The algorithm varies slightly from the standard version because there is no explicit goal condition in our problem. Best-first search usually terminates upon reaching the goal. Our problem is an optimization problem, so the search can be stopped at any point and the best solution found so far can be returned (theoretically improving over time), thus making it an anytime algorithm. In practice, we must stop the run at some stage, and we use what we call a **state search**: if we have not found an improved node in the last k expansions, we terminate the search. An improved node is defined as a node with an accuracy estimation at least ϵ higher than the best one found so far. In the following experiments, k was set to five and epsilon was 0.1%.

While best-first search is a more thorough search technique than hill-climbing, it is not obvious that it is better in practice because of the bias-variance tradeoff (Geman et al., 1992; Kohavi and Wolpert, 1996). It is possible that a more thorough search will increase variance and thus reduce accuracy. The detailed study in Kohavi and John (1997) did show that it was better for the datasets studied.

3.4.2 The State Space: Compound Operators

In the previous section, we looked at the search engine. In this section, we look at the topology of the state space and dynamically modify it based on accuracy estimation results. The state space is commonly organized such that each node represents a feature subset, and each operator represents the addition

or deletion of a feature. The main problem with this organization is that the search must expand (*i.e.*, generate successors of) every node on the path from the initial feature subset to the best feature subset. This section introduces a new way to change the search space topology by creating dynamic operators that directly connect a node to nodes considered promising given the evaluation of the node's children.

The motivation for compound operators is that the feature subsets can be partitioned into strongly relevant, weakly relevant, and irrelevant features. In practice, an optimal feature subset is likely to contain only relevant features (strongly and weakly relevant features). A backward elimination search starting from the full set of features that removes one feature at a time after expanding all children reachable using one operator will have to expand all the children of each node before removing a single feature. If there are i irrelevant features and f features, $(i \cdot f)$ nodes must be evaluated. Similar reasoning applies to forward selection search starting from the empty set of features. In domains where feature subset selection might be most useful, there are many features but such a search may be prohibitively expensive.

Compound operators are operators that are dynamically created *after* the standard set of children (created by the add and delete operators) has been evaluated. They are used for a single node expansion and then discarded. Intuitively, there is more information in the evaluation of the children than just the identification of the node with the maximum evaluation. Compound operators combine operators that led to the best children into a single dynamic operator. Figure 3.4 depicts a possible set of compound operators for forward selection. Formally, if we rank the operators by the estimated accuracy of the children, then we can define the **compound operator** c_i to be the combination of the best $i + 1$ operators. For example, the first compound operator will combine the best two operators. If the best two operators each added a feature, then the first compound operator will add both; if one operator added and one operator deleted, then we try to do both in one operation. The compound operators are applied to the parent, thus creating children nodes that are farther away in the state space. Each compound node is evaluated and the generation of compound operators continues as long as the estimated accuracy of the compound nodes improves.

Compound operators improve the search by finding nodes with higher accuracy faster. The main advantage of compound operators is that they make backward feature subset selection computationally feasible. Without compound operators, the number of features could only decrease or increase by one at every node expansion. For example, in the DNA dataset with C4.5, only 3555 nodes were evaluated in a best-first backward feature subset selection with compound operators, and a subset of 12 features was selected. Without compound operators, the algorithm would have to expand $(180 - 12) \cdot 180 = 30,240$ nodes just to get to this feature subset.

Backward feature subset selection is still a very slow technique compared with forward feature subset selection. Compared to the original algorithm,

Table 3.1 A comparison of C4.5 with no feature selection, with the Relieved-F filter (RLF), and with the wrapper using backward best-first search with compound operators (BFS).

Dataset	C4.5	C4.5-RLF	C4.5-BFS	C4.5-RLF		
				vs C4.5	vs C4.5	vs C4.5-BFS
breast cancer	95.42± 0.7	94.42± 1.1	95.28± 0.6	0.14	0.41	0.83
cleve	72.30± 2.2	74.95± 3.1	77.88± 3.2	0.84	0.98	0.82
crx	85.94± 1.4	84.06± 1.2	85.80± 1.3	0.07	0.46	0.91
DNA	92.66± 0.8	92.75± 0.8	94.44± 0.7	0.54	0.99	0.99
horse-colic	85.05± 1.2	85.88± 1.0	84.77± 1.3	0.77	0.41	0.17
Pima	71.60± 1.9	64.18± 2.3	70.18± 1.3	0.00	0.19	1.00
sick-euthyroid	97.73± 0.5	97.73± 0.5	97.91± 0.4	0.50	0.65	0.65
soybean-large	91.35± 1.6	91.35± 1.6	91.93± 1.3	0.50	0.65	0.65
Corral	81.25± 3.5	81.25± 3.5	81.25± 3.5	0.50	0.50	0.50
m-of-n-3-7-10	85.55± 1.1	91.41± 0.9	85.16± 1.1	1.00	0.36	0.00
Monk1	75.69± 2.1	88.89± 1.5	88.89± 1.5	1.00	1.00	0.50
Monk2-local	70.37± 2.2	88.43± 1.5	88.43± 1.5	1.00	1.00	0.50
Monk2	65.05± 2.3	67.13± 2.3	67.13± 2.3	0.82	0.82	0.50
Monk3	97.22± 0.8	97.22± 0.8	97.22± 0.8	0.50	0.50	0.50
Average real:	86.51	85.67	87.27			
Average artif.	79.19	85.72	84.68			

The p-val columns indicates the probability that the top algorithm is improving over the lower algorithm.

wrapper runs are about two to three orders of magnitude slower. For example, running C4.5 on the DNA dataset takes about 1.5 minutes. The wrapper approach has to run C4.5 five times for every node that is evaluated in the state space and in DNA there are thousands of nodes.

3.4.3 Experimental Comparison

Since C4.5 is an algorithm that performs well on a variety of real databases, we might expect it to be difficult to improve upon its performance using feature selection. Table 3.1 shows that this is the case: overall, the accuracy on real datasets actually decreased when using Relieved-F, but the accuracy slightly increased using the wrapper (a 5.5% relative reduction in error). Note however that Relieved-F did perform well on some artificial databases, all of which (except for Corral) contain only strongly relevant and totally irrelevant attributes. On three artificial datasets, Relieved-F was significantly better than plain C4.5 at the 99% confidence level. On the real datasets, where relevance is ill-determined, Relieved-F often did worse than plain C4.5: on one dataset its performance was significantly worse at the 99% confidence level, and in no case was its performance better at even the 90% confidence level.

Table 3.2 A comparison of Naive-Bayes (NB) with no feature selection, with the Relieved-F filter (RLF), and with the wrapper using backward best-first search with compound operators (BFS).

Dataset	NB	NB-RLF	NB-BFS	NB-BFS		
				vs NB	vs NB	vs NB-RLF
breast cancer	97.00± 0.5	95.14± 1.3	96.00± 0.6	0.03	0.04	0.80
cleve	82.88± 2.3	82.53± 2.4	82.56± 2.5	0.44	0.45	0.50
crx	87.10± 0.8	85.51± 0.8	84.78± 0.8	0.02	0.00	0.18
DNA	93.34± 0.7	93.25± 0.7	96.12± 0.6	0.45	1.00	1.00
horse-colic	79.86± 2.6	80.95± 2.3	82.33± 1.3	0.67	0.89	0.77
Pima	75.90± 1.8	64.57± 2.4	76.03± 1.6	0.00	0.53	1.00
sick-euthyroid	95.64± 0.6	95.64± 0.6	97.35± 0.5	0.50	1.00	1.00
soybean-large	91.80± 1.2	91.65± 1.2	94.29± 0.9	0.45	0.99	0.99
Corral	90.62± 2.6	90.62± 2.6	90.62± 2.6	0.50	0.50	0.50
<i>m-of-n-3-7-10</i>	86.43± 1.1	85.94± 1.1	87.50± 1.0	0.33	0.85	0.93
Monk1	71.30± 2.2	72.22± 2.2	72.22± 2.2	0.66	0.66	0.50
Monk2-local	60.65± 2.4	63.43± 2.3	67.13± 2.3	0.88	1.00	0.95
Monk2	61.57± 2.3	63.43± 2.3	67.13± 2.3	0.79	0.99	0.95
Monk3	97.22± 0.8	97.22± 0.8	97.22± 0.8	0.50	0.50	0.50
Average real:	87.94	86.16	88.68			
Average artif.	77.96	78.81	80.30			

The p-val columns indicates the probability that the top algorithm is improving over the lower algorithm.

The wrapper algorithm did significantly better than plain C4.5 on two real databases and two artificial databases, and was never significantly worse. Note that the most significant improvement on a real database was on the one real dataset with many features: DNA. Relieved-F was outperformed by the wrapper significantly on two real datasets, but it outperformed the wrapper on the *m-of-n-3-7-10* dataset.

On the Corral dataset, the wrapper selected the correct features {A1, A2, B1, B2} as the best node early in the search, but later settled on only the features A1 and A2, which gave better cross-validation accuracy. The training set is very small (32 instances), so the problem was that even though the wrapper gave the ideal feature set to C4.5, it built the correct tree (100% accurate) but then pruned it back because according to its pruning criterion the training set data was insufficient to warrant such a large tree.

Perhaps surprisingly, the Naive-Bayes algorithm turned out to be more difficult to improve using feature selection (Table 3.2). Both the filter and wrapper approaches significantly degraded performance on the breast cancer and crx databases. The filter caused significantly worse performance in one other dataset, Pima diabetes, and never significantly improved on plain Naive-Bayes,

Table 3.3 The number of features in each dataset, the number selected by Relieved-F, the number used by the plain versions of the algorithms, and the number used by the wrapped versions using backward best-first search with compound operators (BFS).

Dataset	All	RLF	C4.5	C4.5-BFS	NB-BFS	ID3	ID3-BFS
breast cancer	10	5.7	7.0	3.9	5.9	9.1	5.3
cleve	13	10.5	9.1	5.3	7.9	11.4	4.6
crx	15	11.5	9.9	7.7	9.1	13.6	7.7
DNA	180	178	46	12	48	72	36
horse-colic	22	18.2	5.5	4.3	6.1	17.4	7.2
Pima	8	1.2	8.0	4.8	4.4	8.0	5.7
sick-euthyroid	25	24	4	3	3	14	4
soybean-large	35	34.8	22.0	17.1	16.7	25.8	17.7
Corral	6	5	4	2	5	4	4
<i>m-of-n-3-7-10</i>	10	7	9	6	7	10	7
Monk1	6	3	5	3	4	6	3
Monk2-local	17	8	12	6	5	14	6
Monk2	6	4	6	0	0	6	3
Monk3	6	3	2	2	2	6	2
Average Reduction		30% vs. All	37% vs. All	46% vs. RLF	28% vs. RLF	6% vs. All	19% vs. ID3

even on the artificial datasets. This is partly due to the fact that the severely restricted hypothesis space of Naive-Bayes prevents it from doing well on the artificial problems (except for Monk3), and partly because Naive-Bayes' accuracy is hurt more by conditional dependence between features than the presence of irrelevant features.

In contrast, the wrapper approach significantly improved performance on five databases over the plain Naive-Bayes accuracy. In the Monk2 dataset it did so by discarding all features! Because the conditional independence assumption is violated, one actually obtains better performance with Naive-Bayes by throwing out all features and using only the marginal probability distribution over the classes (*i.e.*, always predict the majority class). The wrapper approach significantly improved over the filter in six cases, and was never significantly outperformed by the filter approach.

Results for ID3 are detailed in Kohavi and John (1997). The filter approach significantly degraded performance on one real dataset but significantly improved all of the artificial datasets except for Monk2, as did the wrapper approach.

We have focused only on accuracy above, so other criteria merit some consideration, such as misclassification costs or model complexity. First, the wrapper method extends directly to minimizing misclassification cost. Most Irvine datasets do not include cost information and so accuracy is a natural perfor-

mance metric, but one can trivially use a cost function instead of accuracy as the evaluation function for the wrapper. For filter approaches, adapting to misclassification costs is an open research topic. Second, we should compare the number of features selected by the filter and wrapper. Table 3.3 shows the number of features in each dataset, the number selected by the Relieved-F filter (note that since the filter is independent of the induction algorithm, it prescribes the same set of features whether using ID3, C4.5, or Naive-Bayes), and the number selected by the plain versions of the algorithms and their wrapper-enhanced versions. (Plain Naive-Bayes always uses all features, so it does not have its own column.) The average reduction column shows that the wrapper reduces the number of features used significantly more than Relieved-F.

In summary, feature subset selection using the wrapper approach significantly improves ID3, C4.5 and Naive-Bayes on some of the datasets tested. On the real datasets, the wrapper approach is clearly superior to the filter method. Perhaps the most surprising result is how well Naive-Bayes performs on real datasets once discretization and feature subset selection are performed. Some explanations for the apparently high accuracy of Naive-Bayes even when the independence assumptions are violated, are given in Domingos and Pazzani (1997). However, we can see that in some real-world domains such as DNA, the feature selection step is important to improve performance.

3.4.4 Overfitting

An induction algorithm **overfits** the dataset if it models the training data too well and its predictions are poor. An example of an over-specialized hypothesis, or classifier, is a lookup table on all the features. Overfitting is closely related to the bias-variance tradeoff (Kohavi and Wolpert, 1996; Geman et al., 1992; Breiman et al., 1984): if the algorithm fits the data too well, the variance term is large, and hence the overall error is increased.

Most accuracy estimation methods, including cross-validation, evaluate the predictive power of a given hypothesis over a feature subset by setting aside instances (holdout sets) that are not shown to the induction algorithm and using them to assess the predictive ability of the induced hypothesis. A search algorithm that explores a large portion of the space and that is guided by the accuracy estimates can choose a bad feature subset: a subset with a high accuracy estimate but poor predictive power.

Overuse of the accuracy estimates in feature subset selection may cause overfitting in the feature-subset space. Because there are so many feature subsets, it is likely that one of them leads to a hypothesis that has high predictive accuracy for the cross-validation holdout sets. A good example of overfitting can be shown using a *no-information* dataset where the features and the label are binary and completely random. When run on a small sample of 100 instances, the best-first search found a feature subset with estimated accuracy of 76% (26% optimistic) after 300 node expansions, illustrating the problem of over-searching leading to subsets with high estimated accuracy, but not necessarily high real accuracy. Although the theoretical problem exists, our experiments

with the wrapper approach indicate that overfitting is mainly a problem when the number of instances is small (Kohavi and Sommerfield, 1995). Moreover, even if the estimates are biased, the algorithm may still choose the correct feature subsets because it is the relative accuracy that matters most.

3.5 RELATED WORK

The pattern recognition and statistics literature offers many filter approaches for feature subset selection (Devijver and Kittler, 1982; Neter et al., 1990). Sequential backward elimination was introduced by Marill and Green (1963). Most machine learning induction algorithms do not obey the monotonic restrictions that underlie much of the early work in statistics and pattern recognition, and they are applied to databases with a large number of features, so they require special heuristic methods.

More recent work in feature selection in the machine learning community includes Langley (1994), which reviews feature subset selection methods in machine learning and contrasted the wrapper and filter approaches. A filter approach by Koller and Sahami (1996) based on cross-entropy seems to work well in practice. Turney (1996) defines *primary* and *contextual* features, which are related to but different from our ideas of strong and weak relevance.

The idea of wrapping around induction algorithms appeared several times in the literature without the explicit name “wrapper approach.” The closest formulation is the *Search of the Bias Space* approach described in Provost and Buchanan (1995). Moore and Lee (1994) describe an algorithm for feature subset selection that uses a search method motivated by genetic algorithms with leave-one-out cross-validation. Using a clever trick, instead of fully evaluating each node in their search space, they perform partial evaluations of all frontier nodes in parallel in a “race,” until one node becomes a clear winner.

Since the introduction of the wrapper approach (John et al., 1994), several authors have experimented with it in various contexts. Langley and Sage (1994) used the wrapper approach to select features for Naive-Bayes. Pazzani (1995) joined features (created super-features that compound others) for Naive-Bayes using the wrapper approach and showed that it indeed finds correct combinations when features interact. Singh and Provan (1995) used the wrapper approach to select features for Bayesian networks and showed significant improvements over the original K2 algorithm. Kohavi and John (1997) describe the wrapper with other search methods and as search using probabilistic estimates.

The wrapper idea has been used in several contexts in addition to feature selection. In other work, we have applied the wrapper approach to parameter tuning (specifically, setting the parameters of C4.5 for maximal performance) in Kohavi and John (1995). Brunk, Kelly and Kohavi (1997) describe a commercial data mining system, MineSet, that includes the wrapper algorithm for feature subset selection. Atkeson (1991) used leave-one-out cross-validation to search a multidimensional real-valued space which includes feature weights in addition to other parameters for local learning, similar to the generalized

memory-based learning approach (Moore et al., 1992). Skalak (1994) uses the wrapper approach for selecting a prototype subset to use with nearest-neighbor, in addition to feature selection—an interesting example of choosing training instances as opposed to features.

One might consider the wrapper methods in a larger class of methods that involve running an induction algorithm multiple times to get better results. This class would include ensemble methods such as bagging, boosting, and stacking (cf. Dietterich, 1998). The question arises whether, for a given amount of computation, it would be better to use an ensemble method than a wrapper for feature subset selection. Generally, ensembles produce high accuracy models that are much more complex than a single model produced by one run of an inducer. In contrast, the model produced by running an inducer on the feature subset selected by the wrapper will usually be even simpler than the model produced by the inducer without subset selection. So, if interpretability of the final model is important, using the wrapper for subset selection might be a better choice.

3.6 FUTURE WORK

Many variations and extensions of the current work are possible. In this chapter and previous papers, we have investigated hill-climbing and best-first search engines, starting with either the full or empty subset. Other search methods or initial states might lead to better candidate subsets. The algorithm described in this chapter explores one general area of the search space heavily when it is found to be good. It might be worthwhile to introduce some diversity into the search by restarting at random points. Exploring the search space more fully could magnify the problems with overfitting, as discussed above. Strategies for evaluating promising states more fully, by doing extra cross-validation runs or possibly using another accuracy estimation method altogether, might improve the results.

Both of these extensions—considering more nodes in the search space, and evaluating candidate nodes more fully—will obviously increase the running time of the wrapper, which can already be very slow, so another area for future work would be runtime performance enhancements. A method similar to Moore and Lee’s races would allow the wrapper to waste less time evaluating unpromising nodes. For larger datasets, it is possible to use cheaper accuracy estimation methods, such as holdout, or decrease the number of folds. Even if we continue to do a full cross-validation on every candidate subset, some inducers allow incremental addition and deletion of instances, leading to the possibility of doing incremental cross-validation as suggested in Kohavi (1995), thus drastically reducing the running time. The wrapper approach is also very easy to parallelize. In a node expansion, all children can be evaluated in parallel, which will cut the running time by a factor bounded by the number of attributes (assuming enough processors are available).

3.7 CONCLUSION

We have described the feature subset selection problem in supervised learning, which involves identifying the relevant or useful features in a dataset and giving only that subset to the learning algorithm. We have investigated the relevance and irrelevance of features, and defined two degrees of relevance: weak and strong. We have then shown that these definitions are mainly useful with respect to a Bayes optimal rule, but that in practice one should look for optimal features with respect to the specific learning algorithm and training set at hand. Such optimal features do not necessarily correspond to relevant features (either weak or strong). The optimal features depend on the specific biases and heuristics of the learning algorithm, and hence the wrapper approach naturally fits with this definition. Feature relevance helped motivate compound operators, which work well in practice and are currently the only practical way to conduct backward searches for feature subsets using the wrapper approach when the datasets have many features.

We compared Relieved-F, a filter algorithm, and our wrapper algorithm on two different families of induction algorithms: decision trees and Naive-Bayes. Significant performance improvement is achieved for both on some datasets. For the DNA dataset, the wrapper approach using Naive-Bayes reduced the error rate from 6.1% to 3.9% (a relative error reduction of 36%), making it the best induction algorithm for this problem out of all the methods used in the StatLog experiments (Taylor et al., 1994). One of the more surprising results was how well Naive-Bayes performed overall: Naive-Bayes outperforms C4.5 (with and without feature selection) on the real datasets. On average, the performance using feature subset selection improved both algorithms.

With both C4.5 and Naive-Bayes, Relieved-F degraded the average accuracy on real datasets, whereas the wrapper improved accuracy. Both feature selection algorithms improved average accuracy on the artificial datasets. The wrapper algorithm's accuracy was significantly higher than Relieved-F's in eight cases, and significantly worse on only one. Relieved-F significantly improved upon the plain inducers in three cases but was significantly worse in four. The wrapper was significantly better than the plain inducers in nine cases, and worse on only two. The wrapper reduced the number of features significantly more than the filter.

These results support our claim that subset selection can improve accuracy, and that a wrapper method should be preferable to a filter. However, we have also shown some problems with the wrapper approach, namely overfitting and the large amounts of CPU time required.

Acknowledgments

Karl Pfleger contributed to our first wrapper paper. This chapter incorporates a number of corrections and improvements suggested by readers of our earlier publications on the wrapper method, especially Pat Langley, Nick Littlestone, Nils Nilsson, and Peter Turney. Dan Sommerfield implemented large parts

of the wrapper in *MLC++* (Kohavi et al., 1996), which was used for all of the experiments. George John's work was supported under a National Science Foundation Graduate Research Fellowship. Most of the research for this chapter was completed while the authors were at Stanford University.

References

- Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.
- Almuallim, H. and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Ninth National Conference on Artificial Intelligence*, pages 547–552. MIT Press.
- Almuallim, H. and Dietterich, T. G. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–306.
- Atkeson, C. G. (1991). Using locally weighted regression for robot learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 958–963.
- Blum, A. L. and Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, 5:117–127.
- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.
- Brunk, C., Kelly, J., and Kohavi, R. (1997). MineSet: an integrated system for data mining. In Heckerman, D., Mannila, H., Pregibon, D., and Uthurusamy, R., editors, *Proceedings of the third international conference on Knowledge Discovery and Data Mining*, pages 135–138. AAAI Press.
<http://www.sgi.com/Products/software/MineSet>.
- Cardie, C. (1993). Using decision trees to improve case-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 25–32. Morgan Kaufmann Publishers, Inc.
- Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. Prentice-Hall International.
- Dietterich, T. G. (1997). Machine learning research: Four current directions. *AI Magazine*, 18(4):97–136.
- Domingos, P. and Pazzani, M. (1997). Beyond independence: Conditions for the optimality of the simple Bayesian classifier. *Machine Learning*, 29(2/3):103–130.
- Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. Wiley.
- Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–48.
- Ginsberg, M. L. (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann.
- Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. M.I.T. Press.
- Hyafil, L. and Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17.

- John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann.
- John, G. H. (1997). *Enhancements to the Data Mining Process*. PhD thesis, Stanford University, Computer Science Department.
<http://robotics.stanford.edu/~gjohn>.
- Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*. Morgan Kaufmann.
- Kohavi, R. (1995a).. The power of decision tables. In Lavrac, N. and Wrobel, S., editors, *Proceedings of the European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence 914, pages 174–189, Berlin, Heidelberg, New York. Springer Verlag. <http://robotics.stanford.edu/~ronnyk>
- Kohavi, R. (1995b). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Mellish, C. S., editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143. Morgan Kaufmann. <http://robotics.stanford.edu/~ronnyk>
- Kohavi, R. and John, G. (1995). Automatic parameter selection by minimizing estimated error. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 304–312. Morgan Kaufmann.
- Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.
- Kohavi, R. and Sommerfield, D. (1995). Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *The First International Conference on Knowledge Discovery and Data Mining*, pages 192–197.
- Kohavi, R., Sommerfield, D., and Dougherty, J. (1996). Data mining using *MLC++*: A machine learning library in C++. In *Tools with Artificial Intelligence*, pages 234–245. IEEE Computer Society Press.
<http://www.sgi.com/Technology/mlc>.
- Kohavi, R. and Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In Saitta, L., editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 275–283. Morgan Kaufmann. <http://robotics.stanford.edu/~ronnyk>.
- Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 284–292. Morgan Kaufmann Publishers, Inc.
- Kononenko, I. (1994). Estimating attributes: Analysis and extensions of Relief. In Bergadano, F. and Raedt, L. D., editors, *Proceedings of the European Conference on Machine Learning*.
- Langley, P. (1994). Selection of relevant features in machine learning. In *AAAI Fall Symposium on Relevance*, pages 140–144.

- Langley, P. and Sage, S. (1994). Induction of selective Bayesian classifiers. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 399–406, Seattle, WA. Morgan Kaufmann.
- Mallows, C. L. (1973). Some comments on c_p . *Technometrics*, 15:661–675.
- Marill, T. and Green, D. M. (1963). On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9:11–17.
- Moore, A. W., Hill, D. J., and Johnson, M. P. (1992). An empirical investigation of brute force to choose features, smoothers and function approximators. In Hanson, S., Judd, S., and Petsche, T., editors, *Computational Learning Theory and Natural Learning Systems Conference*, volume 3. MIT Press.
- Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In Cohen, W. W. and Hirsh, H., editors, *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann Publishers, Inc.
- Neter, J., Wasserman, W., and Kutner, M. H. (1990). *Applied Linear Statistical Models*. Irwin: Homewood, IL, 3rd edition.
- Pazzani, M. (1995). Searching for attribute dependencies in Bayesian classifiers. In Fisher, D. and Lenz, H., editors, *Fifth International Workshop on Artificial Intelligence and Statistics*, pages 424–429, Ft. Lauderdale, FL.
- Provost, F. J. and Buchanan, B. G. (1995). Inductive policy: The pragmatics of bias selection. *Machine Learning*, 20:35–61.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.
- Singh, M. and Provan, G. M. (1995). A comparison of induction algorithms for selective and non-selective Bayesian classifiers. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 497–505.
- Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In Cohen, W. W. and Hirsh, H., editors, *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann Publishers, Inc.
- Taylor, C., Michie, D., and Spiegelhalter, D. (1994). *Machine Learning, Neural and Statistical Classification*. Paramount Publishing International.
- Turney, P. D. (1996). The identification of context-sensitive features, a formal definition of context for concept learning. In Kubat, M. and Widmer, G., editors, *Proceedings of the Workshop on Learning in Context-Sensitive Domains*, pages 53–59.

II Subset Selection

4 DATA-DRIVEN CONSTRUCTIVE INDUCTION: METHODOLOGY AND APPLICATIONS

Eric Bloedorn^{1,2}, and Ryszard S. Michalski^{2,3}

¹The MITRE Corporation
1820 Dolley Madison Blvd., W640
McLean, VA 22102, USA

bloedorn@mitre.edu

²Machine Learning and Inference Laboratory
George Mason University
4400 University Drive
Fairfax, VA 22030, USA
{bloedorn|michalsk}@aic.gmu.edu

³Institute of Computer Science
Polish Academy of Sciences
Warsaw, ul. Ordona 21, Poland

Abstract: Constructive induction, viewed generally, is a process combining two intertwined searches: first for the best representation space, and second for the best hypothesis in that space. The first search employs operators for improving the initial representation space, such as those for generating new attributes, selecting best attributes among the given ones, and abstracting attributes. In the methodology presented, these operators are chosen on the basis of an analysis of the training data, hence the term data-driven. The second search employs an AQ inductive learning method to the examples projected at each iteration into the newly modified representation space. The aim of the second search is to determine a generalized description of examples. Experimental applications of the methodology to text categorization and natural scene interpretation demonstrate a significant practical utility of the proposed methodology.

4.1 INTRODUCTION

Inductive learning algorithms are increasingly being pressed into service, as data mining and knowledge discovery tools, to help users detect patterns or regularities in large amounts of data. These tools are needed to help human analysts make sense of the increasingly voluminous and complex data available electronically. A major limitation of conventional learning algorithms is that the descriptions they build such as decision trees, decision rules, and Bayesian nets employ only terms (attributes) selected from among those explicitly provided in the data. For that reason, such algorithms have been called *selective*. When the attributes provided in the data are inadequate then descriptions created by a (selective) learning system will likely be excessively complex, and their accuracy will be low.

In using selective learning methods the responsibility of determining attributes relevant to the problem falls entirely on the data analyst. This task may be quite difficult in practice. Yet, it is crucial for the success for the learning process.

To cope with inadequacy of attributes provided in the original data, the idea of constructive induction (CI) has been proposed (Michalski, 1978). The original formulation of the idea was concerned primarily with generating additional, more task relevant attributes from the originally given, in order to improve the learning process. It was subsequently observed that attributes used in the training data define a representation space, and a learning algorithm searches for boundaries delineating individual concepts or classes in this space. Adding more relevant attributes, removing irrelevant ones, or modifying the measurement precision of attributes are different methods for improving of the example representation space. These methods can be applied individually or jointly.

In this general view, constructive induction is a process of learning concepts descriptions that employs two intertwined searches: one for the best example representation space, and the second for the best hypothesis in the space. When searching for the best representation space, the system may make no commitment as to the description language used for creating a hypothesis (as done in various feature selection methods), or may be dependent on the description language. The second search determines a hypothesis that combines attributes (spanning the representation space) according to the assumed description language. Therefore, what constitutes the *best* representation space is, in principle, dependent on the description language used. The search for the best space and for the best hypothesis in the space are thus interrelated.

In general, the more expressive is the description language used by a learning algorithm, the lesser is the need for improving the original representation space (assuming that the original attributes are sufficiently adequate). On the other hand, the higher is the expressive power (capacity) of the description language, the higher is the complexity of the search for the best hypothesis in that language. By separating searches for the best space and for the best hypothesis, constructive induction simplifies the overall learning process in the

case of a weak representation space, and makes possible to generate an accurate hypothesis even with a relatively simple learning algorithm.

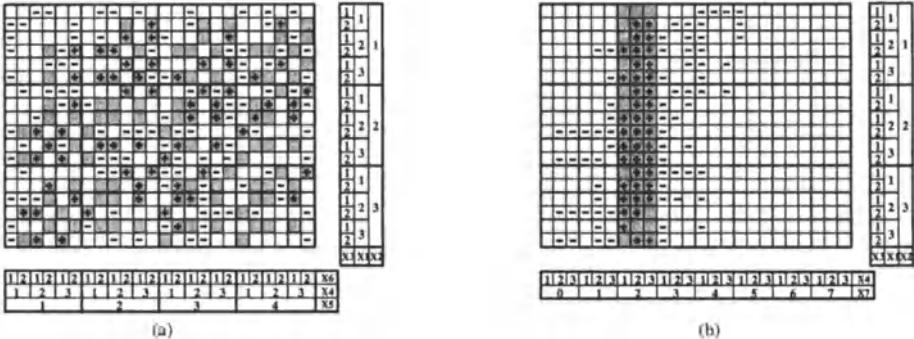
This general view of constructive induction has been used in the last several years to guide the development of a number of programs including AQ17-DCI (Bloedorn and Michalski, 1996). This chapter concerns the latest methodology underlying data-driven constructive induction in AQ17-DCI, and describes new results from its application to two practical problems. Specifically, the methodology now combines the AQ-15c learning algorithm with a wide range of data-driven representation space improvement operators. These operators are classified as either *constructors* which extend the representation space, or *destructors* which reduce the space. Constructors are based on methods of attribute generation (feature construction¹), while destructors are based on methods for attribute selection (sometimes called feature selection) or attribute abstraction. All these operators, which are usually considered in the literature separately, have been integrated in our methodology in a synergistic fashion. Experimental applications of the methodology to text categorization and natural scene interpretation have confirmed the benefits of the methodology and demonstrated the power of the proposed constructive induction approach to machine learning.

4.2 AN ILLUSTRATION OF THE IMPORTANCE OF THE REPRESENTATION SPACE

We distinguish between two types of representation spaces. One is *concept representation space* that is defined as the space being searched for determining a hypothesis generalizing given examples. Thus, this space is spanned over descriptors (attributes, predicates, transformations) that are directly used to generate an inductive hypothesis. In conventional machine learning methods concept representation space is identical to the *example representation space*, defined as the space in which training examples are represented.

As mentioned earlier, the choice of the representation space has a profound effect on the quality of the generated hypotheses. In some inductive learning problems, the boundaries of concepts in the (example/concept) representation space are very complex and thus difficult to learn. Such a problem is illustrated in Figure 4.1(a). This is the so-called second Monk's problem, which was used in an international competition of machine learning programs (Thrun, et al., 1991).

In the diagram, spanned over six attributes x_1, x_2, \dots, x_6 , individual cells represent unique combination of attribute values. Positive training examples are marked by "+" and negative training examples are marked by a "-". The shaded area represents the target concept, that is, the concept to be learned. As one can see, the boundaries of the target concept are very complex; therefore, learning the target concept in this representation space is difficult. For this reason, conventional symbolic learning methods did not perform well on this problem.



(a) the example representation space, (b) the concept representation space derived from the example space by the data-driven CI method herein described

Figure 4.1 Diagrammatic visualization of the Monk2 representation spaces

When the example representation space is transformed to an appropriate concept representation space, this difficult learning problem can become much easier. An improved representation space, as found by the AQ17-DCI system described in this chapter, is shown in Figure 4.1(b). In this space, the target concept is highly regular and easy to learn. It is then desirable to use this improved space as the concept representation space. The improved representation space was found by the program by applying a single representation space modification operator. Section 4.4 provides a detailed description of how this improvement was made by AQ17-DCI. After this transformation of the representation was made the hypothesis search phase found a simple rule that has predictive accuracy of 100%.

4.3 THE NEED FOR CONSTRUCTIVE INDUCTION

The constructive induction methodology presented here addresses the problems posed by an inadequacy of the representation space. Specifically, it offers ways to cope with an overprecision of attributes, indirect relevancy of attributes, and the presence of a large number of irrelevant attributes.

4.3.1 Problems of Overprecision

An attribute overprecision occurs when the given attributes contain a larger number of values than needed for adequately representing a given concept. This is frequently the case when learning from attributes of numeric type. When overprecision is present, the example representation space is usually large and examples are sparsely distributed. To avoid this problem, the representation space should be reduced by discretizing the attributes, that is, splitting their domains into ranges of values. Formally, such a discretization has the effect of performing an abstraction operation on the example described by the attribute (Michalski, 1994).

Dougherty et al. (Dougherty, et al., 1995) proposes a taxonomy of discretization methods based on three different axes: global vs. local, supervised vs. unsupervised and static vs. dynamic. The global vs. local axis refers to the scope of discretization, whether a small part of the space is discretized in the context of other attributes or the entire space is discretized over each attribute independently. The supervised versus unsupervised axis refers to the use of class information when making decisions about appropriate interval boundaries, and static-dynamic describes whether discretization is performed for all attributes simultaneously or in sequence. The ChiMerge algorithm (Kerber, 1992) adapted for use within AQ17-DCI is a supervised, local, static discretization method which iteratively merges adjacent intervals of an attribute until a threshold set by the user is met.

4.3.2 Problems of Attribute Interaction

Attributes are indirectly relevant when their relevance to the given classification task is dependent on an interaction with one or more other attributes. The difficulty of describing an interaction depends on the description language used by the learning algorithm. For most symbolic inductive learning algorithms interactions involving logical conjunction or disjunction are easy to describe. However, other simple interactions, such as those expressed by the equality or product of attributes, may create significant difficulties for such methods. Even more difficult to capture are interactions that are represented by complex equations involving multiple attributes. An example of a multi-attribute interaction is the even/odd parity classification problem (classifying binary strings on the basis of parity of the number they represent). Attribute generation methods can be used to overcome these interactions by building new attributes which explicitly encode attribute interaction.

The initial idea of constructive induction introduced by Michalski (1978) focussed on constructing new attributes to overcome this problem. Since then a number of other methods for constructive induction have been developed. In (Wnek and Michalski, 1994) proposed a classification of these approaches based on the primary method used for guiding modifications to the representation space. The four methods include knowledge-driven (KCI), data-driven (DCI), hypothesis driven (HCI) and multistrategy (MCI). Each of these methods has been investigated by a variety of researchers. Of particular interest here are data-driven methods for attribute generation.

The GALA system (Hu, and Kibler, 1996) emphasizes the ability of data-driven approaches to be used as a pre-processor for a variety of learning methods, including decision trees, rules and networks. Unlike AQ17-DCI, GALA performs only attribute construction, uses only Boolean operators *and* and *not* and requires a time consuming step of converting attributes to a binary form.

LFC (Ragavan and Rendell, 1993) performs data-driven construction of new attributes while constructing a decision tree. LFC uses a look-ahead before it creates each node in the decision tree. This look-ahead is constrained by a number of heuristics in order to allow LFC to complete in a reasonable time. In

contrast to AQ17-DCI, LFC performs attribute generation during hypothesis generation and uses only logical operators to construct derived terms.

4.3.3 Problems of Irrelevant Attributes

Selective induction learning methods select attributes from the given set, and as such are not significantly affected by small numbers of irrelevant attributes. However, with an increasing need to automate the process of knowledge discovery from data, and to find patterns as quickly as possible, induction methods are needed that are able to handle even a large number of irrelevant attributes.

Methods for efficiently detecting and removing irrelevant attributes in the face of possible attribute interdependencies have been studied for a number of years (Kira and Rendell, 1992), (Vafaie and DeJong, 1994), (Koller and Sahami, 1996). John et al. (John et al., 1994) provides a useful taxonomy of selection methods into wrapper approaches and filtering approaches. In the filter approach attribute selection is performed as a pre-processing step to induction. Because it is separated from the induction algorithm, filters are fast, they can be used with any induction algorithms once filtering is done, and can be used on large datasets. However, they may not agree on the relevancy of certain attributes with the induction algorithm (Imam, 1996). The wrapper approach ensures relevance to the induction algorithm by using the algorithm itself to make estimates of the relevance of the given attributes. An iterative filtering approach is used in AQ17-DCI as a compromise between speed and accurate attribute relevance determination.

4.4 A GENERAL SCHEMA FOR CONSTRUCTIVE INDUCTION

Insights gained from reviewing problems that cause difficulties for conventional learning programs clearly indicate that the quality of the representation space is the major factor. When the representation space has high quality, concepts are represented in it simply and therefore can be learned by almost any method. In real world problems, however, such quality can rarely be assured. The central goal of research on constructive induction is to develop methods capable of generating simple and accurate hypotheses for learning tasks, in which the original representation space is of poor quality. As mentioned earlier, constructive induction splits the process of learning an inductive hypothesis into two intertwined phases. Figure 4.2² shows a general scheme for constructive induction.

Initially the input data consists of a user-provided training data set plus a characterization of the initial representation space, which includes a description of attributes, their types and their domains. The training data set is split into a primary and a secondary data set. The primary training set is supplied to the Decision Rule Generation module which generates general concept descriptions in the form of rulesets. The obtained rulesets are evaluated in terms of their complexity and their performance on the secondary training set. Based on the results of this evaluation, the system decides either to stop the learning process

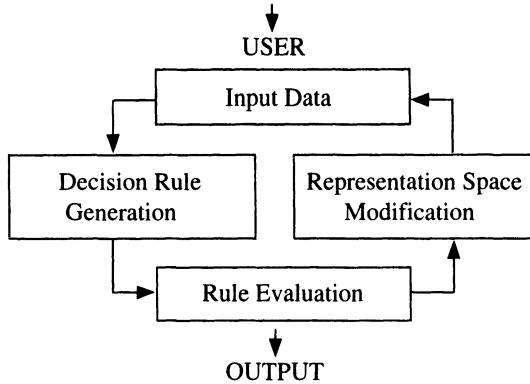


Figure 4.2 A general schema for constructive induction

(the obtained rules are output as the solution), or to move to the Representation Space Modification module. If the rules are of sufficient quality, or all planned modifications to the representation space have been tried, the final decision rules are evaluated on the testing examples to determine their performance.

The search for hypotheses within a given representation space is performed in AQ17-DCI by the AQ algorithm as implemented in AQ15c (Wnek et al., 1995). AQ15c is a progressive covering algorithm (aka "separate and conquer"). It performs a beam search (the size of which can be controlled) for a set of decision rules which cover all the positive examples and none of the negative examples (i.e. a consistent and complete ruleset). This is done in the basic mode; other modes allow it to create a consistent and/or incomplete rulesets. This search is done by randomly selecting a 'seed' event for a class, and then extending the description of this example in all dimensions, stopping each time a negative example is encountered. This process produces a star which is a set of all maximally general rules covering the seed. If all positive examples are covered then a new class is selected and the process is repeated. If not, then a new rule is generated, a new seed is selected from the same class and star generation is repeated. This process of 'seed' selection, and extension-against (negative examples) is repeated until all examples are covered.

Afterwards, the created ruleset may be modified by post-processes which may generalize the ruleset by "dropping conditions." The end result is a ruleset for each class in the data.

Each time new rules are generated, they are evaluated. The evaluation takes into consideration the predictive accuracy on the secondary training set and complexity. Predictive accuracy is measured as the percentage of secondary training examples correctly classified. The complexity of a ruleset is evaluated by counting the number of rules in the ruleset and the total number of conditions (or selectors). Rulesets are ranked lexicographically: first according to their predictive accuracy and then according to their complexity.

The representation space modification (RSM) module is responsible for determining which modification operator to apply at a given stage, and making the changes to the training and testing examples. In AQ17-DCI, the user can select which of the RSM operators should be tried (attribute generation, attribute selection, and/or attribute abstraction), or can accept a default setting that applies all operators. If in the *all operators* mode, operators are applied in a predefined order: attribute selection, attribute abstraction, and then attribute generation.³

Attribute selection in AQ17-DCI is currently performed by using an class-oriented information gain ratio criterion as a filter on the given attributes. Attributes with a value less than a given threshold are removed from the representation space.

Attribute abstraction in AQ17-DCI uses the ChiMerge algorithm to create ranges of attribute values as described in (Kerber, 1992). This is a bottom-up algorithm in which initially all values are stored in separate intervals, and then merged into ranges until a termination condition is met. The interval merging process consists of continuously repeating two steps: 1) compute χ^2 values (correlations between the value of the class attribute and the value of an attribute), and 2) merge the pair of adjacent intervals with the lowest χ^2 value. Intervals are merged until all pairs of intervals have χ^2 values exceeding the user-defined χ^2 -threshold. The chi-threshold can be determined from a table and is a function of the desired significance level and the number of degrees of freedom (1 fewer than the number of classes). The χ^2 value measures the probability that the attribute interval and class value are independent. If the interval has a χ^2 value greater than threshold then class and interval are correlated and should be retained. High χ^2 threshold settings cause more intervals to be merged resulting in fewer total intervals, or attribute values. Empirically we have found that a χ^2 threshold of 0.9 (values range from 0.1 to 1.0) is a good default.

An exhaustive generate and test approach of operator selection and application to the data is used by the RSM module during attribute generation. A number of different operators are available to construct new attributes. New operators can easily be added to this set, but the aim was to provide simple, generally applicable operators that would be easy to generate and easy to interpret by a user. Available operators are shown in Table 4.1.

Newly generated attributes must exceed a minimum discriminatory power threshold (as currently calculated by information gain ratio) and 'cost' less than a *cost increase threshold*. The cost of a new attribute is the sum of the weights given to the original attributes used in the definition of the new attribute, plus the system-defined weight associated with each operator. Binary operators select attributes in pairs and multi-argument operators use unit information to determine set membership. Both attributes must satisfy type (e.g. only metric attributes can be used with the 'addition' operator), as well as unit constraints specific for the operator (e.g. both attributes must have the same units for the subtraction operator to be applied). These constraints are useful

Table 4.1 A summary of data-driven representation space expansion operators

<i>Operator</i>	<i>Arguments</i>	<i>Notation</i>	<i>Interpretation</i>
Equivalence	Attributes x,y	$x = y$	If $x = y$ then 1, otherwise 0
Greater than	Attributes x,y	$x > y$	If $x > y$ then 1, otherwise 0
Greater than or equal	Attributes x,y	$x \geq y$	If $x \geq y$ then 1, otherwise 0
Addition	Attributes x,y	$x + y$	Sum of x and y
Subtraction	Attributes x,y	$x - y$	Difference between x and y
Difference	Attributes x,y	$ x - y $	Absolute difference between x and y
Multiplication	Attributes x,y	$x * y$	Product of x and y
Division	Attributes x,y	x / y	Quotient of x divided by y
Maximum	Attribute set S	Max(S)	Maximum value in set S
Minimum	Attribute set S	Min(S)	Minimum value in set S
Average	Attribute set S	Avg(S)	Average value in set S
Counting	Attribute set S and condition C	#Attr(S,C)	Number of attributes in S satisfying C

in reducing the number of possible combinations generated, as well as insuring the resulting new attributes are meaningful. In addition, the user may set a limit on the number of newly constructed attributes that are added to the representation space.

As an example of how AQ17-DCI works, recall the second Monk's problem described earlier. For this problem, the improved representation space was found by the program by generating and evaluating the result of using multiple operators against the set of available attributes. One of the defaults is an operator that generates counting attributes: #Attr(S, C). Such attributes measure the number of attributes in a set S that satisfy some condition C. As currently implemented, determination of membership in S is based either on user-provided attribute units, or by simply using the entire set of available attributes. The latter proved useful here. The program generated a number of new attributes and found that the counting attribute #Attr(x1,...x6, Firstvalue) is highly relevant for this problem (the condition Firstvalue means that an attribute takes the first value of from its domain for a given object).

In the hypothesis determination phase, the AQ17-DCI program found the following consistent and complete description of all the examples: #Attr(x1,...x6, Firstvalue) = 2, which can be paraphrased: an example belongs to the concept, if exactly two of six attributes take their first value. It turned out that this rule exactly represents the target concept, and thus has a predictive accuracy of 100%.

4.5 EXPERIMENTAL APPLICATIONS

The presented methodology recognizes that attribute generation, attribute abstraction and attribute selection all serve the same purpose, that is, to improve the initial representation space. The following section provides some empiri-

Table 4.2 Description of the complete set attributes used in describing text

<i>Features</i>	<i>Description</i>
x1..x5	Top 5 subject categories as computed by the SFC text classifier
x6..x59	Person tags as computed by the IDD POL tagger. For each person identified, the vector contains the following string attributes: [name, gender, honorific, title, occupation, age]. 9 people (each with these subfields) are identified for each article
x60..x104	Organization tags as computed by the IDD POL tagger. For each organization identified, the vector contains the following string attributes: [name, type, acronym, country, business]. 9 organizations (each with these subfields) are identified for each article.
x105..x140	Location tags as computed by the IDD POL tagger. For each location identified, the vector contains the following string attributes: [name, type, country, state]. 9 locations (each with these subfields) are identified for each article.
x141..x145	The top 5 ranked tf.idf terms $t_1 \dots t_n$ selected over all articles.

cal evidence for the usefulness of the integration of these representation space modification operators. These results were previously described in (Bloedorn and Michalski, 1998).

4.5.1 Text Categorization

Text categorization is the problem of classifying segments of text (usually documents) into the best single class from a set of possible classes. In this problem, the specific task is to classify incoming newswire text as either of interest, or not of interest to a given user. The goal of constructive induction is to learn a description of the user's interest (profile) from feedback from the user. Using machine learning to categorize text is an area of increasing interest because of the increasing amounts of text electronically available.

One of the most difficult aspects of this problem is in finding a good representation for English text. The use of constructive induction methods for this application builds on work reported in (Bloedorn, Mani and MacMillan, 1996), which found that a hybrid representation of extracted proper names, keywords and general subject categories coupled with a generalization hierarchy performed well for modelling newswire text. In this previous work subsets of attributes were generated and evaluated by hand. The goal of this work is to determine the effectiveness of a constructive induction approach to this problem of finding a good representation automatically.

In this problem, a user interested in medicine in the United States, has provided feedback on the relevance of 38 (18 positive and 20 negative) articles from a collection of 442 taken from the Colorado Springs Gazette Telegraph (October through November 1994). The goal of learning is to find a description of the user's interest profile so that future news articles of interest may be automatically directed to the user. In this domain articles are represented by

Table 4.3 Comparison of Predictive Accuracy on COMPLETE attribute set

<i>Method</i>	<i>Average Predictive Accuracy</i>
AQ	54.2 ± 6.9
AQ+DCI-Select	70.1 ± 5.8
AQ+DCI-Generate	67.4 ± 7.6

the set of attributes including subject categories, named entities, and keywords as shown in Table 4.2.

In the previous work, the experiments were performed on representations consisting of a) only keywords (KEYWORDS), b) only POLs (POL), c) only subjects (SUBJECTS) and d) all attributes (COMPLETE). The COMPLETE and SUBJECTS attribute sets were found to be the highest performing in the experiments previously performed. However, it was felt that constructive induction could improve both of these representations: the COMPLETE set by attribute selection, and the SUBJECTS set by attribute generation.

The COMPLETE attribute set consists of 145 attributes. The DCI-Generate (AQ+DCI-Generate) method was run in addition to the DCI-Select (AQ+DCI-Select), which were then both compared to the results obtained for the COMPLETE (AQ), set. The expectation is that the selection of attributes will have the greatest impact on predictive accuracy. A 10-fold cross-validation methodology with a 70/30 split of the data set was used. The averaged results with the 90% confidence interval are shown in Table 4.3.

These results show that a significant performance improvement was obtained both by attribute selection and attribute generation. The greatest improvement was made by DCI-Select. This was expected because the COMPLETE set contains redundant ways of describing the article by providing a list of keywords, proper names and assigned subjects to each article. It seemed clear for this small sample that only some of these attributes would be needed for discrimination. Eighty five attributes were removed including gender-person1, gender-person5 and type-org, which are not strongly correlated with this user's interest.

Some improvement was also made by DCI-Generate in constructing new attributes using the counting attribute operator $\#Attr(S,C)$. In this domain S is defined as the set $SA = \{Subj1, Subj2, Subj3, Subj4, Subj5\}$ and C is a condition on the values of the attributes in SA (e.g. $a_k \in SA \ \& \ a_k = \text{finance}$). This transformation overcomes an awkwardness of the original representation. In the original representation, an article is described by an ordered vector of subjects, for example, the description of a document as: [subject1 = medicine] or [subject2 = sports] or [subject3 = finance]. Although this ordering on the subjects may be appropriate for this document (reflecting the different strength of match between this given document and these categories), it is not suited to expressing the concept of an unordered interest. For example, if I am interested

Table 4.4 Comparison of predictive accuracy on SUBJECTS attribute set

<i>Method</i>	<i>Average Predictive Accuracy</i>
AQ	78.0 pm 6.6%
AQ+DCISelect	78.0 pm 6.6% - No change
AQ+DCIGenerate	84.7 pm 6.9%

in sports, I may not care if sports is in the subject1 or subject2 place. My interest is best stated as “Does any of the subjects include the term sports”. In this dataset there are 100 different possible subjects so extending the attribute set with all possible binary attributes would make the total vector very long and reduce the capability of the system to produce useful generalizations. These concepts are difficult to represent without the DCI generated attributes.

The counting operator of DCI solves this problem. For this problem, the counting operator generates a new attribute for each of the 100 possible subject values. Each new attribute represents the number of times (value-cardinality) that value is present in the vector. Each new attribute is filtered for discriminatory power and cost as described earlier, so as not to overwhelm the learning algorithm. This operator constructed new attributes in six of the ten runs. With the counting operator, an article’s subject can be more succinctly stated. For example to say “the article is about finance” is represented as: [$\#Attr(SA, a_k \in SA \& a_k = \text{finance}) = 1$]. The condition [$\#Attr(SA, a_k \in SA \& a_k = \text{computer}) = 0$] is used to state the “the article is not about computers”. Here is a rule describing articles of interest to the user (relevant) that used the counting generated attributes. The total coverage of the rule is shown by the t-weight. The u-weight counts the number of uniquely covered examples ::

$$\begin{aligned}
[Class = \text{relevant}] \leftarrow & [Subj1 = \text{science\&tech}] \\
& \& [Subj2 = \text{institutions}] \\
& \& [Subj3 = \text{economy\&medicine}] \\
& \& [\#Attr(SA, a_k \in SA \& a_k = \text{finance}) = 0] \\
& \& [\#Attr(SA, a_k \in SA \& a_k = \text{computer}) = 0] (t : 3, u : 3)
\end{aligned}$$

where $SA = \{\text{Subj1}, \text{Subj2}, \text{Subj3}, \text{Subj4}, \text{Subj5}\}$. An article is of interest, if subject1 is about science and technology, subject2 is about institutions, subject3 is about the economy or medicine, and if none of the subjects include “finance” or “computers”. Table 4.4 contains the results of this DCI-Generate transformation to the SUBJECTS attribute set, as well as the DCI-Select results compared against the baseline of no modification (AQ).

Data-driven constructive induction was able to improve upon both the COMPLETE Table 4.3 and SUBJECTS Table 4.4 attribute sets for the task of predicting user interest in newswire text. The transformations were quite com-

Table 4.5 Average predictive accuracy, number of rules, and training time after learning in the original representation space

Predictive Accuracy	# Rules	T. Time
72.5 %	27.7	231.7 sec.

hensible, in addition to bringing about up to a 29% improvement in predictive accuracy. Additionally, the construction of only useful forms of $\#Attr(\text{Subjects}, x)$ by the counting operator of AQ17-DCI allowed the learning algorithm to more efficiently overcome an awkwardness in the original representation of newswire articles that would have been otherwise greatly expanded the representation space, if generated by hand.

4.5.2 Natural Scene Interpretation

In the previous application, it was found the both attribute selection and attribute generation improved the representation space and the predictive accuracy and simplicity of the learned rules. This section details an application of how methods can be used together to solve a difficult problem in computer vision of classifying regions in images of natural scenes. In this application the goal is to learn a ruleset which can accurately distinguish objects in outdoor scenes under varying perceptual conditions. The approach used is to learn characterizations of classes of natural objects (sky, trees, road) from images that have been labelled. These characterizations, based on features extracted for pixel windows, can then be applied to new scenes in order to predict the presence of natural objects

In the experiment, the input to the learner was a training image which includes selected examples of the visual concepts to be learned: sky, trees and road. A windowing operator, of size 5x5 scanned over the training area, was used to extract a number of attributes including: color intensity (red, green and blue), horizontal and vertical line, high frequency spot, horizontal and vertical v-shape, and Laplacian operators. The quality of the generated rules was evaluated using a 10-fold cross-validation method. This data set has 450 examples equally distributed between the three classes.

The standard approach to solving this problem is to apply a selective induction learning algorithm to the raw data directly. The results obtained for this approach are shown in Table 4.5. This is the baseline performance.

The results obtained after attribute generation, attribute selection and attribute abstraction are shown in Table 4.6. This table shows a dramatic improvement resulting from quantization of the data. This abstraction reduced average attribute domain size from 256 to 14. The rules learned after the attribute values had been quantized were significantly more accurate, and the training time was significantly shorter than for rules learned in the original space. However, rule complexity increased from an average of 27.7 rules to

Table 4.6 Average predictive accuracy, number of rules, and training time after a single RSM operator

<i>Method</i>	<i>Accuracy</i>	<i># Rules</i>	<i>T. Time</i>	<i>CI Time</i>
DCI-Select	75.3 %	34.7	215.1 sec.	3.1 sec.
DCI-Quant	85.9% ¹	34.6	10.8 sec. ¹	5.7 sec.
DCI-Generate	87.1 % ¹	18.5 ¹	171.1 sec ¹	8.1 sec.

34.6 rules. This increase in complexity is surprising and points to the need for applying the TRUNC method for ruleset reduction and flexible matching (Bergadano, et al., 1992).

The rules learned from the space expanded by DCI-Generate were also significantly more accurate than the rules learned in the original space. DCI-Generate added on average 10 new attributes, the strongest of which described absolute and relative differences in the amount of red, green and blue color intensities. Given the green trees, the dark road and the blue sky present in the training images this is not surprising. The tree class included new attributes that stated: [green > red] and [green > blue]. The introduction of these new attributes (relations) to the representation space resulted in a significant improvement to all aspects of the resulting rules. In the new DCI-Generate expanded representation space fewer rules which were significantly more predictively accurate, and less complex were learned in shorter time than in the original representation space.

Importantly, this improvement was achieved after only 8.7 seconds spent on searching for new attributes. The total time spent on constructing new attributes, and then learning in the new space was 22% less than learning alone in the original representation space. DCI-Select consistently removed the horizontal and vertical line, high frequency spot, horizontal and vertical v-shape attributes ($x4..x8$) that describe the patterns within the 5x5 extraction window. These attributes were found irrelevant for this discrimination task. This attribute reduction resulted in a slight increase in predictive accuracy, a significant decrease in training time, but an increase in the number of rules generated. These single-strategy results are encouraging and lead to the investigation of combinations of operators, especially attribute generation through DCI-Generate combined with abstraction of the space through DCI-Quant. The next section describes these experiments.

Table 4.7 shows the results from pairwise combinations of the three RSM operators: DCI-Generate, DCI-Quant, and DCI-Select that made significant improvements to the representation space.⁴ The previous section showed how initial abstraction of the space was useful, but may have been sub-optimal given the increase in rule complexity. This finding was reinforced when DCI-Generate was run on the abstracted space. Although the new attributes resulted in fewer rules, they were less accurate than those learned in the DCI-Quant only space,

Table 4.7 Average predictive accuracy, number of rules, and training time after applying multiple RSM operators

<i>Method</i>	<i>Accuracy</i>	<i># Rules</i>	<i>T. Time</i>	<i>CI Time</i>
DCIQuant → DCIGenerate	85.4% ¹	25.6	283.0 s	7.1 s
DCIGenerate → DCISelect	90.3% ¹	25.7	142.2 s. ¹⁽²⁾	6.7 s
DCIGenerate → DCIQuant	93.4% ¹⁽³⁾	20.5	104.5 s. ¹⁽¹⁾	27.4 s

and were more complex. The contraction of the space may be removing some important information. If the operators are reversed, and DCI-Quant is applied to the space already expanded by DCI-Generate, the space is significantly improved in almost all respects: Both training time and predictive accuracy are now significantly better than in both the original space and the DCI-Generate only space. While rule complexity and number slightly increased from the DCI-Generate only space, but is still significantly smaller than in the original representation.

DCI-Select used after DCI-Generate also results in some improvements. DCI-Select after DCI-Generate results in a small increase in predictive accuracy over DCI-Generate alone, a significant decrease in training time, but an increase in the number of rules. In this space as in the original representation, DCI-Select removed attributes x4..x8. DCI-Generate followed by DCI-Quant and then DCI-Select was tried, but resulted in no improvement in predictive accuracy of the learned rules.

The multiple transformations made to the representation space (attribute generation followed by attribute quantization) resulted in rules which are significantly more accurate, can be learned much faster, and are much less complex than rules learned in the original representation.

The DCI-Generate combined with DCI-Quant result suggests that there is both an interaction between attributes and an excess of detail in the original representation. By performing DCI-Generate first, this interaction appears to be at least partially captured, and the abstraction operator of DCI-Quant can now safely perform its operation without looking at the context provided by the other attributes. Because DCI-Quant (using the ChiMerge algorithm) views each attribute independently it may remove information that is important to classification. Doing DCI-Generate first, this danger is reduced. It is premature, however, to draw any general conclusions about the best ordering of RSM operators from this single experiment. The conclusion that can be drawn from this is simply that some patterns are best described in the original formulation of the problem, and some only become apparent after abstraction. If abstraction is sensitive to interactions between attributes it may be possible to eliminate such ordering effects, but such a method would have to search an enormous space of both combinations and abstraction levels. An approach

which more tightly couples the search for combinations and abstraction level is an interesting and important area for future research.

The attributes constructed by DCI-Generate were not only useful for classification, but also have an easily interpretable meaning, as differences in color intensities. The difference in color intensity between red and green, and between green and blue, were consistently in the top three most informative attributes as measured by information gain. The difference between red and blue was also generated, but was not found to be of high discriminatory power.

The application of multiple data-driven operators helped not only to increase the prediction accuracy, but also generated meaningful new attributes. These transformations explicitly found one combination of color intensities based on difference that was useful.

4.6 CONCLUSION

This chapter presented a methodology of data-driven constructive induction as a double intertwined search: one for an improved representation space, and another for a set of hypotheses. Such constructive induction integrates within a uniform framework ideas and methods previously considered separately, for example methods of attribute selection (aka feature selection), attribute generation (aka term invention) and attribute abstraction (discretization). The advantages of integrating these methods in one system, AQ17-DCI, were demonstrated by its application to two real-world problems: text categorization and natural scene interpretation. In both problems the application of the DCI method resulted in improvements of about 25% in predictive accuracy.

The success of this theory and the resultant programs raises a number of interesting topics for future work. One such topic is the question of how tightly to integrate each of the available representation space modification operators. It was found in the natural scene interpretation problem that an ordering effect exists between attribute construction and abstraction. Developing a method for more tightly integrating these operators so that the correct level of granularity is present for the entire set of attributes available is difficult because of the massive size of the space being searched. Another topic for future work is the problem of learning “metarules” that would automatically guide the application of the representation space modification operators for any given problem. Initial work in this direction is described in (Bloedorn, Wnek and Michalski, 1993).

Acknowledgments

This research was partially conducted in the Machine Learning and Inference Laboratory at George Mason University. The experimental part describing text classification was done by the first author at MITRE under internal research support. The GMU MLI Laboratory’s research activities are supported in part by the National Science Foundation under Grants No. CDA-9309725, IRI-9020266, and DMI-9496192, in part by the Office of Naval Research under Grant No. N00014-91-J-1351, and in part by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, ad-

ministered by the Office of Naval Research, and the Grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research.

Notes

1. This chapter makes a distinction between an attribute (a one argument function that maps objects to attribute values: e.g., color or length of an object) and a feature (that expresses a specific value of property of an object, e.g., red or long)
2. Figure 4.2 appears by permission from IEEE 1998
3. The AQ17-MCI (Bloedorn, Wnek and Michalski, 1993) method extends this model by including additional representation space modification operators and by using meta-rules which relate problem characteristics to appropriate representation space modification operators.
4. Significance over the baseline using a student t-test is shown with the first superscript value. Significance to the first transformation alone is shown with the superscript in parentheses. Degree of significance level is shown with the value of the superscript ¹: significance = 0.01, ²: significance = 0.05, ³: significance = 0.1

References

- Bergadano, F., Matwin, S., Michalski R. S. and Zhang, J. (1992). "Learning Two-tiered Descriptions of Flexible Concepts: The POSEIDON System" *Machine Learning*, Vol. 8, No. 1, pp. 5-43, January.
- Bloedorn, E., Mani, I., and MacMillan T.R. (1996). "Machine Learning of User Profiles: Representational Issues", *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, August 4-8.
- Bloedorn, E., and Michalski, R.S. (1996). "The AQ17-DCI System and its Application to World Economics," *Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems*, p. 108-117. Zakopane, Poland.
- Bloedorn, E., and Michalski, R.S. (1998). "Data-Driven Constructive Induction: A Methodology and Its Applications" *IEEE Intelligent Systems* Special Issue on Feature Transformation and Subset Selection, Huan Liu and Hiroshi Motoda (Eds.), 13(2).
- Bloedorn, E., Wnek J., and Michalski, R.S. (1993). "Multistrategy Constructive Induction", *Proceedings of the Second International Workshop on Machine Learning (MSL93)*, Harpers Ferry, WV, Morgan Kaufmann, , May 27-9.
- Dougherty, J., Kohavi, R., and Sahami, M. (1995). "Supervised and Unsupervised Discretization of Continuous Features". *Proceedings of the Twelfth International Conference on Machine Learning*, p. 194-202. San Francisco.
- Hu, Y., and Kibler, D. (1996). "Generation of Attributes for Learning Algorithms", *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI96)*, p. 806-811, Portland, OR.
- Imam, I., (1996). "An Empirical Study on the Incompetence of Attribute Selection Criteria", *Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems (ISMIS96)*, p. 458-467.
- John, G., Kohavi, R., and Pfleger, K.(1994). "Irrelevant Features and the Subset Selection Problem", *Proceedings of the Eleventh International Conference on Machine Learning (ML94)*, Morgan Kaufmann, p. 121-129.

- Kerber, R. (1992). "ChiMerge: Discretization of Numeric Attributes", *Proceedings of the Tenth National Conference on Artificial Intelligence*, p. 123-128, San Jose, CA.
- Kira, K., and Rendell, L., (1992). "The Feature Selection Problem: Traditional Methods and a New Algorithm", *Proceedings of the Tenth National Conference on Artificial Intelligence*, p. 129-134, San Jose, CA.
- Koller, D., and Sahami, M., (1996). "Toward Optimal Feature Selection", *Proceedings of the Thirteenth International Conference on Machine Learning (ML96)*, p. 284-292.
- Michalski, R.S. (1978). "Pattern Recognition as Knowledge-Guided Computer Induction". *Technical Report No. 927*, Department of Computer Science, University of Illinois, Urbana-Champaign, IL.
- Michalski, R.S. (1994). "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning", *Machine Learning: A Multistrategy Approach*, R.S. Michalski and G. Tecuci (Eds.), San Mateo, CA: Morgan Kaufmann.
- Ragavan, H., and Rendell, L., (1993). "Lookahead Feature Construction for Learning Hard Concepts", *Proceedings of the Tenth International Conference on Machine Learning (ML93)*, p. 252-259.
- Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K.A., Dzeroski, S., Fahlman, S.E., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J., and Zhang, J., (1991). "The MONKs Problems: A Performance Comparison of Different Learning Algorithms". (revised version), *Carnegie Mellon University, Pittsburgh, PA, CMU-CS-91-197*
- Vafaie, H., and DeJong, K., (1994). Improving the Performance of a Rule Induction System Using Genetic Algorithms, in *Machine Learning: A Multistrategy Approach*, Vol. IV, R.S. Michalski, and G. Tecucci (Eds.), Morgan Kaufmann, San Mateo, CA.
- Wnek, J., and Michalski, R.S. (1994). "Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments". *Machine Learning*, 14, p. 139-168.
- Wnek, J., Kaufman, K., Bloedorn, E., and Michalski, R.S. (1995). "Selective Inductive Learning Method AQ15c: The Method and User's Guide", *Reports of the Machine Learning and Inference Laboratory, MLI95-4*, George Mason University, Fairfax, VA.

5 SELECTING FEATURES BY VERTICAL COMPACTNESS OF DATA

Ke Wang and Suman Sundaresan

Department of Information Systems and Computer Science
National University of Singapore
`{wangk|sumans}@iscs.nus.edu.sg`

Abstract: Feature selection is a data preprocessing step for classification and data mining tasks. Traditionally, feature selection is done by selecting fewest features that determine the class label, i.e., by the *horizontal compactness* of data. In this chapter¹, we propose a new selection criterion that aims at the *vertical compactness* of data. In particular, we select a subset of features that yields the fewest projected instances while determining the class label. Limitations of direct adoption of the standard depth-first search (DFS) and breath-first search (BFS) are analyzed. A hybrid approach that is partially DFS and partially BFS is described. To see the effectiveness of the new criterion on the classification task, we compare the result induced by C4.5 before and after the feature selection.

5.1 INTRODUCTION

While continuously searching for new efficient classification algorithms, less work has been reported on an important component of classification and data mining: the preprocessing of data. Techniques of data preprocessing include *data smoothing*, *discretization*, *feature selection*, etc. In this chapter, we investigate the effectiveness of feature selection on the classification process and the quality of rules induced. In real world applications, the number of features could be very large. It is not uncommon that the class label depends only on the values of a small number of features. In such cases, using all features in the original data may confuse the learning algorithm and result in unnecessarily complex rules. It makes sense to select only features that are relevant and sufficient to the learning task. The benefit of feeding only relevant features into a learning algorithm is two-fold: reducing the training time and improving the quality of induced rules.

Traditionally, feature selection is done by selecting the fewest features that determine the class label, see, e.g., (Almuallim and Dietterich, 1994); (Schlimmer, 1993). (There are other definitions of feature selection. See (Kira and Rendell, 1992); (Koller and Sahami, 1996)) Such a selection criterion aims at the optimal horizontal compactness of the data. One obvious problem is that in a medical diagnosis task, the patient's social security number (SSN) would be identified as one solution because SSN determines the diagnosis. This result is not useful from the learning perspective because the rules produced using only SSN would not be able to predict the diagnosis for an unseen patient. The poor generalization power lies in the fact that the selection of SSN yields no reduction in the number of projected instances. Of course, one could first remove SSN before the feature selection starts because SSN is well known to have poor generalization. However, in many other less obvious cases, the choice of which features to be removed is not always straightforward. Answering this question is exactly the task of feature selection.

In this chapter, we propose a new feature selection criterion that aims at the maximum *vertical compactness* of the data. Given a tolerance γ of the inconsistency rate, which specifies to what extent the misclassification in the data can be accepted, we select a subset of features that yield fewest projected instances without exceeding the inconsistency tolerance γ . By "projected instances" we mean the projection of instances onto the selected features with duplicates removed. This selection criterion therefore returns the most vertically compact subset for the given tolerance, thus the name vertical compactness. In the previous example, if *Sex* does not determine the class, selecting *Sex* will create many inconsistencies, even though it yields at most two projected instances, i.e., male and female. On the other hand, selecting SSN is not useful either in the sense that it yields the most number of projected instances. The benefit of the vertical compactness is two-fold. First, the more vertically compact the selected subset of features is, the more duplicates in these features repeatedly map to the same class, therefore, the more likely the selected features are relevant to the class. Second, since all duplicates can be represented by storing only one of them and the duplication number, the criterion has minimized the number of stored instances for the subsequent induction task.

This chapter is organized as follows. The next section formally defines the feature selection using *The Vertical Compactness Criterion*. The section on *Search Methods* considers the pruning potential in search for a solution to the feature selection and analyzes the pruning strength of the standard BFS and DFS methods. Next, the section on *Hybrid Search Algorithm* presents a hybrid search algorithm that combines the strength of BFS and DFS to overcome the weakness of BFS and DFS. The performance of the new criterion and the hybrid algorithm are presented in section *Experiments*. Finally, the last section concludes the chapter.

5.2 THE VERTICAL COMPACTNESS CRITERION

We now formally define the feature selection based on the vertical compactness of data. One possibility is to place an upper bound on the number of projected instances. However, very often it is difficult to know a good upper bound. Placing too small a bound will create many inconsistencies in determining the class, whereas placing too large a bound does not achieve the purpose of removing irrelevant features and reducing the size of datasets. On the other hand, one often has some idea about how much inconsistency or noise can be tolerated in the dataset. With such a tolerance, the role of feature selection is to select a subset of features that produces the least number of projected instances, provided that the tolerance is respected.

To state this notion of feature selection precisely, we define a few terms first. Consider a subset S of features. Two instances are *inconsistent* over S if they match on all features in S but mismatch on the class label. For a set of instances matching on all features in S , the *inconsistency count* of S is the number of instances in the set minus the number of instances belonging to a major class in the set. For example, there are n matching instances, among them, I_1 instances belong to one class, I_2 instances belong to another class, and I_3 instances belong to the third class, where $I_1 + I_2 + I_3 = n$. If I_3 is the largest among the three, the inconsistency count of S for these matching instances is $n - I_3$. The *inconsistency rate* of S is the sum of all inconsistency counts of S over all matching sets divided by the total number of instances. The inconsistency rate measures the level of misclassification in the dataset after changing minority classes in each matching set to a majority class. Another term for the inconsistency rate in the literature is *sum_minority*. The *data size* of S refers to the number of instances that are distinct in at least one feature in S , that is, the size of the projection onto the features in S without counting duplicates.

Here is the definition of feature selection based on the vertical compactness of data.

Definition 1 (Feature Selection) *Given an inconsistency rate γ , we select a subset S of features such that (a) the inconsistency rate of S is no more than γ and (b) the data size of S is the smallest among all subsets of features satisfying (a).*

Usually, γ is set to the noise level of the original dataset, i.e., the inconsistency rate of all features. The default value is 0 for the noise-free case. For any γ that is larger than or equal to the inconsistency rate in the original data, there is at least one, possibly many, solution to the above feature selection. A *suboptimal solution* refers to a subset S of features that satisfies (a) but not necessarily (b). Therefore, a solution is a suboptimal solution that has the smallest data size. For the rest of the chapter, we adopt Definition 1 for the feature selection.

Example 1 Consider the dataset in Table 5.1(a), which are instances of weather conditions for determining whether to play or not to play. The data

Table 5.1 A small example

(a) A small training set

<i>Outlook(O)</i>	<i>Temperature(T)</i>	<i>Humidity(H)</i>	<i>Windy(W)</i>	<i>Class</i>
sunny	hot	high	false	N
sunny	hot	high	true	N
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
rain	mild	high	true	N

(b) Subsets of features

No.	Subset	Data Size	Inconsistency Count
1	{H, O}	4	2
2	{H, T}	4	2
3	{H, W}	4	2
4	{O, T}	5	3
5	{O, W}	4	2
6	{T, W}	6	2
7	{H, O, W}	8	0
8	{H, T, W}	8	1
9	{H, O, T}	7	2
10	{O, T, W}	9	0
11	{H, O, T, W}	10	0

is noise-free. The data size and inconsistency count of all subsets of features are listed in Table 5.1(b). From the table, none of subsets of two features can determine the class, and subsets 7 and 10 determine the class. If we set $\gamma = 0$, subsets 7, 10, 11 are suboptimal solutions, of which 7 is the only solution to the feature selection. If we set $\gamma = 1$, subsets 7, 8, 10, 11 are suboptimal solutions, of which 7 and 8 are solutions. If we set $\gamma = 2$, all subsets are suboptimal solutions, of which subsets 1, 2, 3, 5 are the solutions.

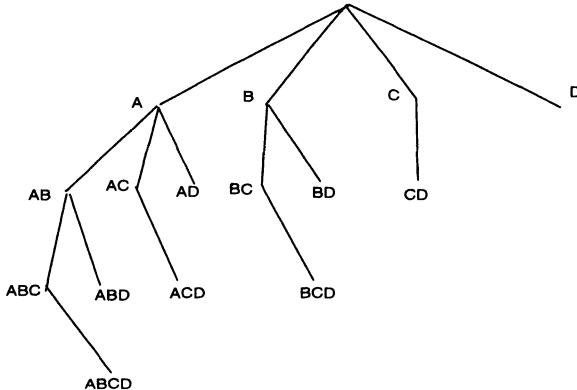


Figure 5.1 A trie-like representation of search space

5.3 SEARCH METHODS

For a dataset of n features, there are $2^n - 1$ non-empty subsets of features. Feature selection requires a complete search, i.e., not overlooking the possibility that any of these subsets might be a solution. However, being a complete search does not mean that it must be exhaustive, i.e., examining all subsets. Crucial to the performance of the algorithm is to prune useless searching as much and as early as possible. By *evaluating* a subset of features, we mean that we check the possibility of the subset as a solution. One can easily verify the correctness of the following pruning strategies.

5.3.1 Pruning Strategies

Let S_1 and S_2 be two subsets of features. Assume that S_1 is known to be a suboptimal solution.

Pruning I: If S_1 is a subset of S_2 , S_2 need not be evaluated. In fact, the data size of S_2 is at least as large as that of S_1 , thus, S_1 being a suboptimal solution eliminates the possibility of S_2 being a solution while S_1 being not. For example, the data size of $S_2 = \{Sex, Smoking, Age\}$ is at least as large as that of $S_1 = \{Sex, Smoking\}$. Therefore, if we know S_1 is a suboptimal solution, we do not need to evaluate S_2 and, in general, any superset of S_1 . This is a case where search for subsets of features can be eliminated without scanning the dataset.

Pruning II: If the data size of S_1 is not more than that of S_2 , S_2 need not be evaluated. The reason is the same as in Pruning I. We can keep track of the smallest data size of suboptimal solutions evaluated and use it to prune suboptimal solutions having a larger data size.

Pruning II is more general than Pruning I because subset S1 cannot have a larger data size than superset S2. Pruning I is separated for the convenience of analyzing several search strategies.

5.3.2 Candidates of Search Strategies

To search for a solution, we explore the space consisting of all possible subsets of features and prune the search leading to no solution. From Pruning I, finding a suboptimal solution of a smaller data size immediately prunes all supersets of the suboptimal solution. To take advantage of this pruning, we borrow the tree-like enumeration scheme in (Schlimmer, 1993), to generate successively larger subsets of features. However, we use different pruning strategies and traversal order of nodes to maximize the pruning potential of the problem. Initially, all the features are placed in the fixed order as children of the root. At the next level of the tree, children of a leaf node l are created by unioning features at l with those at one sibling node of l . Figure 5.1 illustrates the enumeration scheme for four features. Let us first examine two obvious search methods, namely, breath first search and depth first search.

Breadth First Search (BFS). BFS evaluates all subsets before their supersets, thus, fully exercises Pruning I. However, BFS does not allow a suboptimal solution or a good suboptimal solution to be found as quickly as possible so as tighten the data size bound of pruning. For example, suppose that all suboptimal solutions have at least 10 features. Then at least $2^9 - 1$ subsets of features must be evaluated before hitting the first suboptimal solution. Moreover, due to such a delayed pruning, there would be too many “open” nodes to be kept around in BFS. This combinatorial explosion may prevent the algorithm from finding a solution using the available memory space.

Depth First Search (DFS). Unlike BFS, DFS enables a suboptimal solution to be reached quickly because DFS does not require to evaluate all subsets first. However, DFS does not exercise Pruning I well, i.e., evaluating subsets before supersets. For example, in Figure 5.1 DFS will evaluate ABC before evaluating AC. Suppose that AC were evaluated and pruned before ABC. By Pruning I all supersets of AC can be pruned without scanning the dataset. This pruning potential is not well entertained by DFS.

Both BFS and DFS have obvious cases of bad performance due to the extremed search orders. We propose a hybrid search to overcome this problem.

Hybrid Search (HS). The Hybrid Search is partial BFS and partial DFS. Consider the search space in Figure 5.2 for five features A,B,C,D,E. We ignore the number attached to each node at the moment. Though node AB is a “physical” child of node A, it is also a “logical” child of node B in that AB was spawn by A and B. In general, a node of form $A_1A_2\dots A_n$ has the *physical parent* of form $A_1A_2\dots A_{n-2}A_{n-1}$ and the *logical parent* of form $A_1A_2\dots A_{n-2}A_n$, where each A_i is a feature. $A_1A_2\dots A_n$ is called the *child* of these two parents. Note that the two parents of the same child must be siblings of each other. For example, the physical parent and logical parent of ABCD are ABC and ABD,

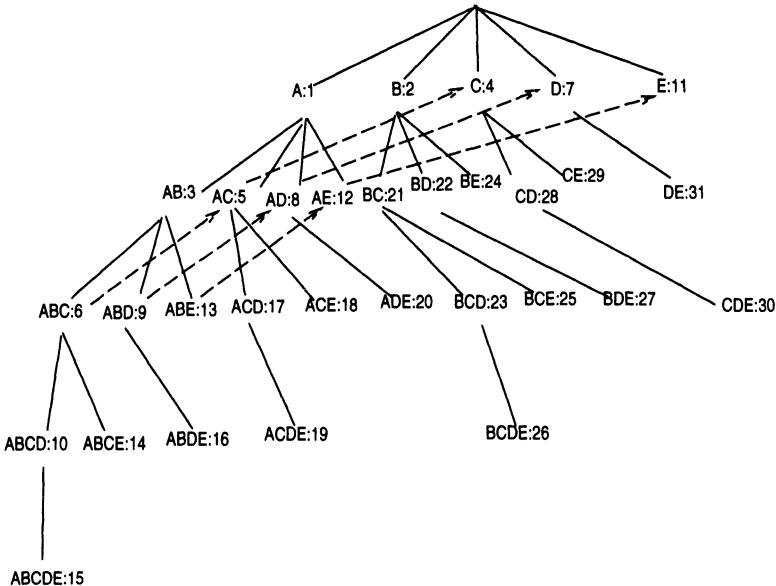


Figure 5.2 Pruning potential

and the physical parent and logical parent of AB are A and B. The essence of HS is captured by the two principles governing the search order:

Principle 1: Evaluate both logical and physical parents before evaluating a child. This is the BFS component that evaluates subsets (i.e., parents) before supersets (i.e., children). Pruning I is exercised here.

Principle 2: Evaluate the child immediately after the two parents are evaluated. This is the DFS component that goes down the tree without evaluating all nodes at the higher level. Pruning II is exercised here because going down the tree tightens the data size bound of suboptimal solutions.

The next section presents an implementation of the Hybrid Search.

5.4 HYBRID SEARCH ALGORITHM

In Hybrid Search (HS) algorithm, the search order is determined at two levels. The search order at the first level is guided by DFS. To ensure that Principle 1 is enforced, for each node visited, the second level interleaves the checking of the logical parent of the node with the DFS. The following is the general description. On visiting a node v (whose physical parent is obviously evaluated) determined by the DFS, we check the status of the logical parent v_1 of v . If v_1 is unevaluated, v_1 is pushed onto the stack and the checking goes to the logical parent of v_1 recursively. This checking of logical parents repeats until

encountering a logical parent that is either pruned or evaluated. Then the unwinding stage of the recursion pops each node from the stack and evaluates it. This is where Principle 2 is enforced. When the stack becomes empty, the computation comes back to v , at which time both parents of v have been evaluated, so we can evaluate v . Then the DFS resumes and determines the next node to be visited from unpruned nodes. We illustrate the hybrid search using an example.

Example 2 Consider Figure 5.2. If no search is pruned, the number at each node denotes the order in which the node is visited in HS. Let us see how this order is generated and how some part of the tree is pruned if pruning is allowed. HS starts with A . AB is the next node to be visited as guided by the first level DFS. The parent checking at A leads to the evaluation of B first. After B is evaluated, AB is evaluated immediately (i.e., Principle 2). The DFS resumes at AB and ABC is visited next. The parent checking at ABC leads to pushing ABC , AC , C onto the stack, as indicated by the dotted line starting at ABC , and evaluating C , AC , ABC in the unwinding stage. Suppose that ABC is the current best suboptimal solution. From Pruning I, all nodes under ABC , which supersets of ABC , are pruned. ABD is the next unpruned node to be visited in the DFS order. The parent checking at ABD pushes ABD , AD , D onto the stack. Assume that the data size of D is larger than the current best size (i.e., the data size of ABC). In the unwinding stage, by Pruning II and I, D , AD , ABD are popped off the stack and tagged as pruned.

ABE is the next unpruned node to be visited in the DFS order. The parent checking at ABE pushes ABE , AE , E onto the stack. Assume that the data size of E is larger than the current best size, by Pruning II and I, E , AE , ABE are tagged as pruned and popped off the stack. The ACD is visited next in the DFS order. Since the logical parent AD was pruned before, ACD is pruned immediately by Pruning I. ACE is visited next. Similarly, since its logical parent AE was pruned before, ACE is pruned immediately. Then BC becomes the next unpruned node in the DFS order. Such search goes on until no more unpruned can be reached. Once a node is pruned, DFS will not go into the subtree at the node.

We use the following notation. $node.dsiz$ denotes the data size for features at $node$. $best$ denotes the data size of the best suboptimal solution so far. Initially, $best$ is equal to the number of instances in the original dataset and the best suboptimal solution is the set of all features. $node.incon$ denotes the inconsistency rate at $node$. $node.physical$ and $node.logical$ denote the physical and logical parents of $node$. $node.status$ denotes the status of $node$, which is one of *pruned*, *evaluated*, *unevaluated*. Initially, all nodes are *unevaluated*. A *pruned* node has a data size larger than the best suboptimal solution, therefore, need not be further expanded. An *evaluated* node may have descendant (i.e., supersets) as a better solution and will be further expanded. $node.feature$ denotes the set of features at $node$. The HS algorithm is given in Figure 5.3. A few procedures are explained below.

```

Hybrid_Search(node):
  CheckParent(node);
  if node.status = evaluated then
    CreateChild(node);
    for each child of node do Hybrid_Search(child);

CreateChild(node):
  if node = root then
    for each feature Ai do
      node.child(i).feature = {Ai};
      node.child(i).physical = root;
      node.child(i).logical = root;
  else
    for each right sibling(j) of node do
      node.child(j).feature = node.feature  $\cup$  sibling(j).feature;
      node.child(j).physical = node;
      node.child(j).logical = sibling(j);
  tag all children as unevaluated;

ParentCheck(node):
  if node is the root then
    node.status = evaluated;
  if node.status = pruned then
    while Stack is not empty do
      node = POP;
      node.status = pruned;
  else if node.status = evaluated then
    while node.status = evaluated and Stack is not empty do
      node = POP;
      Evaluate(node);
    while Stack is not empty do
      node = POP;
      node.status = pruned;
  else
    PUSH(node);
    ParentCheck(node.logical);

Evaluate(node):
  scan the dataset to determine node.dszie and node.incon;
  Case 1: node.dszie < best and node.incon  $\leq \gamma$ .
    update node as the best suboptimal solution; /* a better solution */
    tag node as pruned; /* no better solution below node */
  Case 2: node.dszie < best and node.incon >  $\gamma$ .
    tag node as evaluated; /* there may be better solutions below node */
  Case 3: node.dszie  $\geq$  best.
    tag node as pruned; /* no better solution below node */

```

Figure 5.3 The HS algorithm

Evaluate(node): It evaluates a node by scanning the dataset to determine the data size, inconsistency rate and the status at *node*. We should distinguish *evaluating* a node from *visiting* a node. The latter does not need to scan the dataset if the status of the node is known from its logical parent. To determine the data size *node.dszie* and inconsistency rate *node.incon* at *node*, the dataset will be scanned to bring together all instances that match on the features at *node*. This can be done by hashing or sorting on such features. Hashing takes n time and sorting takes $n * \log n$ time for a dataset of n instances.

CreateChild(node): This expands the search space by creating children for *node*.

CheckParent(node): This is the key to implementing HS. When a node is reached in the DFS order, to ensure that its both parents are evaluated first, the logical parent that is unevaluated is pushed onto the stack. This is done recursively for the logical parent of the logical parent, and so on until reaching either an evaluated or pruned logical parent. If the parent was tagged as pruned, all nodes in the stack are popped and tagged as pruned. If the parent was tagged as evaluated, nodes in the stack are popped off and evaluated one at a time. Once a popped node is pruned by the evaluation, all remaining nodes in the stack are pruned.

Remarks: In HS algorithm, each node is evaluated at most once, even though some node may be visited more than once. The worst case for HS algorithm is when all features are relevant and no node can be pruned. In this case, $2^n - 1$ nodes are evaluated, where n is the number of original features. In most cases, however, the class level depends only on a small number of features and much fewer number of nodes are evaluated. To return the solution that has the least number of features, we can keep track of the number of features in the best suboptimal solution. The best suboptimal solution is updated if a suboptimal solution of smaller data size or a suboptimal solution of the same data size but fewer features is found. In some applications, learned rules are frequently used and the accuracy of the rules are important, it may be desirable to choose the best learning result by running the learning algorithm on several solutions of feature selection. To obtain all solutions, the HS algorithm only has to keep ALL current best suboptimal solutions of the same data size.

5.5 EXPERIMENTS

We tested the proposed feature selection on a few datasets in UCI Repository of Machine Learning Databases (Murphy and Aha, 1994). We focus on the reduction in the data size, the data scans incurred by the feature selection, and the quality of rules induced with and without the feature selection.

5.5.1 Feature Selection

First, we consider the feature selection itself. Table 5.2(a) shows the features selected, the data size with and without the feature selection, and the number of data scans using the three search strategies. The last three datasets have

continuous features. All datasets use consistency rate $\gamma = 0$, except Monk3(5%) which uses $\gamma = 5\%$ because the dataset contains 5% noise. If the training data and testing data are separately specified, such as all Monks, feature selection is applied to the training data; otherwise, feature selection is applied to the entire dataset. (However, this does not mean that the entire dataset is used as the training set. We use the 10-fold cross-validation for partitioning training set and testing set, as explained in the next section.)

For the first seven datasets for which the relevant features are known, our algorithm selects exactly the relevant features, as listed in the table. In all datasets, only a proper subset of original features are selected. Except for Tic-Tac-Toe, the data size is always reduced by the feature selection. Led 7 has the most reduction, i.e., from 2000 instances to 10 instances, after the last two features are removed. For Abalone and Pima, though the reduction in data size is small, only three features are used instead of the original eight features. Therefore, the size of the input to the learning algorithm is still reduced by more than 60%.

We have chosen DFS and BFS coupled with Pruning I and Pruning II, denoted DFS(P) and BFS(P), as the benchmark for comparison of the pruning strength with HS. The last three columns in Table 5.1(a) shows the number of dataset scans performed, which is equal to the number of *Evaluate(node)* performed. In most cases, HS makes fewer scans than DFS(P) and BFS(P). For the three Monks, Parity3+3, and Pima, the reduction in the number of scans ranges between 19% and 45%. Tic-Tac-Toe and Led 7 are the two cases where all the three methods make the same number of scans. We expect that HS will perform more effective pruning on real data where there are more irrelevant features.

5.5.2 Effectiveness on Induction

We also study the effect of the feature selection on learned rules. We run C4.5 (Quinlan, 1993) on the above datasets with and without feature selection. In each case, the 10-fold cross validation is applied to the whole dataset on the features selected in Table 5.2(a), and the 10-fold cross validation partitions the training and testing sets in the standard way. In running C4.5, duplicates are represented by one copy plus a duplication number, so as to take advantage of the reduced data size. Table 5.2(b) summarizes the size of pruned decision trees and the error rate on the testing set. As can be seen, in general, selected features induce better decision trees than original features. For CorrAL, Bupa, and Abalone, both tree size and error rate are reduced after feature selection. (Abalone has 29 classes, therefore, a random classification has 96.6% error rate.) For example, without feature selection the decision tree picks the correlated feature C as the root. With feature selection the correlated feature (C) and the irrelevant feature (I) are removed and the induced rules capture the target concept, i.e., $(A_0 \wedge B_0) \vee (A_1 \wedge A_2)$. See (John et al., 1994) for a description of CorrAL.

Table 5.2 Experiment

(a) Feature selection

<i>Dataset</i>	<i>Size</i>	<i># of Features</i>	<i>Features Selected</i>	<i>New Size</i>	<i>Scan Hybrid</i>	<i>Scan DFS(P)</i>	<i>Scan BFS(P)</i>
Monk1	124	7	$A_1 A_2 A_5$	35	66	107	94
Monk2	169	7	all but ID	169	69	126	126
Monk3(5%)	122	7	$A_2 A_4 A_5$	39	54	101	92
Corral	64	6	$A_0 A_1 B_0 B_1$	16	61	61	60
Parity5+5	200	10	$A_1 A_2 A_3 A_4 A_5$	32	677	683	672
Parity3+3	64	12	$A_1 A_2 A_3$	8	320	397	397
Tic-Tac-Toe	958	9	all but A_2	958	510	510	510
Led 7	2000	7	$A_1 A_2 A_3 A_4 A_5$	10	124	124	124
Bupa	345	6	$A_3 A_4 A_5$	337	43	51	51
Abalone	4177	8	$A_2 A_6 A_8$	4175	79	81	95
Pima	768	8	$A_1 A_4 A_7$	764	121	164	156

(b) Results of C4.5

<i>Dataset</i>	<i>Tree Size</i>		<i>Error Rate (%)</i>	
	<i>W/O FS</i>	<i>W FS</i>	<i>W/O FS</i>	<i>W FS</i>
Monk1	38.6	41.0	1.1	0.0
Monk2	1.0	1.0	32.9	32.9
Monk3(5%)	19.0	19.0	0.0	0.0
Corral	14.6	13.0	6.0	0.0
Parity 5+5	41.8	62.4	45.0	2.0
Parity 3+3	13.4	15.0	39.5	0.0
Tic-Tac-Toe	133.0	116.2	13.8	14.7
Led 7	19.0	19.0	0.0	0.0
Bupa	77.6	55.8	39.1	35.9
Abalone	2174.5	1882.2	79.9	79.0
Pima	125.4	136.2	29.5	38.1

The feature selection has similar effect on the two Parity datasets: a bigger decision tree is induced, but the original error rate of 40% is reduced to nearly 0. Since the error rate is collected on test data, such an improvement is not always achievable by sacrificing the tree size. In the case of Tic-Tac-Toe, the tree size has reduced significantly, with a slightly higher error rate after the feature selection. The feature selection does not affect the result on Monk2, Monk3, and Led 7 because C4.5 selects exactly same features. For Monk1, after 4 features are removed, the tree size is slightly bigger, but the error rate reduces to 0. In general, feature selection on datasets with continuous features gives

a mixed result, as shown by the last three datasets in Table 5.2(b). Unlike discrete features, continuous features have much less frequency of repeating values in the dataset, therefore, a direct feature selection on such features may not be effective in reducing the data size. A more promising approach could be combining feature selection with discretization of continuous values. This clearly needs more study.

5.6 CONCLUSIONS

We have presented a new feature selection criterion based on the vertical compactness of data while preserving the determination power of the class. While the traditional horizontal compactness measures the reduction in the number of features, the vertical compactness measures the reduction in the number of instances. We argued that the vertical compactness better captures the relevance of features than the horizontal compactness. Intuitively, if a subset of features yields most reduction in the number of projected instances and still determines the class within some tolerance of noise, the projected instances will occur most frequently in the original dataset and therefore are most frequently used to determine the class. This observation forms the basis of the vertical compactness criterion. The new criterion handles noisy data naturally by tolerating a non-zero noise level.

An immediate approach is to adopt some simple and standard search methods for solving the feature selection problem at hand. We analyzed BFS and DFS coupled with the pruning rules in the problem and found that both methods have obvious cases of bad performance. We propose a new and simple combination of BFS and DFS, the Hybrid Search, to eliminate these bad cases. Hybrid Search is conceptually as simple as DFS, but has more effective pruning than DFS. We have tested the effectiveness of the new feature selection criterion on a number of standard datasets. The experiments show that features selected by the new criterion can improve both learning process and learning result. We shall mention that Hybrid Search is in fact a general search strategy for any search problem that observes the pruning rules in section Search Methods. In the general setting, the data size and the inconsistency rate should be replaced with a general objective function and a general tolerance measure. Therefore, Hybrid Search has potential application beyond the feature selection studied in the chapter.

Notes

1. This chapter has been expanded from "Selecting Features by Vertical Compactness of Data", Wang, K. and Sundaresh, S., In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, 275-278, Copyright ©1997, American Association for Artificial Intelligence.

References

- Almuallim, H., & Dietterich, T. G. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2), 279-305.
- John, G. H., Kohavi, R., & Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 121-129. Morgan Kaufmann Publishers.
- Kira, K., & Rendell, L. A. (1992). The feature selection problem : Traditional methods and a new algorithm. In *Proceedings of Ninth National Conference on AI*, pp. 129-134. AAAI Press/MIT Press.
- Koller, D., & Sahami, M. (1996). Toward optimal feature selection. In *Machine Learning : Proceedings of the Thirteenth International Conference*. Morgan Kaufmann Publishers.
- P. Murphy and D. Aha. Repository of Machine Learning Databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>
- Quinlan, J. R. (1993). C4.5 : Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo California.
- Schlimer, J. C. (1993). Efficiently Inducing determinations : A complete and systematic search algorithm that uses optimal pruning. In *Proceedings of Tenth International Conference on Machine Learning*, 284-290.

6 RELEVANCE APPROACH TO FEATURE SUBSET SELECTION

Hui Wang, David Bell, and Fionn Murtagh

School of Information and Software Engineering

Faculty of Informatics

University of Ulster Jordanstown

Newtownabbey, Co. Antrim, BT37 0QB

Northern Ireland, UK

{h.wang|da.bell|fd.murtagh}@ulst.ac.uk

Abstract: In this chapter an axiomatic characterisation of feature subset selection is presented. Two axioms are presented: sufficiency axiom — preservation of learning information, and necessity axiom — minimising encoding length. The sufficiency axiom concerns the existing dataset and is derived based on the following understanding: any selected feature subset should be able to describe the training dataset without losing information, i.e., it is consistent with the training dataset. The necessity axiom concerns predictability and is derived from Occam's razor, which states that the simplest among different alternatives is preferred for prediction. The two axioms are then re-stated in terms of relevance in a concise form: maximising both the $r(X; Y)$ and $r(Y; X)$ relevance. Based on the relevance characterisation, a heuristic selection algorithm is presented and experimented with. The results support the axioms.

6.1 INTRODUCTION

The problem of feature subset selection (FSS hereafter) has long been an active research topic within statistics and pattern recognition (e.g., (Devijver and Kittler, 1982)), but most work in this area has dealt with linear regression. In the past few years, researchers in machine learning have realised (see for example, (Langley, 1994; Kohavi and Sommerfield, 1995)) that practical algorithms in supervised machine learning degrade in performance (prediction accuracy) when faced with many features that are not necessary for predicting the desired output. Therefore FSS has since received considerable attention from

machine learning researchers interested in improving the performance of their algorithms.

Broadly speaking, FSS is to select a subset of features from the feature space which is *good* enough regarding its ability to describe the training dataset and to predict for future cases. There is a wealth of algorithms for FSS (see for example, (Almuallim and Dietterich, 1991; Kira and Rendell, 1992; Aha and Bankert, 1994; Kononenko, 1994; John et al., 1994)). With regard to how to evaluate the *goodness* of a subset of features, the FSS methods fall into two broad categories: *filter approach* and *wrapper approach*, which are illustrated in Figures 6.1 and 6.2. In the filter approach, a *good* feature set is selected as a result of pre-processing based on properties of the data itself and independent of the induction algorithm. Section 6.5.1 presents a review on the empirical use of the notion of *goodness* in this category.

There is a special type in this approach — *feature weighting* (Kira and Rendell, 1992), which is slightly different from the mainstream filter approach in the way the search for good feature set is conducted. Basically the mainstream approach evaluates each subset of features and finds the “optimal”, while the weighting approach weighs each individual feature and selects a “quasi-optimal” set of features, typically those whose weights exceed a given threshold (Kira and Rendell, 1992; Kononenko, 1994).

In the wrapper approach, feature selection is done with the help of induction algorithms. The feature selection algorithm conducts a search for a *good* feature set using the induction algorithm itself as part of the evaluation function. Typically, the feature subset which performs best for the induction algorithm will be selected.

Both types of approach to FSS are closely related to the notion of relevance. For example, FOCUS (Almuallim and Dietterich, 1991), RELIEF (Kira and Rendell, 1992) and Schlimmer’s model (Schlimmer, 1993) use “relevance” to estimate the *goodness* of feature subset in one way or another. Section 6.5.2 presents a review in this respect. Although the wrapper approach does not use the relevance measure directly, it is shown (Kohavi and Sommerfield, 1995) that the “optimal” feature subset obtained this way must be from the relevant feature set (strongly relevant and weakly relevant features) .

However, the mathematical foundation for FSS is still lacking. In (Wang, 1996), a unified framework for relevance was proposed. In this framework relevance is quantified and related to mutual information, and furthermore, it was shown that this quantification satisfies the axiomatic characterisations of relevance laid down by leading researchers in this area. This renders the notion of relevance having a solid mathematical foundation.

In light of these, we attempt to characterise FSS in terms of the relevance framework, in order to give FSS a solid foundation for further theoretical study. We then present an algorithm for FSS based on the relevance characterisation. We also present some experimental results applying this algorithm to some real world datasets.

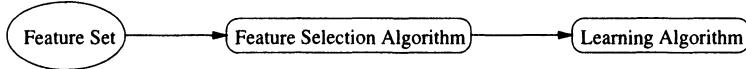


Figure 6.1 Filter model.

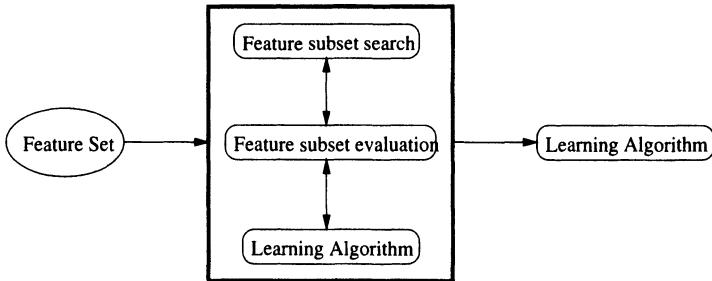


Figure 6.2 Wrapper model.

6.2 CHARACTERISATION OF FEATURE SUBSET SELECTION

In this section we are to characterise FSS in the realm of machine learning, which is confined to the following sense.

The input to a (supervised) learning algorithm is a training set D of m instances of a concept. The instances are labelled by a target variable Y . An unlabelled instance \mathbf{x} is an element of the n dimensional space $X_1 \times X_2 \times \cdots \times X_n$, where X_i is the i th feature (or variable) in the feature set $X = \{X_1, X_2, \dots, X_n\}$ ¹. Labelled instances are tuples $\langle \mathbf{x}, y \rangle$ where y is the label. Let \mathcal{L} be a learning algorithm having a hypothesis space \mathcal{H} . \mathcal{L} maps D to $h \in \mathcal{H}$ and h maps an unlabelled instance to a label. The task of the learning algorithm is to choose a hypothesis that best explains the given data D .

In this chapter, the training set D will be represented by a relation $r[X \cup Y]$ ², where X is the set of features and Y is the *target* variable. In what follows we will use $r[X \cup Y]$ to denote both the *learning task* and the training set.

The problem of feature selection is then to search for a subset Π of X that not only performs well on the training dataset, but also predicts well on unseen new cases — it is *good* enough. Our objective in this section is to characterise what the best feature subset should be from first principles as well as some known principles.

6.2.1 The Preservation of Learning Information

Given a dataset $r[X \cup Y]$, the learning task is to characterise the relationship between X and Y so that this relationship can be used to predict on future cases (either one in the dataset or a new case). Therefore any selected feature subset, if it is expected to work well on the given dataset, should preserve the existing relationship between X and Y hidden in the dataset. A natural measure of

this relationship is the mutual information (Cover and Thomas, 1991). We call this relationship *learning information*.

Specifically, given a learning task $r[X \cup Y]$, the *learning information* is the mutual information $I(X; Y)$. Furthermore, suppose Σ and Π are two subsets of X . If $I(\Sigma; Y) = I(\Pi; Y)$, then we say that Σ and Π have the same contribution to the learning task. A *sufficient feature set* or simply *SFS* of a learning task is a subset, Σ , of X such that $I(\Sigma; Y) = I(X; Y)$. Clearly, all SFS's contribute the same to the learning task. This is re-stated as the following axiom:

Axiom 6.1 (Sufficiency Axiom: preservation of learning information)
For a given learning task $r[X \cup Y]$, the best feature subset, Π , should preserve the learning information contained in the training dataset. That is, $I(\Pi; Y) = I(X; Y)$.

The following two lemmas follow directly from the chain rule for mutual information and the non-negativity of mutual information.

Lemma 6.1 *Given $r[X \cup Y]$. For any $\Pi \subseteq X$, $I(\Pi; Y) \leq I(X; Y)$.*

From this lemma and the additivity of mutual information (Cover and Thomas, 1991) we know that given a SFS Π , removing all the remaining features Σ will not lose learning information contained in the original dataset. In other words, Y is conditionally independent of Σ given Π .

Lemma 6.2 *If Π is a SFS for a learning task $r[X \cup Y]$, then any superset, Σ , of Π is also a SFS.*

This lemma helps in determining SFSs without having to calculate the learning information. This property is exploited in the design of an FSS algorithm later.

6.2.2 The Simplest Description: Occam's Razor

Given a learning task, there may be a number of SFSs. However they may not perform the same on prediction. The best feature subset should perform best in this respect. However it is not easy to determine which subset of features predicts better since there is no full knowledge about the future. Although the dataset is typically assumed to be drawn independently and identically from the labelled instance space according to an unknown distribution, this assumption doesn't help in individual cases. What we can do is to focus on the training dataset itself and then apply some empirical principles. There are a number of empirical principles. Occam's razor is one of them.

Occam's razor, known as *the principle of parsimony*, is a tool that has application in many areas of science, and it has been incorporated into the methodology of experimental science. This principle is becoming influential in machine learning, where this principle can be formulated as: given two hypotheses that both are consistent with a training set of examples of a given task, the simpler one will guess better on future examples of this task (Blumer et al., 1987; Wolpert, 1990; Amirikian and Nishimura, 1994). It has been shown (see

for example, (Blumer et al., 1987)) that, under very general assumptions, Occam's razor produces hypotheses that with high probability will be predictive of future cases.

One basic question is concerned with the meaning of “simplicity”, namely *Occam simplicity*. Typically Occam simplicity is associated with the difficulty of implementing a given task, namely *complexity of implementation*. For example, the number of hidden neurons in neural networks (Amirikian and Nishimura, 1994); the number of leaf nodes of a decision tree (Fayyad and Irani, 1990; Fayyad and Irani, 1992); the minimum description length (MDL) (Rissanen, 1986; Quinlan and Rivest, 1989); and the *minimum encoding length* (Schweitzer, 1995). However, Wolpert (Wolpert, 1990) noticed that the complexity of implementation is not directly related to the issue of prediction or generalisation, therefore there is no direct reason to believe that minimisation of such a complexity measure will result in improvement of generalisation. Wolpert then derived the *uniform simplicity measure*, which is concerned exclusively with how learning generalises. Wolpert showed that when expressed in terms of the uniform simplicity measure Occam's razor is indeed a way to set up a good generaliser.

The main disadvantage of uniform simplicity measure is that the calculation of it needs “all learning sets and all questions”, as well as guessing distribution and simplicity distribution (Wolpert, 1990). This is impossible in practice. It seems that uniform simplicity measures have only theoretical significance. Fortunately many of the conventional simplicity measures are shown to be rough approximations to the uniform simplicity measure (Wolpert, 1990). In practice we can only rely on approximations, like those mentioned above.

Back to our problem: Most of the practical simplicity measures (approximations to uniform simplicity measure) are model-dependent. However we are looking at FSS independently of any learning model, so a model-independent simplicity measure is required. Entropy seems an ideal candidate, as it measures the average number of bits (encoding length) to describe a source (e.g., a random variable).

Using the entropy as the Occam simplicity measure in our context, we have: given a learning task $r[X \cup Y]$, the Occam's razor dictates the selection of a SFS Π which minimises $H(\Pi, Y)$, where H is Shannon's entropy function. To make this formal, we re-state it, in conjunction with the information preservation axiom, as the following axiom:

Axiom 6.2 (Necessity Axiom: minimisation of encoding length) *Given a learning task $r[X \cup Y]$ and a set of sufficient feature subsets. The one Π which minimises the joint entropy $H(\Pi, Y)$ should be favoured with respect to its predictive ability.*

Now we set out to characterise the Π which minimises the joint entropy.

Lemma 6.3 *Given a learning task $r[X \cup Y]$, consider two SFSs $\Pi, \Sigma \subseteq X$. $H(\Pi, Y) \leq H(\Sigma, Y) \iff H(\Pi) \leq H(\Sigma)$.*

Proof. Since both Π and Σ are SFSs, by definition we have $I(\Pi; Y) = I(\Sigma; Y) = I(X; Y)$. Therefore we have $H(Y) - H(Y|\Pi) = H(Y) - H(Y|\Sigma) \iff H(Y|\Pi) = H(Y|\Sigma)$. Furthermore we have $H(\Pi) \leq H(\Sigma) \iff H(\Pi) + H(Y|\Pi) \leq H(\Sigma) + H(Y|\Sigma) \iff H(\Pi, Y) \leq H(\Sigma, Y)$. \square

According to this lemma, the most favourable feature subset would be the sufficient one which has the least marginal entropy.

6.2.3 Characterisation of Feature Subset Selection in Terms of Relevance

In the previous two sections we have derived two axiomatic characterisations of FSS: preservation of learning information, and minimisation of encoding length. In this section we are going to show the above two axioms can all be re-stated in terms of relevance, in an even more concise form.

Given two variables X and Y , by definition (see Appendix), the relevance of X to Y is $I(X; Y)/H(Y)$, wrt. $r(X; Y)$. Therefore for a SFS $\Pi \subseteq X$, i.e., $I(\Pi; Y) = I(X; Y)$, it is clearly $r(\Pi; Y) = r(X; Y)$. So preserving learning information amounts to preserving the relevance relationship. Since $r(\Pi; Y) \leq r(X; Y)$ in general (due to the fact that $I(\Pi; Y) \leq I(X; Y)$), the Π which preserves learning information in fact maximises the relevance $r(X; Y)$.

Consider two SFSs Π and Σ . Since, by definition, $I(\Pi; Y) = I(\Sigma; Y) = I(X; Y)$, we have $H(\Pi) \leq H(\Sigma) \iff I(\Pi; Y)/H(\Pi) \geq I(\Sigma; Y)/H(\Sigma) \iff r(Y; \Pi) \geq r(Y; \Sigma)$. Therefore, in conjunction with the previous requirement, the most favourable feature subset would be the sufficient one which maximises the relevance $r(Y; X)$.

Summarising the above discussion we have the following theorem:

Theorem 1 *Given a learning task $r[X \cup Y]$, the most favourable feature subset is the Π which is sufficient (preserving learning information, $I(\Pi; Y) = I(X; Y)$) and minimises the joint entropy $H(\Pi, Y)$ among all other SFSs. Putting it concisely, this is the one which has maximum $r(\Pi; Y)$ and maximum $r(Y; \Pi)$.*

This theorem formalises the more or less intuitively justified connection between relevance and FSS.

6.3 A RELEVANCE-BASED ALGORITHM FOR FEATURE SELECTION

In this section we present a heuristic FSS algorithm which is based on the characterisation in the previous section. A straightforward algorithm is to systematically examine all feature subsets and find one which satisfies the above two axioms. Unfortunately, as shown in (Davies and Russell, 1994), this class of algorithms turns out to be NP-complete. Branch and bound based on the characteristics of relevance was attempted (Wang, 1996), but it was shown to be also exponential in general. So we attempted heuristic approaches. Here we are to present our preferred heuristic FSS algorithm.

Our objective is to find a sufficient subset of features, which is close to optimal in the above axiomatic sense. The heuristic used here is: if a feature

is highly relevant on its own, it is very likely that this feature is in the optimal feature set. Since features are examined individually, we need to take into account the correlation among individual features. Consider, for example, two features x_1, x_2 , let Y be the target. Suppose $r(x_1; x_2) = 1$, $r(x_1; Y) = 0.9$, and $r(x_2; Y) = 0.95$. If x_1 is selected, then x_2 is not needed any more since $r(x_2; Y|x_1) = 0$ according to Lemma A.4. In other words, x_2 becomes irrelevant given x_1 . Our algorithm should not select them both. To this end, we design our algorithm, which takes advantage of conditional relevance.

CR algorithm: feature selection based on conditional relevance

```
/*Input: a learning task  $r[X \cup Y]$ , where  $|X| = N$ ;*/
/*Output: BSFS;*/
Calculate, for every  $x \in X$ , the relevance  $r(x; Y)$ ;
Find the feature  $x_0$  with largest relevance value;
 $BSFS = \{x_0\}$ ;
Repeat:
    Find  $x_i$  such that  $x_i$  is not in  $BSFS$  and
         $r(x_i; Y|BSFS)$  is largest;
    Add  $x_i$  to  $BSFS$ ;
Until  $r(BSFS; Y) = 1$ ;
Return  $BSFS$ .
```

Clearly the time complexity for calculating relevance and finding the largest is $O(N)$. We now analyse the complexity for the *repeat-until* loop. At loop k where there are $N - k - 1$ features left for inspection, we need to compute conditional relevance $r(x_i; Y|BNAS)$ for all $N - k - 1$ features, hence a complexity of $O(N - k - 1)$. To find the feature with largest conditional relevance value, we need $k - 1$ comparisons, hence a complexity of $O(k - 1)$. In the worst case we need to loop from $k = 1$ to $k = N - 1$, hence the complexity is $\sum_{k=1}^{N-1} (N - k - 1 + k - 1) = O(N^2)$. Therefore the overall complexity for above algorithm is $O(N^2)$.

This algorithm is highly dependent on the choice of the initial set of features, which is the individual feature most relevant to Y . The BSFS selected by CR is guaranteed to be SFS, but not guaranteed to be necessary. It is conjectured that if x_0 is in the optimal SFS, then the BSFS found by CR is indeed optimal.

6.4 EXPERIMENT AND EVALUATION

Here we are to evaluate the performance of the feature selection algorithm presented in the previous section using some real world datasets. We choose three datasets from the U. C. Irvine machine learning repository (Murphy and Aha, 1994): Australian, Diabetes, and Heart. Some general information about these datasets is shown in Table 6.1.

To evaluate the performance of our feature selection algorithm, we chose to use the C4.5 module in the Clementine package (Data mining system) in our experiment. We feed the selected feature subsets to C4.5 and compare the results with and without feature subset selections.

Table 6.1 General information about the datasets

<i>Dataset</i>	<i>features</i>	<i>no. of examples</i>	<i>no. classes</i>	<i>class distribution</i>
Australian	14	690	2	44.5%(+)
Diabetes	8	768	2	65.1%(+)
Heart	13	270	2	55.56%(+)

Table 6.2 Decision tree sizes, test accuracies on decision trees generated by C4.5 without and with feature selection, together with the selected feature sets.

<i>Dataset</i>	C4.5		C4.5-CR		
	<i>Size of trees</i>	<i>Test accuracy</i>	<i>Selected features</i>	<i>Size of trees</i>	<i>Test accuracy</i>
Australian	32	85.2	2,3,8,9,10,13,14	22	85.7
Diabetes	54	72.9	2,5,6,7,8	42	74.2
Heart	16	77.1	5,9,12,13	12	80.8

The test accuracies by C4.5 without and with feature selection are shown in Table 6.2. The evaluation method we used is cross validation implemented in Clementine. From these experiment results we see that applying our feature selection algorithm does indeed improve the test accuracies for all three datasets, and the corresponding decision trees have smaller sizes. However the success is limited in the sense that the accuracy improvements were not very great in this case. The reason is probably that C4.5 has a built-in feature selection facility based on mutual information. It is then reasonable to believe that if the feature selection algorithm described above is used with other learning algorithms without built-in feature selection facilities (e.g., nearest neighbour), the accuracy improvement could be higher than those reported here.

We also carry out an experiment to inspect the change of accuracies through gradually adding features in the order of relevance values. We first rank all the features according to their individual relevance values ($r(X; Y)$ only) and start evaluation from the one with highest relevance value. The results are shown in Figure 6.2, where the $(r(X; Y))$ relevance-based rankings for the three datasets are as follows. Australian: 2,14,8,3,13,7,10,9,5,6,4,12,11,1; Diabetes: 7,6,2,5,8,4,1,3 ; Heart: 5,8,10,13,3,12,1,4,9,11,2,7,6. From this figure we can see that as features are gradually added in the order, the accuracy will on average go up first and reach a peak and then go down. This diagram justifies to some extent our algorithm, although the algorithm may not always find the feature subsets corresponding exactly to the peak points.

6.5 COMPARISON WITH RELATED WORK

In this section we are to take a closer look at some related work from the relevance point of view and compare them with ours.

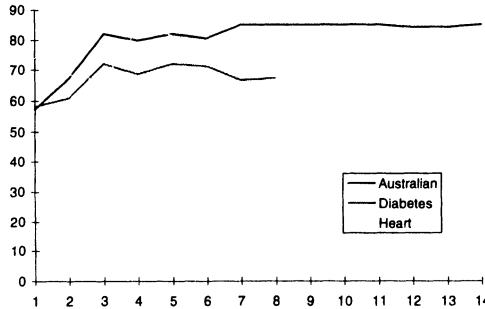


Figure 6.3 Accuracy vs. first k features used in the relevance ranking, where k starts from 1.

6.5.1 How is the best feature subset characterised in the literature?

In (Kira and Rendell, 1992), the best feature subset is characterised as *sufficient and necessary* to describe the target. Ideally the sufficiency and necessity requirement is quantified by a measure $J(\Pi, Y, D)$ which evaluates the feature subset Π for the target Y and the given data D : the best feature subset should have the best value of $J(\Pi, Y, D)$. However the nature of the sufficiency and necessity requirement was not made clear in (Kira and Rendell, 1992). In the context of learning from examples, it seems reasonable that sufficiency concerns the ability of a feature subset to describe the given dataset (called *qualified* later on), while the necessity concerns the optimality among all the qualified feature subsets regarding predictive ability. From this we can say that our two axiomatic characterisations are possible interpretations of the sufficiency and necessity requirement proposed in (Kira and Rendell, 1992).

In practice, the best feature subsets are measured in pragmatic ways. For example, in FOCUS (Almuallim and Dietterich, 1991) a *good* feature subset is a minimal subset which is consistent with the training dataset. Here the consistency can be understood as the sufficiency requirement, since only when the feature subset is consistent with the given dataset can it qualify to describe the dataset without losing learning information. The minimality of feature subset can be understood as the necessity requirement, as it was used as a bias of learning regarding which subset can predict better for future cases. In RELIEF (Kira and Rendell, 1992), a *good* subset is one whose elements each has a relevance level greater than a given threshold. Here the relevancy and the threshold together determine whether a given feature subset is sufficient (or qualified) to describe the given dataset. But there is no direct justification as to why the feature subset determined in this way would perform better in predicting for future cases, i.e., necessary. In (Schlimmer, 1993) a *good* subset is one of the minimal determinations, but nothing is mentioned as to which one is the *best*. Here all the minimal determinations are sufficient, but which of these is necessary is left open.

6.5.2 Re-modelling Using the Relevance Framework

Many FSS algorithms use “relevance” to estimate feature usefulness in one way or another. The FOCUS (Almuallim and Dietterich, 1991) algorithm starts with an empty feature set and carries out breadth-first search until it finds a minimal combination Π of features which is consistent with the training dataset. The features in Π are relevant to the target (concept) C . In terms of the relevance framework (Wang, 1996), this requirement amounts to $r(\Pi; C) = 1$ and $|\Pi|$ being minimum.

RELIEF is a feature relevance estimation algorithm, but the meaning of relevance is different from ours and has not been theoretically justified. It associates with each feature a weight indicating the relative relevance of that feature to the target (concept) (C) and returns a set of features whose weights exceed a threshold. This amounts to firstly calculate, for each feature X , $r(X; C)$, and then select a set of features such that for any X in this set, $r(X; C) \geq \tau$, where τ is the threshold. Compared to FOCUS, this method is computationally efficient. Furthermore, it allows features to be ranked by relevance.

Schlimer (Schlimer, 1993) described a related approach that carries out a systematic search through the space of feature sets for all (not just the one with minimal cardinality) minimal determinations which are consistent with training dataset. The algorithm has an attractive polynomial complexity due to the space-for-time technique: caching the search path to avoid revisiting states. A determination is in fact a SFS, and a minimal determination is such a SFS that removing any element will render it not being a SFS anymore. Therefore this algorithm amounts to finding all SFSs within a given length such that for each of these, Π , $r(\Pi; C) = 1$ and for any $X \in \Pi$, $r(\Pi/\{X\}; C) < 1$.

Most recent research on feature selection differs from these early methods by relying on wrapper strategies rather than filtering schemes. The general argument for wrapper approaches is that the induction method that will use the feature subset should provide a better estimate of accuracy than a separate measure that may have an entirely different inductive bias. John, Kohavi, and Pfleger (John et al., 1994) were the first to present the wrapper idea as a general framework for feature selection. The generic wrapper technique must still use some measure to select among alternative features. One natural scheme involves running the induction algorithm over the entire training data using a given set of features, then measuring the accuracy of the learned structure on the training data. However, John et al argue that a cross-validation method, which they use in their implementation, provides a better measure of expected accuracy on novel test cases.

The major disadvantage of wrapper methods over filter methods is the former’s computational cost, which results from calling the induction algorithm for each feature set considered. This cost has led some researchers to invent ingenious techniques for speeding the evaluation process.

The wrapper scheme in (Kohavi and Sommerfield, 1995) does not use the relevance measure directly; rather, it uses the accuracy obtained by applying an

induction algorithm as the measure for the goodness of feature sets. However, Kohavi and Sommerfield show that the “optimal” feature set X obtained this way must be from the relevant feature set (strongly relevant and weakly relevant features). As shown in (Wang, 1996) their strong relevance and weak relevance can be characterised by our relevance formalism, so the wrap per scheme can also be modelled by our relevance, $r(X; C) > 0$.

However, Caruana and Freitag (Caruana and Freitag, 1994) observe that not all features that are relevant are necessarily useful for induction. They tested FOCUS and RELIEF on the calendar scheduling problem, where they fed the feature sets obtained by those two algorithms to ID3/C4.5, and found that a more direct feature selection procedure, hill-climbing in feature space, finds superior feature sets. They didn’t explain the reason for this. But a possible explanation based on relevance is as follows. For a given target (concept) C there are many SFS’s, where for each SFS, X , $r(X; C) = 1$. One of the many SFS’s, which satisfies some criteria, should be optimal in general. This optimal feature set may not be the minimal one in general. Starting from Occam’s razor, we argue that the optimal one should be such that $r(C; X)$ is maximised.

In conclusion from the above discussion, RELIEF, Schlimmer’s algorithm, and Wrapper take into account only the sufficiency condition, evidenced by their addressing only $r(X; C)$. FOCUS takes into account both sufficiency and necessity conditions. But the necessity is measured by the cardinality of the feature subset being minimal. The relationship of this measurement to the Occam’s razor characterisation above is not clear yet.

6.6 CONCLUSION

In this chapter we have derived, from first principles and Occam’s razor principle, two axiomatic requirements for any feature subset to qualify as “good”: preservation of learning information and minimisation of encoding length. Since FSS has traditionally linked with relevance, we further showed that when identified with the variable relevance in the unified framework for relevance, relevance has a direct relationship with FSS: maximising relevance in both ways (i.e., $r(X; Y)$ and $r(Y; X)$) will result in the favourable feature subset.

Based on the axiomatic characterisation of FSS, one heuristic FSS algorithm was designed and presented. This algorithm weights (ranks) features using conditional relevance $r(X; Y|Z)$ in a step-wise way: it starts with the feature with the highest unconditional relevance value and then keeps selecting features with highest conditional relevance values with respect to the current selected subset. This algorithm can get rid of highly correlated features, and it is shown to have a complexity of $O(n^2)$.

We also presented evaluation results using three real world problems: Australian credit, Diabetes diagnosis, and Heart diagnosis, all from the UCI machine learning repository. The purpose of the evaluation is two fold. Firstly, we evaluated the performance of the algorithm. The results are quite encouraging: the average test accuracies on three datasets were all improved, and the resultant decision trees had smaller tree sizes. Since C4.5 has a built-in feature

selection process, which is based on gain ratio defined by mutual information, we conjecture that if the algorithm is used with other learning algorithms without a built-in feature selection process (e.g., nearest neighbour), the accuracy improvement could be higher.

Secondly, we evaluated the relationship between relevance and learning accuracy. The results show a strong connection between relevance and learning accuracy. When all features are ranked according to their relevance values, adding features one by one to the feature set would lead to a clear pattern of accuracy: first ascending to a peak and then descending gradually. Therefore we conclude that highly relevant features can improve learning accuracies and highly irrelevant features can degrade learning accuracies.

Appendix: A unified framework for relevance

Relevance is a common sense notion in our daily lives. It concerns the relationship among objects. Suppose we have two objects X and Y . We understand X is relevant to Y if knowing X happening would change the likelihood of Y . Based on this understanding, various formulations have been proposed with regard to different problem domains. A unified framework was proposed in (Wang, 1996) which unifies two basic types of relevance in a consistent way: variable relevance and instance relevance. Various guises of relevance can be modelled by this framework. For the purpose of this chapter, we present a brief introduction to the variable relevance.

The relevance of one variable to another (target) variable is understood in information theoretic terms, as the *mutual information between the two variables relative to the entropy of the target variable*, or in other words, the relative reduction of entropy (uncertainty) of one variable due to the knowledge of another. The bigger the reduction, the higher the relevance. Formally we have:

Definition A.1 *Given three random variables X , Y and Z with a joint probability distribution p , if $H(Y|Z) \neq 0$, then the variable relevance of X to Y given Z , denoted $r(X; Y|Z)$, is defined as*

$$r(X; Y|Z) = \frac{I(X; Y|Z)}{H(Y|Z)} = \frac{H(Y|Z) - H(Y|X, Z)}{H(Y|Z)}$$

If $H(Y|Z) = 0$, then $r(X; Y|Z) = 0$.

This definition says that the relevance of X to Y given Z is indicated by the relative reduction of uncertainty of Y when X and Z are known. With this notion we can say that X is relevant to Y given Z with degree $r(X; Y|Z)$. Examples can be found in (Wang, 1996).

Theorem A1 (Dependency vs independency) *Suppose X , Y , and Z are three random variables with a joint distribution p . Then $r(X; Y|Z) = 1 \iff Y$ is conditionally fully dependent on X given Z ; $r(X; Y|Z) = 0 \iff Y$ is conditionally independent of X given Z .*

This theorem shows that the definition of variable relevance agrees with two extreme cases of probabilistic dependence: full dependence and full independence. In other word, two extreme cases can be identified by our variable relevance measure: 0 for extreme irrelevance (conditional independence) and 1 for extreme relevance (full dependence).

There are some useful properties for variable relevance. Here we list some of them.

Lemma A.4 *The following properties hold for variable relevance:*

- *Uniformity:* $0 \leq r(X; Y|Z) \leq 1$. That is, relevance measures lie between two fixed extremes.
- *Self-reflexiveness:* $r(X; X|Z) = 1$. If $Y \subseteq X$, then $r(X; Y) = 1$.
- *Symmetry:* $r(X; Y|Z) \geq 0 \iff r(Y; X|Z) \geq 0$. But in general, $r(X; Y|Z) \neq r(Y; X|Z)$.
- *Monotonicity:* For two sets of variables Σ and Ω , if $\Sigma \subseteq \Omega$, then $r(\Omega; Y) \geq r(\Sigma; Y)$.
- *Intransitivity:* In general $r(X; Y|Z) > 0 \& r(Y; W|Z) > 0 \not\Rightarrow r(X; W|Z) > 0$, and $r(X; Y|Z) = 0 \& r(Y; W|Z) = 0 \not\Rightarrow r(X; W|Z) = 0$.
- *Saturability:* If $r(X; Y|Z) = 1$, then $r(W; Y|Z, X) = 0$, where W is any variable.
- Given two variables X, Y, Z with a joint distribution p , $r(Z; Y) = 1 \iff r(X; Y|Z) = 0 \iff r(Y; X|Z) = 0$.

Other properties can be found in (Wang, 1996).

Notes

1. In this chapter we use X , to refer to both a variable and the domain of the variable, when this can be identified from the context.
2. We use the notation in (Hill, 1991). A relation scheme R is a set of variables (features). A relation over R is an indicator function for a set of tuples, written $r[R] : r[R](t) = 1$ if the tuple t is in the relation; $r[R](t) = 0$ otherwise. For the purpose of this chapter, we extend the indicator function such that $r[R](t) = n$, where n is the frequency of tuple t appearing in the relation. With this extension, we can talk about the distribution of the tuples, which can be easily obtained.

References

- Aha, D. W. and Bankert, R. L. (1994). Feature selection for case-based classification of cloud types. In *Working notes of the AAAI94 Workshop on Case-based Reasoning*, pages 106–112. AAAI Press.
- Almuallim, H. and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proc. Ninth National Conference on Artificial Intelligence*, pages 547–552. MIT Press.

- Amirikian, B. and Nishimura, H. (1994). What size network is good for generalization of a specific task of interest? *Neural Networks*, 7(2):321–329.
- Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1987). Occam's Razor. *Information Processing Letters*, 24:377–380.
- Caruana, R. and Freitag, D. (1994). How useful is relevance? In *Proceedings of the 1994 AAAI Fall Symposium on Relevance*, pages 21–25. AAAI Press.
- Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. John Wiley & Sons, Inc.
- Data mining system, C. A. Integral Solutions Limited (ISL). <http://www.isl.co.uk/>.
- Davies, S. and Russell, S. (1994). NP-Completeness of Searches for Smallest Possible Feature Sets. In *Proceedings of the 1994 AAAI Fall Symposium on Relevance*, pages 37–39. AAAI Press.
- Devijver, P. A. and Kittler, J. (1982). *Pattern recognition: A statistical approach*. New York: Prentice-Hall.
- Fayyad, U. and Irani, K. (1990). What should be minimized in a decision tree? In *AAAI-90: Proceedings of 8th National Conference on Artificial Intelligence*.
- Fayyad, U. and Irani, K. (1992). The attribute selection problem in decision tree generation. In *AAAI-92: Proceedings of 10th National Conference on Artificial Intelligence*.
- Hill, J. R. (1991). Relational Databases: A Tutorial for Statisticians. In Keramidas, E. M. and Kaufman, S. M., editors, *Computing Science and Statistics: Proc. of the 23rd Symposium on the Interface*, pages 86–93.
- John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the 11th international conference on machine learning*, pages 121–129. New Brunswick, NJ: Morgan Kaufmann.
- Kira, K. and Rendell, L. A. (1992). The feature selection problem: traditional methods and a new algorithm. In *AAAI-92*, pages 129–134.
- Kohavi, R. and Sommerfield, D. (1995). Feature Subset Selection Using the Wrapper Method: Overfitting and Dynamic Search Space Topology. In Fayyad, U. M. and Uthurusamy, R., editors, *Proceedings of KDD'95*, pages 192–197.
- Kononenko, I. (1994). Estimating attributes: analysis and extensions of RELIEF. In *Proceedings of the 1994 European Conference on Machine Learning*.
- Langley, P. (1994). Selection of relevant features in machine learning. In *Relevance: proc. 1994 AAAI Fall Symposium*, pages 127–131. AAAI Press.
- Murphy, P. M. and Aha, D. W. (1994). *UCI Repository of Machine Learning Databases and Domain Theories*. Irvin, CA. <ftp://ftp.ics.uci.edu>.
- Quinlan, J. and Rivest, R. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248.
- Rissanen, J. (1986). Stochastic complexity and modeling. *Ann. Statist.*, 14:1080–1100.

- Schlümer, J. C. (1993). Efficiently inducing determinations: a complete and systematic search algorithm that uses optimal pruning. In *ML93*, pages 284–290.
- Schweitzer, H. (1995). Occam algorithms for computing visual motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1033–1042.
- Wang, H. (1996). *Towards a unified framework of relevance*. PhD thesis, Faculty of Informatics, University of Ulster, N. Ireland, UK. <http://www.infm.ulst.ac.uk/~hwang/thesis.ps>.
- Wolpert, D. H. (1990). The relationship between Occam’s Razor and convergent guessing. *Complex Systems*, 4:319–368.

7 NOVEL METHODS FOR FEATURE SUBSET SELECTION WITH RESPECT TO PROBLEM KNOWLEDGE

Pavel Pudil and Jana Novovičová

Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic,
Pod vodarenskou veží 4, Prague 8, Czech Republic*
{pudil|novovic}@utia.cas.cz

Abstract: Recent advances in the statistical methodology for selecting optimal subsets of features for data representation and classification are presented. This chapter attempts to provide a guideline of which approach to choose with respect to the extent of a priori knowledge of the problem. Two basic approaches are reviewed and the conditions under which they should be used are specified. One approach involves the use of the computationally effective Floating search methods. The alternative approach trades off the requirement for a priori information for the requirement of sufficient data to represent the distributions involved. Owing to its nature it is particularly suitable for cases when the underlying probability distributions are not unimodal. The approach attempts to achieve simultaneous feature selection and decision rule inference. According to the criterion adopted there are two variants allowing the selection of features either for optimal representation or discrimination.

7.1 INTRODUCTION

Though the “curse of dimensionality” problem, as called by R. Bellman, can be found in perhaps all the fields of science and engineering, each of these fields has its own specificity and accordingly different ways of treating the problem. Yet the results and methods from one scientific field can be applied not only to

*Supported by the grants of Czech Ministry of Education MŠMT No.VS96063, Czech Acad.Sci. A2075608 and Grant Agency of the Czech Republic No. 402/97/1242.

solve problems in another quite different field, but they can also often enrich its methodology. It is our belief that the novel methods developed recently in the field of statistical pattern recognition to solve the problem of feature selection can enrich the methodology of subset selection used in other fields of artificial intelligence.

Though it would be ideal to make this chapter fully selfcontained, it is impossible to treat here all the details required for their implementation. The reason is that a full size paper has been devoted to each of our three novel methods we are describing here. Therefore, owing to the limited space we have attempted in this chapter to provide just an overview of the methods and techniques recently developed and successfully used in statistical pattern recognition. Instead of overloading the chapter with many mathematical formulas we have rather aimed to give the reader the idea about the basic principle of the methods and their potential usage. References to sources where particular methods are treated in more detail are provided for those interested to employ them.

In the literature one can find abundance of various methods for subset selection, however, for somebody in need of choosing the proper method for his particular problem, it is rather difficult to do so. The optimal choice depends certainly on a number of conditions. The following list of somewhat simplified questions, though not exhaustive, contains perhaps the most important ones:

- Do you aim at optimal data representation or discrimination?
- What is the original dimensionality of your problem?
- What is your *apriori* knowledge of the form of underlying probability structures?
- Do you have a criterion for subsets evaluation appropriate to your knowledge of the problem at hand?
- Do you have a reasonably large set of data with respect to their dimensionality?

With the aim to ease the situation, we are currently developing in our research team a software package for solving the Subset Selection problem. It will be equipped with a kind of expert or consulting system which should guide a less experienced user through the methods included into the package. With respect to the level of problem knowledge (e.g. answers to the above named questions), the user will arrive to the particular method fitting best his knowledge of the problem at hand. Though a number of currently available methods will be included, the core of the package will be formed by the novel methods we have developed ourselves. These methods are briefly described in the sequel. At the end of this chapter we are presenting a simplified example of the flow chart of such a consulting system.

7.2 FEATURE SUBSET SELECTION PROBLEM IN STATISTICAL PATTERN RECOGNITION

7.2.1 Formulation of the Problem

Following the statistical approach to pattern recognition, we assume that a pattern or object described by a real D -dimensional vector $\mathbf{x} = (x_1, x_2, \dots, x_D)^T \in \mathcal{X} \subset \mathcal{R}^D$ is to be classified into one of a finite set of C different classes $\Omega = \{\omega_1, \omega_2, \dots, \omega_C\}$. The patterns are supposed to occur randomly according to some true class conditional probability density functions (pdfs) $p^*(\mathbf{x}|\omega)$ and the respective *a priori* probabilities $P^*(\omega)$. Since the class conditional pdfs and the *a priori* class probabilities are seldom specified in practice, it is necessary to estimate these functions from the training sets of samples with known classification.

In the majority of practical cases, the dimensionality of the pattern descriptor space can be rather high. It is the consequence of the fact that in the design phase it is extremely difficult or practically impossible to evaluate directly the “usefulness” of particular descriptors or input variables. Thus it is desirable and important to include into the original set all the “reasonable” descriptors the designer can think of and to reduce the set later on. The reason is that no subsequent mathematical processing can add information missing in the originally designed measurement set.

Though there are numerous techniques to achieve dimensionality reduction, they can be divided basically either according to their aim into:

1. *dimensionality reduction for optimal data representation*
2. *dimensionality reduction for classification*

or according to the adopted strategy into:

1. *feature selection*
2. *feature extraction*

The differences between dimensionality reduction for representation and for classification are perhaps obvious already from their names. While the first aims to preserve the topological structure of data in a lower-dimensional subspace as much as possible, the second one aims to enhance in a feature subspace the discriminatory power of the selected subset.

On the other hand feature extraction (FE) and feature selection (FS) are sometime confused. As far as FS (or Subset Selection) methods are concerned, they can be defined as follows:

Feature selection strategy: Choose the “best” subset of size d from the given set of D features (by the “best” subset we understand such a subset which optimizes a criterion function $J(\cdot)$ over all possible subsets of d features).

As opposed to FS methods, FE methods are based on some transformation of original features, hence the term feature transformation methods (FT) is also used. The topic of FT is treated elsewhere in this special issue, so we are skipping it. However, though from a purely mathematical point of view feature selection is a special case of feature extraction (the transformation matrix

has '1's on the main diagonal and '0's otherwise), practical differences are so significant that they are worth of mentioning.

We should emphasize that both approaches to the dimensionality reduction - FS and a more general FE have their justification. When deciding between them, one has to be aware of their respective priorities and drawbacks so as to be able to choose the right approach from the point of view of ultimate goals and requirements, concerning the task to be solved.

The priorities and drawbacks of the both approaches from a practical viewpoint are summarized in Table 7.1. From the outlined priorities and drawbacks of the both approaches we can see that their properties are to a certain extent rather contradictory. FS methods will be more suitable in cases when the potential user puts emphasis on preserving the interpretability of original data and prefers decision-making on the basis of meaningful features.

Table 7.1 Comparison of FS and FE from practical viewpoint

	<i>Priorities</i>	<i>Drawbacks</i>
FS	interpretability preservation, data cost acquisition saving	generally lower discriminative power achievable compared to FE
FE	generally higher discriminative power achievable compared to FS	no data acquisition cost saving, loss of interpretability

Furthermore, they will be suitable when one of the goals is to reduce tediousness and costs of data acquisition by finding the data components which can be completely excluded from further acquisition process. For example, most problems of computer-assisted medical decision-making belong to this class of tasks.

7.2.2 State of the Art

In the sequel we shall concentrate on subset selection (or equivalently on FS) problem only, as this is the area to which our own contribution has been acknowledged by the pattern recognition community.

Assuming that a suitable criterion function has been chosen to evaluate the effectiveness of feature subsets, FS is reduced to a search problem that detects an optimal feature subset based on the selected measure. Although an exhaustive search is a sufficient procedure to guarantee the optimality of a solution, in many realistic problems it is computationally prohibitive. In practice, therefore, one has to rely on computationally feasible procedures which avoid the exhaustive search but may yield suboptimal results. A comprehensive list of suboptimal procedures together with the corresponding formulas can be found e.g. in (Devijver and Kittler, 1982). A very useful taxonomy of FS methods currently available in pattern recognition has been recently published by (Jain and Zongker, 1997).

The well known Branch and Bound (B&B) algorithm guarantees to select an optimal feature subset of size d without involving explicit evaluation of all the possible combinations of d measurements. However, the algorithm is applicable only under the assumption that the feature selection criterion used satisfies the monotonicity condition, i.e. that the criterion value increases by adding a new feature. This assumption precludes the use of the error rate as the criterion (it does not fulfill the monotonicity condition). Moreover, the search algorithm becomes computationally prohibitive for problems of high dimensionality. In order to overcome this problem, (Siedlecki and Sklansky, 1988) explored the idea of approximate monotonicity together with other optimization methods like genetic algorithms and Monte Carlo approaches to develop search algorithms which are tolerant to nonmonotonicity.

Our own research and experience with feature selection has led us to the conclusion that **there exists no unique optimal approach** to the problem. Some approaches are more suitable under certain conditions, and different approaches are more appropriate under other conditions. These conditions are related to the level of our **knowledge of the problem**, thus we can talk about “knowledge-based subset selection”. Hence we have attempted to extend the “battery” of available tools by developing several new methods, each of them suitable for different conditions, trying to cover the majority of situations which can be encountered in practice. These methods are introduced in the next pages.

7.3 BASIC SITUATION WITH RESPECT TO PROBLEM KNOWLEDGE

There are perhaps two basic classes of situations with respect to *a priori* knowledge of the form of underlying probability structures:

1. Some *a priori* knowledge is available (at least that pdfs are unimodal) In these cases the use of some probabilistic distance measures (like Mahalanobis, Bhattacharya, etc.) may be appropriate as the evaluation criterion. As pointed out by (Siedlecki and Sklansky, 1988) the error rate is even better (provided it can be reasonably computed). Here we should emphasize that a number of problems from the fields outside pattern recognition may belong to this class of problems. For this type of situations we have developed a family of Floating Search algorithms which yield close to optimal solution, are computationally effective (facilitating FS in high dimensional problems) and do not require the fulfilment of monotonicity condition. In a recent comparative study of currently available subset search strategies carried out by (Jain and Zongker, 1997) were the Floating Search algorithms evaluated as the most powerful ones.

2. No *a priori* knowledge is available

We cannot even assume that pdfs are unimodal (or suspect they are multimodal). The only source of available information is provided by the training data. Feature selection in such a case becomes a very challenging problem. The early solutions of this problem suffer from serious shortcomings. For these

situations we have developed two new approaches aimed to cope reasonably well in such circumstances, conceptually very different from those mentioned above. They are based on approximating unknown conditional pdfs by finite mixtures of a special type and are discussed later.

7.4 FLOATING SEARCH METHODS

Various search strategies are used to find the subset of features optimizing an adopted criterion, once this criterion has been chosen. They range from simple but popular ones, like sequential forward (SFS) and sequential backward (SBS) selection, to more sophisticated but computationally more difficult ones. The so called “nesting of feature subsets” may rapidly result in deteriorating performance of both the SFS and SBS algorithms. It is worth of mentioning that this also applies to the sequential algorithms used to solve multiple stepwise regression problems which are included in almost all popular statistical analysis packages.

This can be *partially* overcome by employing either the so-called (l, r) (in one step include l and exclude r features) or generalized (l, r) algorithms ((Devijver and Kittler, 1982)) which involve a successive feature set augmentation and depletion process. Consequently, the resulting dimensionality in respective stages of both algorithms is fixed depending on the prespecified values of l and r . Unfortunately, there is no theoretical way of predicting the values of l and r so as to achieve the best feature set.

To counteract these problems we developed recently a novel family of search strategies based on the principle of iterative search in both directions, but as opposed to the bidirectional search proposed in (Siedlecki and Sklansky, 1988), exploiting a flexible level of repeated backtracking. Instead of fixing the values of l and r , these values are allowed to “float”, i.e. to flexibly change so as to approximate the optimal solution as much as possible. Consequently, the resulting dimensionality in respective intermediate stages of the algorithm is not changing monotonously but is actually “floating” up and down. Because of this “floating” characteristics, the two methods have been named **floating search methods** ((Pudil et al., 1994)).

Although they both switch between feature inclusion and exclusion, they are based on two different algorithms according to the dominant direction of the search:

Sequential forward floating search (SFFS)

- the search dominantly in the forward direction

Sequential backward floating search (SBFS)

- the search dominantly in the forward direction

A more exact description of the algorithms is given in (Pudil et al., 1994). A simplified flow chart of the SFFS algorithm is given in Figure 7.1:

The terminating condition $k = d + \delta$ in the flowchart means that in order to fully utilize the properties of SFFS search we should not stop the algorithm immediately after it reaches for the first time the dimensionality d . By leaving it to float up and back a bit further, the potential of the algorithm is better

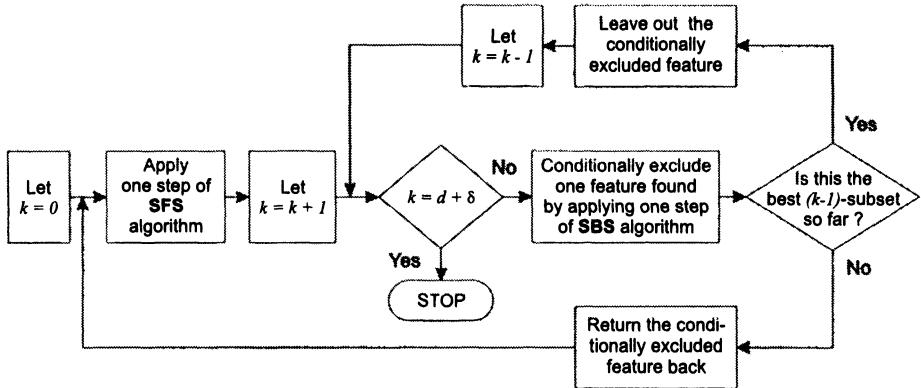


Figure 7.1 Flow Chart of SFFS algorithm

utilized and a subset of cardinality d outperforming the first one achieved is usually found. In practice we can let the algorithm either go up to the original dimensionality D , or if it is too large, then the value of δ can be determined heuristically (e.g. according to the value of maximum number of backtracking steps prior to reaching d first time).

Unlike the (l, r) and generalized (l, r) algorithms in which factors such as the net change in the size of the current feature set, and especially the amount of computational time, are governed by the values of l and r , the SFFS and SBFS methods are not restricted by these factors. By means of conditional “floating down and up” both the methods are freely allowed to correct wrong decisions made in the previous steps so as to approximate the optimal solution as much as possible.

The results achieved so far on various sets of data demonstrate clearly a great potential of floating search strategies ((Pudil et al., 1994; F.J.Ferri et al., 1994; Jain and Zongker, 1997)). Floating search methods yielded in all the cases better results than GSFS(l) and GSBS(l) algorithms(including or deleting l features in a single step) or than (l, r) algorithms. The worst proved to be the Max-Min algorithm which is computationally appealing but theoretically ill-founded, as we have proved ((Pudil et al., 1993)). The subsets found by Floating Search were practically identical with those found by a computationally tedious “exhaustive” combinatorial search. Comparison with the Branch and Bound method showed that Floating Search is much faster ((Pudil et al., 1994; Jain and Zongker, 1997)).

Generally, though of heuristic nature, Floating Search methods provide either the optimal or a close to optimal solution, but also require much less computational time than the Branch and Bound method and most other currently used suboptimal strategies. The computational efficiency allows the use of floating search even for large scale FS problems. To test the different algorithms in a large search space we also considered a document recognition problem. It involved discrimination between correct and defective records of banking doc-

uments consisting of 360 optical measurements. The Floating Search methods outperformed in this problem the traditional suboptimal search methods and also yielded better results than the genetic algorithms, particularly for higher dimensions ((F.J.Ferri et al., 1994)).

Moreover, as opposed to the branch and bound method, the floating search methods are also tolerant to deviations from monotonic behavior of the feature selection criterion function. It makes them particularly suited in conjunction with nonmonotonic FS criterion like the error rate of the classifier which according to a number of researchers seem to be the only legitimate criterion for feature subset evaluation.

7.5 FEATURE SELECTION BY MODIFIED GAUSSIAN MIXTURES

Now we shall address the feature selection problem arising **when we have the data but no other information** which occurs in a number of real situations. Though various nonparametric methods of classification are available, none of them is problem free and universal.

In order to facilitate the solution under these conditions, we have developed a completely new approach that can serve the multiple goal of:

1. learning the structure of multivariate distributions,
2. identifying the most important variables and thus facilitating the dimensionality reduction,
3. deriving automatically a decision rule based on the selected features

The approach is based on approximating the unknown class conditional distributions by finite mixtures of parameterized densities of a special type. In terms of the required computer storage it is considerably more efficient than nonparametric pdf estimation methods.

As far as the FS problem is concerned, there are two different methods with respect to the criterion employed. Though both methods exploit a common basic approximation model, the way of selecting features is completely different. The same holds for their purpose and optimal applicability. Since it is not possible to present a detailed formalized description of the approach in this chapter, the reader is referred to respective original sources. However, we attempt to provide at least an outline of the common statistical model used for both the methods and then the respective methods of FS.

7.5.1 Modified Gaussian Mixture Model

The approach to feature selection taken here is to approximate the ω th class density of \mathbf{x} by the following modified Gaussian model with latent structure or modified latent subclass Gaussian model ((Pudil et al., 1995; Novovičová et al., 1996; Novovičová and Pudil, 1997)):

$$p(\mathbf{x}|\omega) = \sum_{m=1}^{M_\omega} \alpha_m^\omega g_0(\mathbf{x}|\theta_0)g(\mathbf{x}|\theta_m^\omega, \theta_0, \Phi), \quad \mathbf{x} \in \mathcal{X}, \quad (7.1)$$

where α_m^ω is the mixing probability for the m th subclass, $\sum_{m=1}^{M_\omega} \alpha_m^\omega = 1$. The

function g_0 is a nonzero "background" density common to all classes and functions $g(\mathbf{x}|\theta_m^\omega, \theta_0, \Phi)$ include structural parameters ϕ_i :

$$g_0(\mathbf{x}|\theta_0) = \prod_{i=1}^D f_i(x_i|\theta_{0i}), \quad g(\mathbf{x}|\theta_m^\omega, \theta_0, \Phi) = \prod_{i=1}^D \left[\frac{f_i(x_i|\theta_{mi}^\omega)}{f_i(x_i|\theta_{0i})} \right]^{\phi_i}, \quad (7.2)$$

$$\theta_0 = \{\theta_{0i}\}_{i=1}^D, \quad \theta_m^\omega = \{\theta_{mi}^\omega\}_{i=1}^D, \quad \phi_i = \{0, 1\}, \quad \Phi = \{\phi_i\}_{i=1}^D.$$

The univariate function f is assumed to be from a family of univariate normal densities $\{f(\xi|\theta) = f(\xi|\mu, \sigma), \xi \in \mathcal{R}, \mu \in \mathcal{R}, \sigma^2 \in (0, \infty)\}$. Our model is based on the idea to posit a common "background" normal density for all classes and to express each class pdf as a mixture of a product of this "background" density with a class-specific function defined on a subspace of the feature vector space. This subspace is chosen by means of the parameters ϕ_i and the same subspace of \mathcal{X} for each component density is used in all classes. Any specific univariate function $f_i(x_i|\theta_{mi}^\omega)$ is substituted by the respective "background" density $f_i(x_i|\theta_{0i})$ whenever ϕ_i is zero. In this way the binary parameters ϕ_i can be looked upon as *control variables* due to that the structure of the mixture (7.1) can be controlled by means of that parameters.

For any choice of ϕ_i the finite mixture (7.1) can be rewritten by using (7.2) as

$$p(\mathbf{x}|\alpha^\omega, \theta^\omega, \theta_0, \Phi) = \sum_{m=1}^{M_\omega} \alpha_m^\omega \prod_{i=1}^D [f_i(x_i|\theta_{0i})]^{(1-\phi_i)} [f_i(x_i|\theta_{mi}^\omega)]^{\phi_i}, \quad (7.3)$$

$$\alpha^\omega = \{\alpha_m^\omega\}_{m=1}^{M_\omega}, \quad \theta^\omega = \{\mu_m^\omega\}_{m=1}^{M_\omega}$$

Setting some $\phi_i = 1$, we replace the function $f_i(x_i|\theta_{0i})$ in the product in (7.3) by $f_i(x_i|\theta_{mi}^\omega)$ and introduce a new independent parameter θ_{mi}^ω in the mixture (7.3). The actual number of involved parameters we specify by $\sum_{i=1}^D \phi_i = \gamma$, $1 \leq \gamma \leq D$.

The parameter sets $\alpha^\omega, \theta^\omega, \theta_0, \Phi$ are unknown and can be estimated from the training sets. Suppose that the ω th training set is \mathbf{X}_ω and $|\mathbf{X}_\omega|$ is the number of training data from the ω th class. Then the log-likelihood function for the data is

$$L(\alpha, \theta, \theta_0, \Phi) = \sum_{\omega \in \Omega} \frac{P(\omega)}{|\mathbf{X}_\omega|} \sum_{\mathbf{x} \in \mathbf{X}_\omega} \log p(\mathbf{x}|\alpha^\omega, \theta^\omega, \theta_0, \Phi), \quad (7.4)$$

$$\alpha = \{\alpha^\omega\}_{\omega \in \Omega}, \quad \theta = \{\theta^\omega\}_{\omega \in \Omega}.$$

The notation $\sum_{\mathbf{x} \in \mathbf{X}_\omega}$ means summing over all observations belonging to the ω th training data set \mathbf{X}_ω . The EM ("Expectation-Maximization") iterative algorithm ((Redner and Walker, 1984)) provides a convenient method for maximizing $L(\cdot)$. For a given Φ the EM steps are

$$p(m|\mathbf{x}, \omega) = Prob(\mathbf{x} \in m\text{th subclass of class } \omega | \mathbf{x}, \omega) = \frac{\alpha_m^\omega g(\mathbf{x}|\theta_m^\omega, \theta_0, \Phi)}{\sum_{j=1}^{M_\omega} \alpha_j^\omega g(\mathbf{x}|\theta_j^\omega, \theta_0, \Phi)},$$

$$\hat{\alpha}_m^\omega = \frac{1}{|\mathbf{X}_\omega|} \sum_{\mathbf{x} \in \mathbf{X}_\omega} p(m|\mathbf{x}, \omega), \quad \sum_{m=1}^{M_\omega} \hat{\alpha}_m^\omega = 1$$

$$\hat{\mu}_{mi}^\omega = \frac{1}{|\mathbf{X}_\omega| \hat{\alpha}_m^\omega} \sum_{\mathbf{x} \in \mathbf{X}_\omega} x_i p(m|\mathbf{x}, \omega), \quad (\hat{\sigma}_{mi}^\omega)^2 = \frac{1}{|\mathbf{X}_\omega| \hat{\alpha}_m^\omega} \sum_{\mathbf{x} \in \mathbf{X}_\omega} (x_i - \hat{\mu}_{mi}^\omega)^2 p(m|\mathbf{x}, \omega)$$

$$\hat{\mu}_{0i} = \sum_{\omega \in \Omega} P(\omega) \sum_{m=1}^{M_\omega} \hat{\alpha}_m^\omega \hat{\mu}_{mi}^\omega, \quad (\hat{\sigma}_{0i})^2 = \sum_{\omega \in \Omega} P(\omega) \sum_{m=1}^{M_\omega} \hat{\alpha}_m^\omega [(\hat{\sigma}_{mi}^\omega)^2 + (\hat{\mu}_{mi}^\omega - \hat{\mu}_{0i})^2].$$

7.5.2 Feature Selection for Approximation

Two methods have been derived depending on different criteria for features evaluation ((Pudil et al., 1995; Novovičová et al., 1996; Novovičová and Pudil, 1997)). The first method selects a feature subset X_d by choosing Φ_d (i.e. parameter vector Φ restricted to have just d components equal to 1 and $D - d$ components equal to 0) such that the best approximation is obtained. In the **"approximation" method** for FS the criterion we use for measuring the error resulting from approximating the true pdf $p^*(\mathbf{x}|\omega)$ by $p(\mathbf{x}|\alpha_\omega, \theta_\omega, \theta_0, \Phi)$ for all $\omega \in \Omega$ is a mixture, in the true proportions $P(\omega_1), \dots, P(\omega_C)$, of the Kullback-Leibler distances between the true and the postulated class densities of \mathbf{x} . A very simplified float chart of that method can be found in (Pudil and Novovičová, 1998).

The "approximation" method has the following interesting characteristics:

1. owing to the convenient form of the postulated model the "contribution" $Q(\Phi)$ of a feature subset to the chosen criterion is the sum of individual contributions $\hat{Q}_i = \sum_{\omega \in \Omega} \sum_{m=1}^{M_\omega} P(\omega) \hat{\alpha}_m^\omega \log \frac{(\hat{\sigma}_{0i})^2}{(\hat{\sigma}_{mi}^\omega)^2}$, which can be assessed independently for each feature: $Q(\Phi) = \sum_{i=1}^D \phi_i \hat{Q}_i$
2. only the operation of ranking of individual feature contributions is therefore required (without any search procedure) in order to obtain a required subset of d features.

Though the "approximation method" yielded very good results in many problems ranging from image analysis to classification, we should be aware of the fact that the features are selected with respect to their "approximation" or "representation" quality, which may not in particular cases coincide with their "discriminative" quality. Consequently, the method is particularly convenient for the problems of multivariate data representation in a lower-dimensional space or pattern interpretation. It is also applicable to multiclass problems. For the cases when the discrimination between classes is the primary goal, the following "divergence" method has been developed.

7.5.3 Feature Selection for Discrimination

In order to select those features that are most useful in describing differences between two possible classes, we have developed another method ((Novovičová

et al., 1996; Novovičová and Pudil, 1997)) for feature selection. Similarly to the "approximation" method it utilizes the same general model for approximating unknown class conditional pdfs by finite mixtures of parameterized densities (7.1). However, in this case the Kullback's J-divergence (see e.g. (Boekee and Lubbe, 1979)) between two classes defined in terms of a posterior probabilities (or equivalently the Kullback-Leibler measures of discriminatory information between two classes mixed in the proportions in which the classes truly occur) is used as the appropriate evaluation criterion. The goal of the method is to maximize the divergence discrimination, hence the name of "divergence" method.

The proposed approach is especially suitable for multimodal data and is restricted at the moment to two classes. The two interesting characteristics specified above for the approximation method hold for the divergence method too ((Novovičová et al., 1996)).

7.5.4 Properties of Approximation and Divergence Methods

An important characteristic of our approach is that it effectively partitions the feature set X of all D features into two disjunct subsets X_d and $X - X_d$, where the joint distribution of the features from $X - X_d$ is common to all the classes and constitutes the background distribution, as opposed to the features forming X_d , which are significant for discriminating the classes. The joint distribution of these features constitutes the "specific" distribution defined in (7.2). According to these features alone, a new pattern \mathbf{x} is classified into one of C classes.

It has been proved that just under this partition of the set X the following holds:

1. For the "approximation" method: the Kullback-Leibler distance between the true and the postulated class conditional pdfs is minimized.
2. For the "divergence" method: the Kullback J-divergence is maximized.

For those interested in classification problems it may be of interest that our approach is not only a classification procedure but also a data reduction tool. The modified mixture (7.3) reduces also the computational complexity of the corresponding Bayes decision rule. We can represent multiclass data by d features, where $d < D$ if $\phi_i = 1$ for $i = 1, \dots, d$. Given the approximations $p(\mathbf{x}|\hat{\alpha}_\omega, \hat{\theta}_\omega, \hat{\theta}_0, \hat{\Phi}_d)$ it can be easily seen that the background pdf g_0 may be reduced in the Bayes decision rule. Thus we may classify the observation of \mathbf{x} according to the pseudo-Bayes decision rule:

decide that \mathbf{x} is from class ω_l if

$$P(\omega_l) \sum_{m=1}^{M_\omega} \hat{\alpha}_m^{\omega_l} \prod_{k=1}^d f_i(x_{ik} | \hat{\theta}_{mi_k}^{\omega_l}) = \max_{j=1, \dots, C} \{P(\omega_j) \sum_{m=1}^{M_\omega} \hat{\alpha}_m^{\omega_j} \prod_{k=1}^d f_i(x_{ik} | \hat{\theta}_{mi_k}^{\omega_j})\}.$$

We use the term "pseudo-Bayes" as the rule is applied in a lower-dimensional subspace corresponding to the selected subset and, furthermore, the true conditional pdfs are in the formulas substituted by their approximations. It means

that a new feature vector \mathbf{x} is classified into one of the classes according to only d features x_{i_1}, \dots, x_{i_d} .

7.6 EXPERIMENTAL RESULTS

Experiments on various sets of both simulated and real-world data have shown that both the methods outperform “conventional” methods which assume Gaussian class conditional pdfs.

7.6.1 Texture Discrimination Problem

A 26-dimensional texture discrimination problem was considered in the framework of a project at the University of Surrey, U.K., concerning the classification of marble specimens from two classes (“bianco castilla” and “rosa baveno”).

The results obtained with the approximation method of subset selection for mixtures consisting of 2, 3 and 4 components are shown in the following Table 7.2. The quality of the selected subset of features has been tested by the classification error on the independent test set (not used for training). The above discussed pseudo-Bayes decision rule derived simultaneously by the method and based only on the selected features has been used. The estimate of the probability of error for the subset of i selected features is denoted $\text{Pe}(X_i)$.

In order to compare our approach with a “conventional” one, a multivariate Gaussian classifier was used as an alternative. The features have been selected in this case, using the Bhattacharyya distance for the subset evaluation, by means of the optimal Branch-and-Bound (B&B) algorithm that finds the optimal solution to maximize the criterion. Please, note that though for this another approach \tilde{X}_i is also a subset of the cardinality i , it is generally different from the subset X_i selected by the approximation method.

Table 7.2 Classifier performance on different feature subset size of image data

METHOD	<i>Estimate of probability of classification error</i>			
1. Pseudo-Bayes classifier based on Approx. method	$\text{Pe}(X_6)$	$\text{Pe}(X_{10})$	$\text{Pe}(X_{14})$	$\text{Pe}(X_{26})$
Mixture of 2 components	0.100	0.064	0.030	0.030
Mixture of 3 components	0.065	0.028	0.016	0.007
Mixture of 4 components	0.055	0.012	0.007	0.007
2. Gaussian classifier with features selected by B&B	$\text{Pe}(\tilde{X}_6)$	$\text{Pe}(\tilde{X}_{10})$	$\text{Pe}(\tilde{X}_{14})$	$\text{Pe}(\tilde{X}_{26})$
	0.235	0.287	0.169	0.169

From Table 7.2, it is obvious that the Gaussian classifier with features selected by B&B algorithm gives rather big errors as apparently the assumption that the distributions are unimodal Gaussian is not appropriate. In contrast,

when a mixture of normal densities is used, a much smaller error rate is obtained. A substantial dimensionality reduction can be made without significantly deteriorating the classifier performance. In other words, redundant features have been detected especially with the mixture of four components. This can be clearly seen for the subsets of 10 and 14 features.

7.6.2 Speech Recognition Problem

The data used to train, test and compare the proposed method was a set of 1418 pattern vectors of the utterances "YES" and "NO" spoken over the public switched telephone network. Each 15-dimensional feature vector contained 5 segments of 3 features derived by low order linear prediction analysis. From this set, 798 samples were used for training and the remaining 620 samples to form the test set. Both sets contain nearly equal number of samples for each pattern class. The data was supplied by British Telecom. The results of the experiment are shown in Figure 7.2.

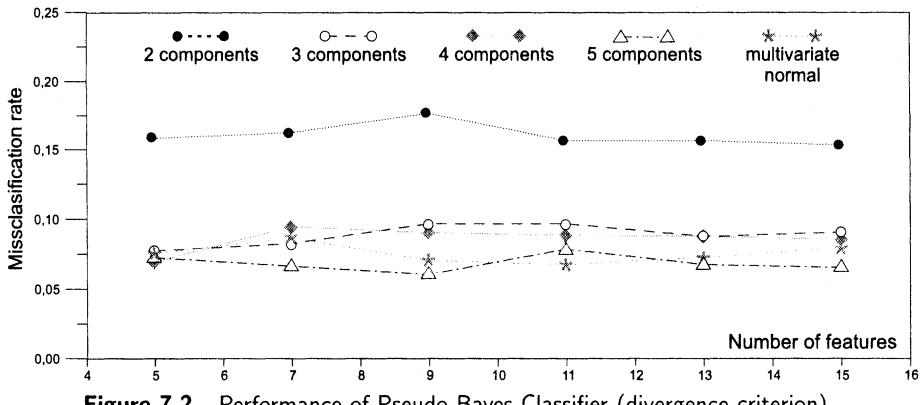


Figure 7.2 Performance of Pseudo-Bayes Classifier (divergence criterion)

A detailed analysis has shown that the data lies in narrow regions along parallel hyperplanes, i.e. the data are highly correlated and unimodal and it is therefore perfectly suitable to use the Gaussian classifier in this case. The problem was picked up deliberately as it is very difficult for the approach based on the approximation mixture model. To approximate these distributions by a mixture of independent Gaussian distributions properly would require a higher number of mixture components. However, owing to a small sample size of the training set, this is not possible. For this reason the "approximation" method failed to provide good results in this case.

On the other hand, despite the above mentioned unfavorable conditions, the "divergence" method provided the results comparable or even better than the "normal" approach. Note again that the "normal" approach consisted of selecting features by B&B method and using the Gaussian classifier.

From Figure 7.2 it can be seen that even in the case of divergence criterion the mixture of 2 independent normal densities was not sufficient to accurately approximate the unknown distributions. This insufficiency is apparent from the

classification error rate which was approximately twice as high than in the case of more components. With the mixture of 3, 4 and 5 components comparable (for 5 components even better) results to that of the multivariate normal model have been obtained.

7.7 SUBSET SELECTION GUIDE

Though it is certainly too premature to claim a real development of the knowledge guided approach to subset selection, the presented methods can serve as a starting point to achieve such an ambitious goal. To implement it, we are developing an integrated environment which would incorporate a whole family of methods together with a sort of expert system. It should guide a user according to his goals and the degree of the problem knowledge he has to the semi-automatic choice of the most suitable method.

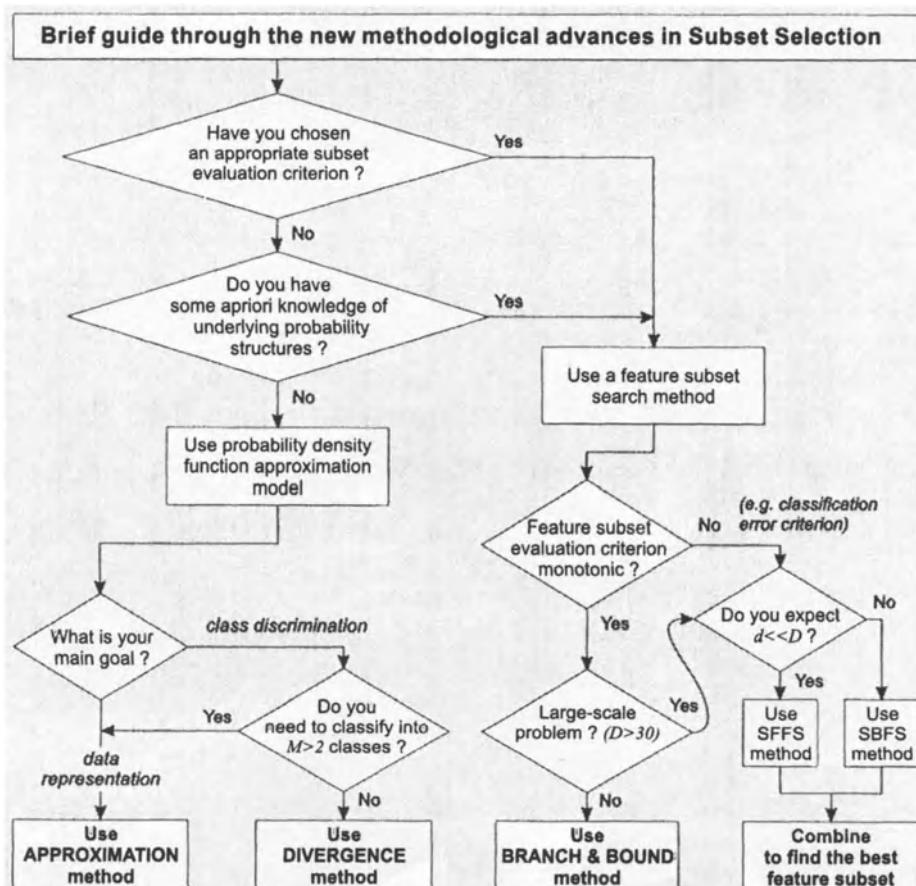


Figure 7.3 Flow chart of "prototype" Subset Selection Guide

We hope that the flow chart of the “prototype” Subset Selection Guide presented in Figure 7.3 will give the reader the idea how it is being built.

The flow-chart and the questions are certainly very simplified here. Nevertheless, we would like to draw readers’ attention to the fact that in a great number of real situations there exists a generally accepted and appropriate criterion to evaluate the selected subset for a given problem (e.g. for multiple stepwise regression analysis). In all these cases the subset search techniques may be used as depicted in the flow-chart.

Finally, we should stress that accordingly to the flow chart title emphasizing “new methodological advances in Subset Selection”, it contains only our own new or recently developed methods, though the FS Guide system will eventually contain also other methods.

Obviously, there are some research issues yet to be pursued, like a possible combination of genetic algorithms and floating search, however, we believe that the research communities in various branches of artificial intelligence may benefit already from the currently achieved results.

Acknowledgments

The authors would like to thank to P. Somol for preparing the drawings.

References

- Boekee, D. E. and Lubbe, J. C. A. V. D. (1979). Some aspects of error bounds in feature selection. *Pattern Recognition*, 11:353–360.
- Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. Prentice-Hall.
- F.J.Ferri, P.Pudil, M.Hatef, and J.Kittler (1994). Comparative study of technique for large-scale features selection. In *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*; edited by E.S. Gelsema and L.N.Kanal, pages 403–413, North-Holland Elsevier.
- Jain, A. K. and Zongker, D. (1997). Feature selection: Evaluation, application and small sample performance. *IEEE Transactions on PAMI*, 19:153–158.
- Novovičová, J. and Pudil, P. (1997). *Dealing With Complexity: Neural Network Approach*, chapter Feature Selection and Classification by Modified Model with Latent Structure. Springer Verlag.
- Novovičová, J., Pudil, P., and Kittler, J. (1996). Divergence based feature selection for multimodal class densities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:218–223.
- Pudil, P. and Novovičová, J. (1998). Novel methods for subset selection with respect to problem knowledge. *IEEE Intelligent Systems - Special Issue on Feature Transformation and Subset Selection*, in print.
- Pudil, P., Novovičová, J., Choakjarernwanit, N., and Kittler, J. (1993). An analysis of the max-min approach to feature selection. *Pattern Recognition Letters*, 14(11):841–847.

- Pudil, P., Novovičová, J., Choakjarernwanit, N., and Kittler, J. (1995). Feature selection based on the approximation of class densities by finite mixtures of special type. *Pattern Recognition*, 28(9):1389–98.
- Pudil, P., Novovičová, J., and Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125.
- Redner, R. A. and Walker, H. F. (1984). Mixture densities, maximum likelihood and the EM algorithm. *SIAM J. Appl. Math.*, 26(2):195–239.
- Siedlecki, W. and Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):197–220.

8

FEATURE SUBSET SELECTION USING A GENETIC ALGORITHM

Jihoon Yang and Vasant Honavar

AI Research Group, Department of Computer Science
226 Atanasoff Hall, Iowa State University
Ames, IA 50011, U.S.A.

{yang|honavar}@cs.iastate.edu

Abstract: Pattern classification and knowledge discovery problems require selection of a subset of features to represent the patterns to be classified. This is due to the fact that the performance of the classifier and the cost of classification are sensitive to the choice of the features used to construct the classifier. Genetic algorithms offer an attractive approach to find near-optimal solutions to such optimization problems. This chapter presents an approach to feature subset selection using a genetic algorithm. Our experiments demonstrate the feasibility of this approach to feature subset selection in the automated design of neural networks for pattern classification and knowledge discovery.

8.1 INTRODUCTION

Many practical pattern classification tasks (e.g., medical diagnosis) require learning of an appropriate classification function that assigns a given input pattern (typically represented using a vector of attribute or feature values) to one of a finite set of classes. The choice of features, attributes, or measurements used to represent patterns that are presented to a classifier affect (among other things):

- The accuracy of the classification function that can be learned using an inductive learning algorithm: The features used to describe the patterns implicitly define a pattern language. If the language is not expressive enough, it would fail to capture the information that is necessary for classification and hence regardless of the learning algorithm used, the accuracy of the classification function learned would be limited by this lack of information.

- The time needed for learning a sufficiently accurate classification function: For a given representation of the classification function, the features used to describe the patterns implicitly determine the search space that needs to be explored by the learning algorithm. An abundance of irrelevant features can unnecessarily increase the size of the search space, and hence the time needed for learning a sufficiently accurate classification function.
- The number of examples needed for learning a sufficiently accurate classification function: All other things being equal, the larger the number of features used to describe the patterns in a domain of interest, the larger is the number of examples needed to learn a classification function to a desired accuracy (Langley, 1995; Mitchell, 1997).
- The cost of performing classification using the learned classification function: In many practical applications e.g., medical diagnosis, patterns are described using observable symptoms as well as results of diagnostic tests. Different diagnostic tests might have different costs as well as risks associated with them. For instance, an invasive exploratory surgery can be much more expensive and risky than say, a blood test.
- The comprehensibility of the knowledge acquired through learning: A primary task of an inductive learning algorithm is to extract *knowledge* (e.g., in the form of classification rules) from the training data. Presence of a large number of features, especially if they are irrelevant or misleading, can make the knowledge difficult to comprehend by humans. Conversely, if the learned rules are based on a small number of relevant features, they would much more concise and hence easier to understand, and use.

This presents us with a *feature subset selection problem* in automated design of pattern classifiers. The feature subset selection problem refers the task of identifying and selecting a useful subset of features to be used to represent patterns from a larger set of often mutually redundant, possibly irrelevant, features with different associated measurement costs and/or risks. An example of such a scenario which is of significant practical interest is the task of selecting a subset of clinical tests (each with different financial cost, diagnostic value, and associated risk) to be performed as part of a medical diagnosis task. Other examples of feature subset selection problem include large scale data mining applications, power system control (Zhou et al., 1997), construction of user interest profiles for text classification (Yang et al., 1998a) and sensor subset selection in the design of autonomous robots (Balakrishnan and Honavar, 1996).

8.2 RELATED WORK

A number of approaches to feature subset selection have been proposed in the literature. (See (Siedlecki and Sklansky, 1988; Doak, 1992; Langley, 1994; Dash and Liu, 1997) for surveys). These approaches involve searching for an optimal subset of features based on some criteria of interest. Feature subset selection problem can be viewed as a special case of the *feature weighting* problem. It

involves assigning a real-valued weight to each feature. The weight associated with a feature measures its relevance or significance in the classification task (Cost and Salzberg, 1993; Wettschereck et al., 1995). If we restrict the weights to be binary valued, the feature weighting problem reduces to the feature subset selection problem. The focus of this chapter is on feature subset selection.

Let $\mu(S)$ be a performance measure that is used to evaluate a feature subset S with respect to the criteria of interest (e.g., cost and accuracy of the resulting classifier). Feature subset selection problem is essentially an optimization problem which involves searching the space of possible feature subsets to identify one that is optimal or near-optimal with respect to μ . Feature subset selection algorithms can broadly be classified into three categories according to the characteristics of the search strategy employed.

8.2.1 Feature Subset Selection Using Exhaustive Search

In this approach, the candidate feature subsets are evaluated with respect to the performance measure μ and an *optimal* feature subset is found using exhaustive search. The **Focus** algorithm (Almuallim and Dietterich, 1994) employs the breadth-first search algorithm to find the minimal combination of features sufficient to construct a hypothesis that is consistent with the training examples. The algorithm proposed by (Sheinvald et al., 1990) uses the *minimum description length* criterion to select an optimal feature subset using exhaustive enumeration and evaluation of candidate feature subsets. Exhaustive search is computationally infeasible in practice, except in those rare instances where the total number of features is quite small.

8.2.2 Feature Subset Selection Using Heuristic Search

Since exhaustive search over all possible subsets of a feature set is not computationally feasible in practice, a number of authors have explored the use of *heuristics* for feature subset selection, often in conjunction with branch and bound search, a technique that is well-known in combinatorial optimization and artificial intelligence. *Forward selection* and *backward elimination* are the most common sequential branch and bound search algorithms used in feature subset selection (Narendra and Fukunaga, 1977; Foroutan and Sklansky, 1987). Forward selection starts with an empty feature set and adds a feature at a time, at each stage choosing the addition that most increases μ . Backward elimination starts with the entire feature set and at each step drops the feature whose absence least decreases μ . Both forward and backward selection procedures are optimal at each stage, but are unable to anticipate complex *interactions* between features that might affect the performance of the classifier. A related approach, called the *exchange strategy* starts with an initial feature subset (perhaps found by forward selection or backward elimination) and then tries to exchange a feature in the selected subset with one of the features that is outside it. We can often find a feature subset that is guaranteed to be the best for a given size of the feature subset without considering all possible subsets

using branch and bound search (Narendra and Fukunaga, 1977) if we assume that μ is monotone. That is, adding features is guaranteed to not decrease μ . It is worth pointing out that in many practical pattern classification scenarios, the monotonicity assumption is not satisfied. For example, addition of irrelevant features (e.g., social security numbers in medical records in a diagnosis task) can significantly worsen the generalization accuracy of a decision tree classifier (Mitchell, 1997). Furthermore, feature subset selection techniques that rely on the monotonicity of the performance criterion, although they appear to work reasonably well with linear classifiers, can exhibit poor performance with non-linear classifiers such as neural networks (Ripley, 1996).

Five greedy hillclimbing procedures (with different sequential search methods) for obtaining good generalization with decision tree construction algorithms (ID3 and C4.5) (Mitchell, 1997) were proposed in (Caruana and Freitag, 1994). In related work, (John et al., 1994) used both forward selection and backward elimination to minimize the cross validation error of decision tree classifiers; (Kohavi, 1994) used hillclimbing and best-first search for feature subset selection for decision tree classifiers. (Koller and Sahami, 1996; Koller and Sahami, 1997) used forward selection and backward elimination to select a feature that is subsumed by the remaining features (determined by the *Markov blanket*, the set of features that render the selected feature conditionally independent of the remaining features) for constructing Naive Bayesian (Mitchell, 1997) and decision tree classifiers. The *Preset* algorithm (Modrzejewski, 1993) employs the *rough set theory* to select a feature subset by rank ordering the features to generate a minimal decision tree.

8.2.3 Feature Subset Selection Using Randomized Search

Randomized algorithms make use of randomized or probabilistic (as opposed to deterministic) steps or sampling processes. Several researchers have explored the use of such algorithms for feature subset selection. The *Relief* algorithm (Kira and Rendell, 1992) assigns weights to features (based on their estimated effectiveness for classification) using the randomly sampled instances. Features whose weights exceed a user-determined threshold are selected in designing the classifier. Several extensions of *Relief* have been introduced to handle noisy or missing features as well as multi-category classification (Kononenko, 1994). A randomized hillclimbing search for feature subset selection for nearest neighbor classifiers (Cover and Hart, 1967; Dasarathy, 1991) was proposed in (Skalak, 1994). The *LVF* and *LVW* algorithms (Liu and Setiono, 1996b; Liu and Setiono, 1996a) are randomized algorithms that generate several random feature subsets and pick the one that has the least number of *unfaithful* patterns in the space defined by the feature subset (*LVF*) or the one that has the lowest error using a decision tree classifier (*LVW*) giving preference to smaller feature subsets. (Two patterns are said *unfaithful* if they have the same feature values but different class labels). Several authors have explored the use of randomized population-based heuristic search techniques such as genetic algorithms (GA) for feature subset selection for decision tree and nearest neighbor classifiers (Siedlecki and

Sklansky, 1989; Brill et al., 1992; Richeldi and Lanzi, 1996) or rule induction systems (Vafaie and De Jong, 1993). A related approach used *lateral feedback* networks (Guo, 1992) to evaluate feature subsets (Guo and Uhrig, 1992). Feature subset selection techniques that employ genetic algorithms do not require the restrictive monotonicity assumption. They also readily lend themselves to the use of multiple selection criteria. This makes them particularly attractive in the design of pattern classifiers in many practical scenarios.

8.2.4 Filter and Wrapper Approaches to Feature Subset Selection

Feature subset selection algorithms can also be classified into two categories based on whether or not feature selection is done independently of the learning algorithm used to construct the classifier. If feature selection is performed independently of the learning algorithm, the technique is said to follow a *filter* approach. Otherwise, it is said to follow a *wrapper* approach (John et al., 1994). While the filter approach is generally computationally more efficient than the wrapper approach, its major drawback is that an optimal selection of features may not be independent of the inductive and representational biases of the learning algorithm that is used to construct the classifier. The wrapper approach on the other hand, involves the computational overhead of evaluating candidate feature subsets by executing a selected learning algorithm on the dataset represented using each feature subset under consideration. This is feasible only if the learning algorithm used to train the classifier is relatively fast. Figure 8.1 summarizes the filter and wrapper approaches. The approach to feature subset selection proposed in this chapter is an instance of the wrapper approach. It utilizes a genetic algorithm for feature subset selection. Feature subsets are evaluated by computing the generalization accuracy of (and optionally cost of features used in) the neural network classifier constructed using a computationally efficient neural network learning algorithm called DistAl (Yang et al., 1998b).

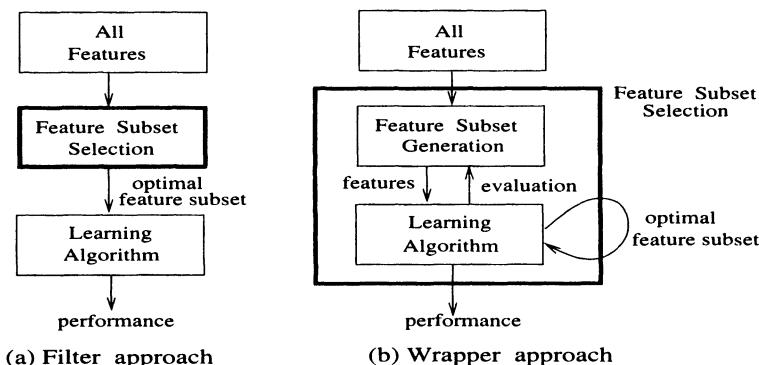


Figure 8.1 Two approaches to feature subset selection based on the incorporation of the learning algorithm.

8.3 FEATURE SELECTION USING A GENETIC ALGORITHM FOR NEURAL NETWORK PATTERN CLASSIFIERS

Feature subset selection in the context of many practical problems (e.g., diagnosis) presents an instance of a multi-criteria optimization problem. The multiple criteria to be optimized include the accuracy of classification, cost and risk associated with classification which in turn depends on the selection of features used to describe the patterns. Genetic algorithms offer a particularly attractive approach for multi-criteria optimization.

Neural networks offer an attractive framework for the design of trainable pattern classifiers for real-world real-time pattern classification tasks on account of their potential for parallelism and fault and noise tolerance (Honavar, 1998a).

While genetic algorithms are generally quite effective for rapid global search of large search spaces in difficult optimization problems, neural networks offer a particularly attractive approach to finetuning promising solutions once they have been identified. Thus, it is attractive to explore combinations of global and local search techniques in the solution of difficult design or optimization problems (Mitchell, 1996). Against this background, the use of genetic algorithms for feature subset selection in the design of neural network pattern classifiers is clearly of interest.

This chapter explores GADistAI, a wrapper-based multi-criteria approach to feature subset selection using a genetic algorithm in conjunction with a relatively fast inter-pattern distance-based neural network learning algorithm called DistAI. However, the general approach can be used with any inductive learning algorithm. The interested reader is referred to (Langley, 1995; Mitchell, 1997; Honavar, 1998a) for surveys of different approaches to inductive learning.

8.3.1 Genetic Algorithms

Evolutionary algorithms include a class related to randomized, population-based heuristic search techniques which include genetic algorithms, genetic programming, evolutionary programming, and variety of related approaches (Mitchell, 1996). They are inspired by processes that are modeled after biological evolution. Central to such evolutionary systems is the idea of a population of potential solutions (individuals) that corresponds to members of a high-dimensional search space.

The individuals represent candidate solutions to the optimization problem being solved. A wide range of genetic representations (e.g., bit vectors, LISP programs, matrices, etc.) can be used to encode the individuals depending on the space of solutions that needs to be searched. In genetic algorithms the individuals are typically represented by n -bit binary vectors. The resulting search space corresponds to an n -dimensional boolean space. In the feature subset selection problem, each individual would represent a feature subset.

It is assumed that the quality of each candidate solution (or fitness of the individual in the population) can be evaluated using a fitness function. In the feature subset selection problem, the fitness function would evaluate the

selected features with respect to some criteria of interest (e.g., cost of the resulting classifier, classification accuracy of the classifier, etc.).

Evolutionary algorithms use some form of fitness-dependent probabilistic selection of individuals from the current population to produce individuals for the next generation. A variety of selection techniques have been explored in the literature. Some of the most common ones are *fitness-proportionate*, *rank-based*, and *tournament-based* selection (Mitchell, 1996). The selected individuals are subjected to the action of genetic operators to obtain new individuals that constitute the next generation. The genetic operators are usually designed to exploit the known properties of the genetic representation, the search space, and the optimization problem to be solved. Genetic operators enable the algorithm to *explore* the space of candidate solutions.

Mutation and *crossover* are two of the most common operators used with genetic algorithms that represent individuals as binary strings. Mutation operates on a single string and generally changes a bit at random. Thus, a string 11010 may, as a consequence of random mutation, get changed to 11110. Crossover, on the other hand, operates on two parent strings to produce two offspring. With a randomly chosen crossover position 4, the two strings 01101 and 11000 yield the offspring 01100 and 11001 as a result of crossover. Other genetic representations require the use of appropriately designed genetic operators.

The process of fitness-dependent selection and application of genetic operators to generate successive generations of individuals is repeated many times until a satisfactory solution is found (or the search fails). It can be shown that evolutionary algorithms of the sort outlined above simulate highly opportunistic and exploitative randomized search that explores high-dimensional search spaces rather effectively under certain conditions. In practice, the performance of evolutionary algorithms depends on a number of factors including: the choice of genetic representation and operators, the fitness function, the details of the fitness-dependent selection procedure, and the various user-determined parameters such as population size, probability of application of different genetic operators, etc. The specific choices made in the experiments reported in this chapter are summarized in Section 8.4.

8.3.2 Neural Networks

Neural networks are densely connected, massively parallel, shallowly serial networks of relatively simple computing elements or neurons (Ripley, 1996; Mitchell, 1997; Honavar, 1998a). Each neuron computes a relatively simple function of its inputs and transmits outputs to other neurons to which it is connected via its output links. A variety of neuron functions are used in practice. Each neuron has associated with it a set of parameters which are modifiable through learning. The most commonly used parameters are the so-called *weights*.

The computational capabilities (and hence pattern classification abilities) of a neural network depend on its architecture (connectivity), functions computed

by the individual neurons, and the setting of parameters or weights used. It is well-known that multi-layer networks of non-linear computing elements (e.g., threshold neurons) can realize any classification function.

Since the function computed by a neural network is determined by its topology as well as the computations performed by individual neurons, designing a neural network for a particular pattern classification task reduces to determination of the network architecture (number of neurons, their connectivity, etc.), the types of neurons (e.g., linear, sigmoid, threshold, etc.), as well as the parameter or weight values. This is typically accomplished through a combination of design (using a-priori knowledge or guesswork) and inductive learning (which may be used to modify, among other things, the weights, network architecture, or both) (Parekh et al., 1997; Honavar, 1998b).

8.3.3 Genetic Algorithm Wrapper Approach to Feature Subset Selection for Neural Network Pattern Classifiers: Some Practical Considerations

Genetic algorithms offer an attractive technique for feature subset selection for neural network pattern classifiers for several reasons, some of which were mentioned above. However, we are faced with several difficulties in using this approach in practice.

Traditional neural network learning algorithms perform an error gradient guided search for a suitable setting of weights in the weight space determined by a user-specified network architecture. This ad hoc choice of network architecture often inappropriately constrains the search for an appropriate setting of weights. For example, if the network has fewer neurons than necessary, the learning algorithm will fail to find the desired classification function. If the network has far more neurons than necessary, it can result in overfitting of the training data leading to poor generalization. In either case, it would make it difficult to evaluate the usefulness of a feature subset employed to describe (or represent) the training patterns used to train the neural network.

Gradient based learning algorithms although mathematically well-founded for unimodal search spaces, can get caught in local minima of the error function. This can complicate the evaluation of a feature subset employed to represent the training patterns used to train the neural networks. This is due to the fact that the poor performance of the classifier might be due to the failure of the learning algorithm, and not the feature subset used.

Fortunately, constructive neural network learning algorithms (Honavar, 1998b) eliminate the need for ad hoc, and often inappropriate a-priori choices of network architectures; and can potentially discover near-minimal networks whose size is commensurate with the complexity of the classification task that is implicitly specified by the training data. Several new, provably convergent, and relatively efficient constructive learning algorithms for multi-category real as well as discrete valued pattern classification tasks have begun to appear in the literature (Parekh et al., 1997; Yang et al., 1998b). Many of these algorithms have demonstrated very good performance in terms of reduced network size, learning time, and generalization in a number of experiments with both

artificial and fairly large real-world datasets (Parekh et al., 1997; Yang et al., 1998b). However, most of them, with the exception of DistAI (Yang et al., 1998b) use time-consuming iterative training algorithms for setting the weights of the neurons.

Using genetic algorithms for feature subset selection for the design of neural network pattern classifiers involves running a genetic algorithm for several generations. In each generation, evaluation of an individual (a feature subset) requires training the corresponding neural network and computing its accuracy and cost. This evaluation has to be performed for each of the individuals in the population. Thus, it is not feasible to use computationally expensive iterative weight update algorithms for training neural network classifiers for evaluating candidate feature subsets. Against this background, DistAI offers an attractive approach to training neural networks.

8.3.4 DistAI: A Fast Algorithm for Constructing Neural Network Pattern Classifiers

DistAI (Yang et al., 1998b) is a simple and relatively fast constructive neural network learning algorithm for pattern classification. The results presented in this chapter are based experiments using neural networks constructed by DistAI. The key idea behind DistAI is to add *hyperspherical* hidden neurons one at a time based on a greedy strategy which ensures that the hidden neuron correctly classifies a maximal subset of training patterns belonging to a single class. Correctly classified examples can then be eliminated from further consideration. The process terminates when the pattern set becomes empty (that is, when the network correctly classifies the entire training set). When this happens, the training set becomes linearly separable in the transformed space defined by the hidden neurons. In fact, it is possible to set the weights on the hidden to output neuron connections without going through an iterative process. It is straightforward to show that DistAI is guaranteed to converge to 100% classification accuracy on any finite training set in time that is polynomial in the number of training patterns (Yang et al., 1998b). Experiments reported in (Yang et al., 1998b) show that DistAI, despite its simplicity, yields classifiers that compare quite favorably with those generated using more sophisticated (and substantially more computationally demanding) learning algorithms. This makes DistAI an attractive choice for experimenting with evolutionary approaches to feature subset selection for neural network pattern classifiers. Key steps in our approach are shown in Figure 8.2.

8.4 IMPLEMENTATION DETAILS

Our experiments were run using a genetic algorithm using rank-based selection strategy. The probability of selection of the highest ranked individual is p (where $0.5 < p < 1.0$ is a user-specified parameter), that of the second highest ranked individual is $p(1 - p)$, that of the third highest ranked individual is $p(1 - p)^2$, ..., that of the last ranked individual is $1 - (\text{sum of the probabilities of}$

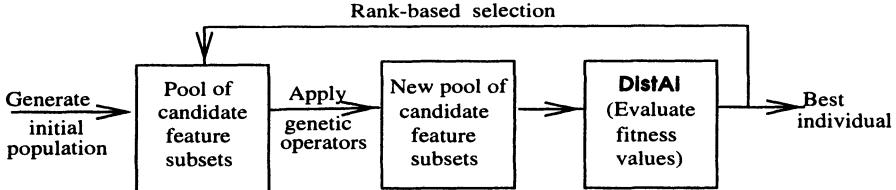


Figure 8.2 GADistAI: Feature subset selection using a genetic algorithm with DistAI.

selection of all the other individuals). The rank-based selection strategy gives a non-zero probability of selection of each individual (Mitchell, 1996). Our experiments used the following parameter settings: Population size is 50; Number of generation is 20; Probability of crossover is 0.6; Probability of mutation is 0.001; Probability of selection of the highest ranked individual is 0.6. The parameter settings were based on results of several preliminary runs. They are comparable to the typical values mentioned in the literature (Mitchell, 1996).

Each individual in the population represents a candidate solution to the feature subset selection problem. Let m be the total number of features available to choose from to represent the patterns to be classified. It is represented by a binary vector of dimension m (where m is the total number of features). If a bit is a 1, it means that the corresponding feature is selected. A value of 0 indicates that the corresponding feature is not selected. The fitness of an individual is determined by evaluating the neural network constructed by DistAI using a training set whose patterns are represented using only the selected subset of features. If an individual has n bits turned on, the corresponding neural network has n input nodes.

The fitness function has to combine two different criteria – the accuracy of the classification function realized by the neural network and the cost of performing classification. The accuracy of the classification function can be estimated by calculating the percentage of patterns in a test set that are correctly classified by the neural network in question. A number of different measures of the cost of classification suggest themselves: cost of measuring the value of a particular feature needed for classification (or the cost of performing the necessary test in a medical diagnosis application), the risk involved, etc. To keep things simple, we chose a 2-criteria fitness function defined as follows:

$$\text{fitness}(x) = \text{accuracy}(x) - \frac{\text{cost}(x)}{\text{accuracy}(x) + 1} + \text{cost}_{\max} \quad (8.1)$$

where $\text{fitness}(x)$ is the fitness of the feature subset represented by x , $\text{accuracy}(x)$ is the test accuracy of the neural network classifier trained using DistAI using the feature subset represented by x , $\text{cost}(x)$ is the sum of measurement costs of feature subset represented by x , and cost_{\max} is an upper bound on the costs of candidate solutions. In this case, it is simply the sum of the costs associated with all of the features. This is clearly a somewhat ad hoc choice. However, it does discourage trivial solutions (e.g., a zero

cost solution with a very low accuracy) from being selected over reasonable solutions which yield high accuracy at a moderate cost. It also ensures that $\forall x \ 0 \leq fitness(x) \leq (100 + cost_{max})$. In practice, defining suitable tradeoffs between the multiple objectives has to be based on knowledge of the domain. In general, it is a non-trivial task to combine multiple optimization criteria into a single fitness function. A wide variety of approaches have been examined in the utility theory literature (Keeney and Raiffa, 1976).

8.5 EXPERIMENTS

8.5.1 Description of Datasets

The experiments reported here used a wide range of real-world datasets from the machine learning data repository at the University of California at Irvine (Murphy and Aha, 1994) as well as a carefully constructed artificial dataset (3-bit parity) to explore the feasibility of using genetic algorithms for feature subset selection for neural network classifiers. The feature subset selection using DistAI is also applied to document classification problem for journal paper abstracts and news articles.

3-bit Parity Dataset (3P). This dataset was constructed to explore the effectiveness of the genetic algorithm in selecting an appropriate subset of relevant features in the presence of redundant features so as to minimize the cost and maximize the accuracy of the resulting neural network pattern classifier. The modified training set is constructed as follows: The original features are replicated once (to introduce redundancy) thereby doubling the number of features. Then an additional set of irrelevant features are generated and are assigned random boolean values. 100 7-bit random vectors were generated and augmented with the 6-bit vectors (corresponding to the original 3 bits plus an identical set of 3 bits). Each feature in the resulting dataset is assigned a random cost between 0 and 9. The performance considering the random costs in addition to the accuracy (see Equation 8.1) was compared with that obtained by considering the accuracy alone.

Datasets from UCI Repository. In our experiments with real world datasets, our objective was to compare the neural networks built using feature subsets selected by the genetic algorithm with those that use the entire set of features. Some medical datasets include measurement costs for the features, but most of the datasets lack this information. Therefore, our experiments with the datasets from UCI repository focused on identifying a minimal subset of features that yield high accuracy neural network classifiers. Where measurement costs were available, the performance considering the cost in addition to the accuracy was compared with that obtained by considering the accuracy alone.

Document Datasets. The paper abstracts were chosen from three different sources: IEEE Expert magazine, Journal of Artificial Intelligence Research and

Neural Computation. The news articles were obtained from Reuters dataset. Each document is represented in the form of a vector of numeric weights for each of the words (terms) in the vocabulary. The weights correspond to the *term frequency-inverse document frequency (TFIDF)* (Salton, 1989) values for the corresponding words. The training sets for paper abstracts were generated based on the classification of the corresponding documents into two classes (interesting and not interesting) by two different individuals, resulting in two different data sets (**Abstract1** and **Abstract2**). The classifications for news articles were given based on their topics (6, 4 and 8 classes) following (Koller and Sahami, 1997), resulting in three different datasets (**Reuters1**, **Reuters2** and **Reuters3**). Since these datasets do not have measurement costs for the features, our experiments with document datasets also focused on identifying a minimal subset of features that yield high accuracy neural network classifiers.

8.5.2 Experimental Results

Two different sets of experiments were run to explore the performance of GADistAI. The first set of experiments were designed to explore the effect of feature subset selection on the performance of DistAI on a given choice of training and test sets. Each dataset was randomly partitioned into a training and test set (with 90% of the data used for training and the remaining 10% for testing). The genetic algorithm was used to select the best feature subset on the basis of this choice of training and test sets. The results were averaged over 5 independent runs of the genetic algorithm, for a given choice of training and test set. This process was repeated 10 times with 10 different choices of training and test set. The results of these experiments (which represent $5 \times 10 = 50$ runs of the genetic algorithm) are shown in Table 8.1 (GADistAI(rand)) and 8.3.

The second set of experiments explored a somewhat different, but related question. Since feature subset selection in GADistAI is guided by the fitness function, it seems reasonable to expect that the quality of fitness estimates will have some impact on the performance of DistAI. Thus, it is interesting to explore the performance of GADistAI when the fitness estimates are obtained using several training and test sets. Thus, in this set of experiments, fitness estimates used by GADistAI were obtained by averaging the observed fitness values for 10 different partitions of the data into training and test sets (i.e., 10-fold cross validation). The results are shown in Table 8.1, 8.2 and 8.4. The results in Table 8.1 represent averages over 5 independent runs (GADistAI(avg)) and the best run among the 5 runs (GADistAI(best)) of the algorithm.

Improvement in Generalization using Feature Subset Selection. To study the effect of feature subset selection on generalization, experiments were run using classification accuracy as the fitness function. The results in Table 8.1 indicate that the networks constructed using GA-selected subset of features (GADistAI (rand)) compare quite favorably with networks that use all of the features in all randomly partitioned datasets. In particular, feature subset selection resulted in substantial improvement in generalization on many of the

Table 8.1 Comparison of neural network pattern classifiers constructed by DistAI using the entire set of features with the best network constructed by GADistAI.^a

Dataset	DistAI ^c		GADistAI (rand) ^d		GADistAI (avg) ^e		GADistAI (best) ^f	
	Dim	Acc	Dim	Acc	Dim	Acc	Dim	Acc
3P	13	79.0	6.6	100	4.8	100	4	100
Annealing	38	96.6	21.0	99.5	20.0	98.8	18	99.5
Audiology	69	66.0	36.4	83.5	37.2	72.6	39	76.5
Bridges	11	63.0	5.6	81.6	4.9	56.9	5	67.0
Cancer	9	97.8	5.4	99.3	6.0	98.0	8	98.6
CRX	15	87.7	8.0	91.5	7.4	87.7	6	88.0
Flag	28	65.8	14.0	78.1	14.2	63.9	18	70.0
Glass	9	70.5	5.5	80.8	4.4	69.3	5	71.0
Heart	13	86.7	7.2	93.9	7.6	85.5	7	85.9
HeartCle^b	13	85.3	7.3	92.9	8.4	86.9	9	87.7
HeartHun^b	13	85.9	7.0	93.0	7.4	85.4	8	87.2
HeartLB^b	13	80.0	7.1	91.0	7.6	79.8	6	83.0
HeartSwi^b	13	94.2	6.6	98.3	7.4	95.3	8	96.7
Hepatitis	19	84.7	9.2	97.1	10.2	85.2	10	88.7
Horse	22	86.0	11.1	92.6	9.6	83.2	5	85.0
Ionosphere	34	94.3	17.3	98.6	16.6	94.5	13	96.0
Pima	8	76.3	3.8	79.5	4.0	73.1	2	76.8
Promoters	57	88.0	28.8	100	30.6	89.8	31	92.0
Sonar	60	83.0	30.7	97.2	32.2	84.0	28	85.5
Soybean	35	81.0	19.4	92.8	21.0	83.1	19	84.3
Vehicle	18	65.4	9.1	68.8	9.4	50.1	11	59.4
Votes	16	96.1	8.9	98.8	8.2	97.0	7	97.9
Vowel	10	69.8	6.5	78.4	6.8	70.2	6	71.5
Wine	13	97.1	6.7	99.4	8.2	96.7	7	97.1
Zoo	16	96.0	9.3	100	8.8	96.8	9	99.0
Abstract1	790	89.0	393.7	97.6	402.2	89.2	387	91.0
Abstract2	790	84.0	393.8	94.4	389.8	84.0	382	85.0
Reuters1	1568	91.6	786.1	94.9	766.0	90.2	750	91.5
Reuters2	435	88.5	218.3	97.5	222.4	90.3	195	91.5
Reuters3	1440	96.4	715.4	98.7	721.0	96.2	712	96.9

^a Dim is the number of features and Acc is the generalization accuracy.^b {Cleveland,Hungarian,Long Beach,Switzerland} heart disease.^c The standard deviation of Acc is less than 12.2.^d The standard deviation of Dim is less than 3.7 for most of the datasets and less than 20.3 for document datasets. The standard deviation of Acc is less than 5.0 for almost all datasets.^e The standard deviation of Dim is less than 3.0 for most of the datasets and less than 16.6 for document datasets. The standard deviation of Acc is less than 3.0 for almost all datasets.^f The standard deviation of Acc is less than 13.8.datasets. (For example, 100% accuracy was obtained with **3P**, **Promoters**,

and **Zoo** datasets). Also, the number of features selected is significantly smaller than the total number of features present in the original data representation.

Table 8.2 Comparison between various approaches for feature subset selection.⁹

<i>Dataset</i>	non-GA		ADHOC		GADistAI	
	<i>Dim</i>	<i>Acc</i>	<i>Dim</i>	<i>Acc</i>	<i>Dim</i>	<i>Acc</i>
Annealing	-	-	8	95.0	18	99.5
Cancer	4	74.7	-	-	8	98.6
CRX	6	85.0	7	85.1	6	88.0
Glass	4	62.5	4	70.5	5	71.0
Heart	3	79.2	5	80.8	7	85.9
Hepatitis	4	84.6	-	-	10	88.7
Horse	4	85.3	-	-	5	85.0
Pima	-	-	3	73.2	2	76.8
Sonar	-	-	16	76.0	28	85.5
Vehicle	-	-	7	69.6	11	59.4
Votes	4	97.0	5	95.7	7	97.9
Reuters1	40	94.1	-	-	750	91.5
Reuters2	40	90.0	-	-	195	91.5
Reuters3	80	98.6	-	-	712	96.9

⁹ A '-' indicates that the result is not reported in the corresponding reference.

The results also indicate that the networks constructed using GA-selected subset of features by average fitness values (GADistAI (avg)) compare favorably with networks that use all of the features in most of the datasets. Clearly, GADistAI outperformed DistAI (with all features) in the parity problem: it successfully selected relevant features that resulted in 100% accuracy. For the remaining datasets, the improvement in generalization ranged from modest in some cases to marginal in others. The best individual generated by GADistAI (GADistAI (best)) outperformed DistAI in almost all datasets. Again, the number of features selected is significantly smaller than the total number of features present in the original data representation in all of the datasets.

Table 8.2 compares the results of GADistAI with the results of other GA-based (ADHOC) (Richeldi and Lanzi, 1996) and the best among several non GA-based approaches (non-GA) that are available in the literature (Liu and Setiono, 1996a; Liu and Setiono, 1996b; Kohavi, 1994; Koller and Sahami, 1996; Koller and Sahami, 1997). The results indicate that GADistAI gave higher generalization accuracy than the other techniques or comparable accuracy in almost all cases (except **Vehicle** dataset) although it occasionally selected more features. GADistAI produced feature subsets with larger number of features than the approach in (Koller and Sahami, 1996; Koller and Sahami, 1997) for **Reuters** datasets. This can be explained by that the former found the feature subsets using a genetic algorithm for datasets with relatively large number of features while the latter set up the number of features to select a-priori.

It should be noted that it is not generally feasible to do a completely fair and thorough comparison between different approaches without the complete knowledge of the parameters and the set up used in the experiments.

Minimizing Cost and Maximizing Accuracy using Feature Subset Selection. The selection was based on both the generalization accuracy and the measurement cost of features. **3P**, **HeartCle**, **Hepatitis** and **Pima** datasets were used for the experiment (with the random costs in the 3-bit parity problem). The results shown in Table 8.3 and 8.4 are obtained by the fitness estimates over randomly partitioned datasets and the average fitness estimates over datasets arranged by 10-fold cross validation, respectively.

In Table 8.3, the fitness function that combined both accuracy and cost outperformed that based on accuracy alone in every respect: the number of features used, generalization accuracy, and cost. This is not surprising because the former tries to minimize cost (while maximizing the accuracy), which reduces the number of features, while the latter emphasizes only on the accuracy.

Table 8.4 also shows the fitness function that combined both accuracy and cost outperforms that based on accuracy alone in all datasets except **HeartCle**. The generalization accuracy was higher and the cost was also higher with the fitness function that is based on accuracy alone in **HeartCle** dataset. This explains how the fitness function (Equation 8.1) works in GADistAI and verifies the rationale behind it. Also, note that some of the runs resulted in feature subsets which did not necessarily have minimum cost. This suggests the possibility of improving the results by the use of a more principled choice of a fitness function that combines accuracy and cost.

Table 8.3 Comparison of neural network pattern classifiers constructed by GADistAI with different fitness evaluations for randomly partitioned datasets.

<i>Dataset</i>	Accuracy only			Accuracy & Cost		
	<i>Dim</i>	<i>Acc</i>	<i>Cost</i>	<i>Dim</i>	<i>Acc</i>	<i>Cost</i>
3P	6.6	100	46.1	4.3	100	26.7
HeartCle	7.3	92.9	335.7	6.1	93.0	261.5
Hepatitis	9.2	97.1	22.8	8.3	97.3	19.0
Pima	3.8	79.5	28.5	3.1	79.5	22.8

8.6 SUMMARY AND DISCUSSION

An approach to feature subset selection using a genetic algorithm for neural network pattern classifiers is proposed in this chapter. A fast inter-pattern distance-based constructive neural network algorithm, DistAI, is employed to evaluate the fitness (in terms of the generalization accuracy) of candidate feature subsets in the genetic algorithm. The results presented in this chapter indicate that genetic algorithms offer an attractive approach to solving the

Table 8.4 Comparison of neural network pattern classifiers constructed by GADistAI with different fitness evaluations for datasets arranged by 10-fold cross validation.

<i>Dataset</i>	Accuracy only			Accuracy & Cost		
	<i>Dim</i>	<i>Acc</i>	<i>Cost</i>	<i>Dim</i>	<i>Acc</i>	<i>Cost</i>
3P	4.8	100	35.6	3.8	100	25.4
HeartCle	8.4	86.9	390.5	7.2	85.7	317.8
Hepatitis	10.2	85.2	23.4	10.0	85.3	23.2
Pima	4.0	73.1	29.3	4.2	76.1	20.8

feature subset selection problem in inductive learning of pattern classifiers in general, and neural network pattern classifiers in particular.

The GA-based approach to feature subset selection does not rely on monotonicity assumptions that are used in traditional approaches to feature selection which often limits their applicability to real-world classification and knowledge acquisition tasks. It also offers a natural approach to feature subset selection by taking into account, the distribution of available data. This is due to the fact that feature selection is driven by estimated fitness values, which if based on multiple partitions of the dataset into training and test data, provide a robust measure of performance of the feature subset. This is not generally the case with many of the greedy stepwise algorithms that select features based on a single partition of the data into training and test sets. Consequently, the feature subsets selected by such algorithms are likely to perform rather poorly on other random partitions of the data into training and test sets.

The approach to feature subset selection is able to naturally incorporate multiple criteria (e.g., accuracy, cost) into the feature selection process. This finds applications in cost-sensitive design of classifiers for tasks such as medical diagnosis, computer vision, among others. Another interesting application is automated data mining and knowledge discovery from datasets with an abundance of irrelevant or redundant features. In such cases, identifying a relevant subset that adequately captures the regularities in the data can be particularly useful, particularly in scientific knowledge discovery tasks. Techniques similar to the one discussed in this chapter have been successfully used recently to select feature subsets for pattern classification tasks that arise in power system security assessment (Zhou et al., 1997), sensor subsets in the design of behavior and control structures for autonomous mobile robots (Balakrishnan and Honavar, 1996).

Additional experiments with GADistAI in scientific knowledge discovery tasks in bioinformatics (e.g., discovery of protein structure–function relationships, carcinogenicity prediction, gene sequence identification) are currently in progress. Some directions for future research include: Extension of feature subset selection by incorporating *feature construction* and *genetic programming* (Koza, 1992); Extensive experimental (and wherever feasible, theoretical) comparison of the performance of the proposed approach with that of conventional

methods for feature subset selection; More principled design of multi-objective fitness functions for feature subset selection using domain knowledge as well as mathematically well-founded tools of multi-attribute utility theory (Keeney and Raiffa, 1976).

Acknowledgments

This research was partially supported by National Science Foundation Grant IRI-9409580 and John Deere Foundation Grant to Vasant Honavar. The authors wish to thank Mehran Sahami for providing **Reuters** document datasets. The authors are grateful to Dr. Pazzani of the Department of Information and Computer Science at the University of California at Irvine for managing the repository of machine learning datasets and making it available to us. An earlier version of this chapter appears in IEEE Expert. ©1998 IEEE.

References

- Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305.
- Balakrishnan, K. and Honavar, V. (1996). On sensor evolution in robotics. In Koza, Goldberg, Fogel, and Riolo, editors, *Proceedings of the 1996 Genetic Programming Conference – GP-96*, pages 455–460. MIT Press, Cambridge, MA.
- Brill, F., Brown, D., and Martin, W. (1992). Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks*, 3(2):324–328.
- Caruana, R. and Freitag, D. (1994). Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 28–36, New Brunswick, NJ. Morgan Kaufmann.
- Cost, S. and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1):57–78.
- Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27.
- Dasarathy, B. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA.
- Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, 1(3).
- Doak, J. (1992). An evaluation of feature selection methods and their application to computer security. Technical Report CSE-92-18, Department of Computer Science, University of California, Davis, CA.
- Foroutan, I. and Sklansky, J. (1987). Feature selection for automatic classification of non-gaussian data. *IEEE Transactions on Systems, Man and Cybernetics*, 17:187–198.
- Guo, Z. (1992). *Nuclear Power Plant Fault Diagnostics and Thermal Performance Studies Using Neural Networks and Genetic Algorithms*. PhD thesis, University of Tennessee, Knoxville, TN.

- Guo, Z. and Uhrig, R. (1992). Using genetic algorithms to select inputs for neural networks. In *Proceedings of COGANN'92*, pages 223–234.
- Honavar, V. (1998a). Machine learning: Principles and applications. In Webster, J., editor, *Encyclopedia of Electrical and Electronics Engineering*. Wiley, New York. To appear.
- Honavar, V. (1998b). Structural learning. In Webster, J., editor, *Encyclopedia of Electrical and Electronics Engineering*. Wiley, New York. To appear.
- John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121–129, New Brunswick, NJ. Morgan Kaufmann.
- Keeney, R. and Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, New York.
- Kira, K. and Rendell, L. (1992). A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 249–256. Morgan Kaufmann.
- Kohavi, R. (1994). Feature subset selection as search with probabilistic estimates. In *AAAI Fall Symposium on Relevance*.
- Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *Machine Learning: Proceedings of the Thirteenth International Conference*. Morgan Kaufmann.
- Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *International Conference on Machine Learning*, pages 170–178.
- Kononenko, I. (1994). Estimating attributes: Analysis and extension of relief. In *Proceedings of European Conference on Machine Learning*, pages 171–182.
- Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.
- Langley, P. (1994). Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*, pages 1–5, New Orleans, LA. AAAI Press.
- Langley, P. (1995). *Elements of Machine Learning*. Morgan Kaufmann, Palo Alto, CA.
- Liu, H. and Setiono, R. (1996a). Feature selection and classification - a probabilistic wrapper approach. In *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES*.
- Liu, H. and Setiono, R. (1996b). A probabilistic approach to feature selection - a filter solution. In *Proceedings of the Thirteenth International Conference on Machine Learning*. Morgan Kaufmann.
- Mitchell, M. (1996). *An Introduction to Genetic algorithms*. MIT Press, Cambridge, MA.
- Mitchell, T. (1997). *Machine Learning*. McGraw Hill, New York.
- Modrzejewski, M. (1993). Feature selection using rough sets theory. In *Proceedings of the European Conference on Machine Learning*, pages 213–226. Springer.

- Murphy, P. and Aha, D. (1994). Repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA.
- Narendra, P. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26:917–922.
- Parekh, R., Yang, J., and Honavar, V. (1997). Constructive neural network learning algorithms for multi-category real-valued pattern classification. Technical Report ISU-CS-TR97-06, Department of Computer Science, Iowa State University.
- Richeldi, M. and Lanzi, P. (1996). Performing effective feature selection by investigating the deep structure of the data. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 379–383. AAAI Press.
- Ripley, B. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, New York.
- Salton, G. (1989). *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Massachusetts.
- Sheinvald, J., Dom, B., and Niblack, W. (1990). A modelling approach to feature selection. In *Proceedings of the Tenth International Conference on Pattern Recognition*, pages 535–539.
- Siedlecki, W. and Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition*, 2:197–220.
- Siedlecki, W. and Sklansky, J. (1989). A note on genetic algorithms for large-scale feature selection. *IEEE Transactions on Computers*, 10:335–347.
- Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill-climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 293–301, New Brunswick, NJ. Morgan Kaufmann.
- Vafaie, H. and De Jong, K. (1993). Robust feature selection algorithms. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, pages 356–363.
- Wettschereck, D., Aha, D., and Mohri, T. (1995). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Technical Report AIC95-012, Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, Washington, D.C.
- Yang, J., Pai, P., Honavar, V., and Miller, L. (1998a). Mobile intelligent agents for document classification and retrieval: A machine learning approach. In *14th European Meeting on Cybernetics and Systems Research. Symposium on Agent Theory to Agent Implementation*, Vienna, Austria.
- Yang, J., Parekh, R., and Honavar, V. (1998b). DistAI: An inter-pattern distance-based constructive learning algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, Alaska. To appear.

Zhou, G., McCalley, J., and Honavar, V. (1997). Power system security margin prediction using radial basis function networks. In *Proceedings of the 29th Annual North American Power Symposium*, Laramie, Wyoming.

9 A RELEVANCY FILTER FOR CONSTRUCTIVE INDUCTION

Nada Lavrač¹, Dragan Gamberger², and Peter Turney³

¹J. Stefan Institute
Jamova 39, 1001 Ljubljana, Slovenia
nada.lavrac@ijs.si

²Rudjer Bošković Institute
Bijenička 54, 10000 Zagreb, Croatia
gambi@lehp1.irb.hr

³Institute for Information Technology
National Research Council Canada
M-50 Montreal Road, Ottawa, Ontario, Canada, K1A 0R6
peter@ai.iit.nrc.ca

Abstract: Some machine learning algorithms enable the learner to extend its vocabulary with new terms if, for a given a set of training examples, the learner's vocabulary is too restricted for solving the learning task. In this work we propose a filter that selects the potentially relevant terms from the set of constructed terms, and eliminates the terms which are irrelevant for the learning task. By biasing constructive induction (or predicate invention) to relevant terms only, the explored space of constructed terms can be much larger. The elimination of irrelevant terms is specially well suited for learners of large time or space complexity, such as genetic algorithms and neural nets. This chapter presents the REDUCE algorithm for eliminating irrelevant terms and a case study in which REDUCE is used to preprocess data for a hybrid genetic algorithm RL-ICET. In addition, a noise elimination algorithm is proposed to handle real-life datasets involving noisy data.

9.1 INTRODUCTION

Inductive concept learning can be viewed as a process of searching a space of concept descriptions (hypotheses). The language bias determines the space of

hypotheses to be searched. The language bias is determined by the syntactic restrictions of the hypothesis language and the vocabulary of terms in the language, as well as the vocabulary of functions and relations defined in the background knowledge.

Consider a propositional learning task where, given a fixed set of attributes, training examples are represented by tuples of features (attribute values). If the given vocabulary is too restricted for the learning task, the task of *constructive induction* is to extend the hypothesis language with new terms, automatically constructed from the terms in the learner's vocabulary or generated based on the functions and relations defined in the background knowledge (Michalski, 1983). The problem is then how to select, from the set of constructed terms, only those terms that are *relevant* for the given task.

First-order learning of relational descriptions (inductive logic programming) assumes a given set of training examples represented by a relational table, and background knowledge, represented either extensionally (in the form of relational tables) or intentionally (in the form of rules) (Lavrač and Džeroski, 1994). If the given language bias is too restricted for the learning task, the task of *predicate invention* is to invent definitions of new predicates from the definitions of predicates in the background knowledge, thus causing bias shift (Stahl, 1996). Alternatively, bias shift may occur by allowing the learner to switch its search to a hypothesis space defined by a syntactically more expressive hypothesis language.

Constructive induction and predicate invention may be invaluable for the success of learning. However, uncontrolled expansion of the hypothesis language can decrease the learner's performance, particularly in terms of efficiency. Moreover, despite the extended vocabulary and the increased time and space complexity of learning, there is no guarantee that the new vocabulary will indeed help the learner to induce a better solution, since the new terms may be irrelevant for the task at hand.

Let us assume that we have a mechanism for expanding the hypothesis language by constructing new terms, and propose a filter that distinguishes the potentially *relevant* terms from the terms which are *irrelevant* for the learning task. The relevancy filter can be used to eliminate the irrelevant terms in preprocessing of the set of training examples. By biasing constructive induction (or predicate invention) to relevant terms only, the explored space of constructed terms can be larger, which increases the chance of successfully solving the learning task.

The proposed filter is directly applicable in two-class propositional learning problems described with binary-valued terms, and those inductive logic programming tasks for which a LINUS-like transformation approach is applicable (Lavrač and Džeroski, 1994). When applying the filter to a multi-class learning problem, the problem needs to be transformed to a two-class learning problem; the transformation can be done the same way as in the well known covering algorithms AQ and CN2. Multi-valued attributes need to be transformed into

binary-valued literal form; the transformation to this form is described in this chapter.

In the case study that illustrates our approach to feature construction and irrelevant feature elimination, we apply the proposed relevancy filter to the 20 and 24 trains East-West challenge problems posed by D. Michie et al. in the Computer Journal (Michie et. al., 1994). The case study indicates that the elimination of irrelevant features is well suited for learners with a large time or space complexity such as genetic algorithms (and artificial neural networks). The experiments show that the performance of a hybrid genetic algorithm RL-ICET was significantly improved by applying the relevancy filter in preprocessing of the dataset (Turney 1996).

The chapter concludes by a brief description of an approach to noise handling by noise elimination.

9.2 RELEVANCE OF LITERALS AND FEATURES

Consider a two-class learning problem where the training set E consists of positive and negative examples of a concept, $E = P \cup N$, and examples $e \in E$ are tuples of truth-values of terms in a hypothesis language. The set of all terms are called *literals*, L .

Let us represent the training set E as a table where rows correspond to training examples and columns correspond to literals. An element in the table has the value *true* when the example satisfies the condition (literal) in the column of the table, otherwise its value is *false*.

9.2.1 How to Achieve the Required Propositional Representation

If the training set does not have the form of tuples of truth-values, a transformation to this form is performed in preprocessing of the training set. For attribute-value learning, the transformation procedure is based on analysis of the values of examples in the training set. For each attribute A_i , let v_{ix} ($x = 1..k_{ip}$) be the k_{ip} different values of the attribute that appear in the positive examples and let w_{iy} ($y = 1..k_{in}$) be the k_{in} different values appearing in the negative examples. The transformation results in the following set of literals L :

- For discrete attributes A_i , literals of the form $A_i = v_{ix}$ and $A_i \neq w_{iy}$ are generated.
- For continuous attributes A_i , literals of the form $A_i \leq (v_{ix} + w_{iy})/2$ are created for all neighboring value pairs (v_{ix}, w_{iy}) , and literals $A_i > (v_{ix} + w_{iy})/2$ for all neighboring pairs (w_{iy}, v_{ix}) .
- For integer valued attributes A_i , literals are generated as if A_i were both discrete and continuous, resulting in literals of four different forms: $A_i \leq (v_{ix} + w_{iy})/2$, $A_i > (v_{ix} + w_{iy})/2$, $A_i = v_{ix}$, and $A_i \neq w_{iy}$.

9.2.2 Construction of New Terms

We assume that new terms also have the form of *literals*, that have values *true* or *false* for the given training examples. Suppose that the original training set is described by attributes A_i . Let us illustrate some of the new terms that can be created by constructive induction: literals that test the relations $A_i = A_j$, $A_i \neq A_j$ for attributes of the same type (same sets of values, or continuous attributes); literals introducing internal disjunctions $A_i = [v_{ik} \vee v_{il}]$, or intervals $A_i \in [v_{ik}, v_{il}]$; conjunctions of features ($A_i = v_{ik} \wedge A_j = v_{jl}$); literals that test values of functions defined in the background knowledge, e.g., $f(A_i, A_j, \dots) \leq v$; literals using the relations defined in the background knowledge $r(A_i, A_j)$, etc. An obvious way to construct new terms is also by generating negations of literals (the so-called *negative literals*): for every $r(A_i, A_j)$ construct a literal $\neg r(A_i, A_j)$.

A general procedure for the construction of terms based on the information in the background knowledge is proposed in LINUS (Lavrač and Džeroski, 1994). It can be used for the construction of new terms in attribute-value learning, as well as in first-order learning (inductive logic programming), where the relation learning tasks can be solved by using the restricted hypothesis language of LINUS, i.e., the language of constrained nonrecursive clauses with typed variables.

Let us illustrate the construction of new terms in LINUS (Lavrač and Džeroski, 1994). Consider a relation learning task where the training set consists of examples of the target relation *daughter*(A_1, A_2), and the background knowledge consists of definitions of a unary relation *female* and a binary relation *parent*. The transformation of training examples results in a table whose rows are truth-value tuples corresponding to individual training examples, and columns correspond to literals, constructed by the appropriate variabilisation of background knowledge predicates: *female*(A_1), *female*(A_2), *parent*(A_1, A_2), *parent*(A_2, A_1), *parent*(A_1, A_1), *parent*(A_2, A_2), and $A_1 = A_2$. Notice that the variabilisation of constructed literals is restricted to the use of variables A_1 and A_2 only, since in constrained clauses only the variables appearing in the head of an induced clause may appear in the literals of the body of the clause. Thus, for the target relation *daughter*(A_1, A_2) the constructed literals may only use A_1 and A_2 in the arguments, whereas literals introducing new variables, such as *parent*(A_1, Z), are disregarded. Despite this limitation of the hypothesis language, the complexity analysis of the LINUS transformation approach reveals that the number of constructed literals is exponential in the number of arguments (of different types) of the background knowledge predicates, therefore the elimination of irrelevant literals constructed by the LINUS transformation procedure is invaluable for the success of learning, when the number and arity of predicates in the background knowledge is large.

As opposed to the general procedure proposed in LINUS, the case study in this chapter presents a special-purpose procedure for constructing new terms in the East-West challenge problem, where the construction of new terms turns out to be the key reason for the successful solution of the learning task.

9.2.3 The p/n Pairs of Examples

Let the set of training examples E be represented by a truth-value table where columns correspond to the set of (positive and negative) literals L , and rows are tuples of truth-values of literals, representing individual examples in E . The table is divided in two parts, P and N , where P are the positive examples, and N are the negative examples. Let $P \cup N$ denote the truth-value table E .

Let us introduce the following definitions and notation (Lavrač et. al., 1996):

- A p/n pair is a pair of training examples where $p \in P$ and $n \in N$.
- Literal $l \in L$ covers a p/n pair if in column l of the table of training examples E the positive example p has value *true* and the negative example n has value *false*.
- $E(l)$ denotes the set of all p/n pairs covered by literal l .
- Literal l covers literal l' if $E(l') \subseteq E(l)$.

9.2.4 The Relevance of Literals

Consider a simple learning problem with five training examples forming example set E : three positive $P = \{p_1, p_2, p_3\}$, and two negative $N = \{n_1, n_2\}$. Examples are described by truth-values of positive and negative literals $l_i \in L$. Table 9.1 shows some of the truth-values of examples E .

Table 9.1 Coverage of literals, coverage of p/n pairs.

<i>Examples</i>	<i>Literals</i>							
	l_2	l_4	l_8	
P								
	p_1							
	p_2		<i>true</i>		<i>true</i>		<i>false</i>	
N	n_1		<i>false</i>		<i>false</i>		<i>true</i>	
	n_2		<i>false</i>		<i>true</i>		<i>true</i>	

To illustrate the meaning of p/n pairs and the notions of coverage and relevance of literals, let us consider just three literals, l_2 , l_4 and l_8 . Literal l_2 in Table 9.1 appears to be relevant for the formation of an inductive hypothesis since it is true for a positive example and false for both negative examples; a hypothesis constructed from l_2 only would cover the positive example p_2 and would not cover the negative examples n_1 and n_2 ; literal l_2 thus appears to be a reasonable ingredient of an inductive hypothesis aimed at covering all the positive examples and none of the negatives examples.

Literal l_2 covers two p/n pairs: $E(l_2) = \{p_2/n_1, p_2/n_2\}$. Literal l_8 is inappropriate for constructing a hypothesis, since it does not cover any p/n pair: $E(l_8) = \emptyset$. Literal l_4 seems to be less relevant than l_2 and more relevant than

l_8 since it covers one p/n pair: $E(l_4) = \{p_2/n_1\}$. Literal l_2 covers l_4 and l_8 , and literal l_4 covers l_8 , since $E(l_8) \subseteq E(l_4) \subseteq E(l_2)$.

The example in Table 9.1 enables us to reach the following intuition: the more p/n pairs a literal covers, the more relevant it is for hypothesis formation. We can formalize this intuition in the next definition:

- Literal l' is *irrelevant* if there exists a literal $l \in L$ such that l covers l' ($E(l') \subseteq E(l)$). In other words, literal l' is irrelevant if it covers a subset of p/n pairs covered by some other literal $l \in L$.

Let us now assume that literals are assigned costs. Let $c(l)$ denote the cost of literal $l \in L$. The definition of irrelevance needs to be modified if we want to take into account the costs of literals.

- Literal l' is *irrelevant* if there exists a literal $l \in L$ such that l covers l' ($E(l') \subseteq E(l)$) and the cost of l is lower than the cost of l' ($c(l) \leq c(l')$).

In the case study presented in this chapter, cost is a measure of complexity: the more syntactically complex the literal, the higher its cost.

9.2.5 A Filter for Cost-Sensitive Elimination of Irrelevant Literals

Figure 9.1 presents a cost-sensitive algorithm for irrelevant literal elimination.

Given: costs $c(l)$ of literals in $L \cup L_{New}$

Input: L – initial set of relevant literals, L_{New} – new literals,
 $E = P \cup N$ – table of positive and negative examples,
consisting of tuples of truth-values of L

```

for  $\forall l_i \in L_{New}$  do
     $\forall p \in P$  and  $\forall n \in N$  evaluate  $l_i$  as true or false
    if  $\forall p \in P$   $l_i$  has value false then  $l_i$  is irrelevant
    if  $\forall n \in N$   $l_i$  has value true then  $l_i$  is irrelevant
    if  $l_i$  is covered by any  $l_j \in L$  for which  $c(l_j) \leq c(l_i)$ 
        then  $l_i$  is irrelevant
    else  $L \leftarrow L \cup \{l_i\}$ , and
        add column of truth-values of  $l_i$  to table  $E = P \cup N$ 
endfor

```

Output: L – extended set of relevant literals,
 $E = P \cup N$ – extended table of positive and negative examples,
consisting of tuples of truth-values of L

Figure 9.1 Relevancy filter: Cost-sensitive elimination of irrelevant literals.

If irrelevant literals are generated by a constructive induction procedure, they can be detected and eliminated before entering the learning process.

The algorithm assumes a set of training examples E described by an initial set of relevant literals L , a set of constructed literals L_{New} , and the assignment of literal costs $c(l)$. If no costs are assigned, we assume for all $l \in L \cup L_{New}$ the cost $c(l) = 1$.

9.2.6 Implementation Issues

The relevancy filter can be efficiently implemented using simple bitstring manipulation on the table of training examples E . For this purpose the table E is transformed into E_t as follows:

$\forall p \in P$: replace *true* by 1 and *false* by 0

$\forall n \in N$: replace *false* by 1 and *true* by 0

In this representation, examples $e \in E_t$ and literals $l \in L$ are bitstrings. Coverage can now be checked by set inclusion. Recall that literal l covers literal l' if $E(l') \subseteq E(l)$. Thus, if l' has value 1 only in (some of) those rows in which l has 1 and in no other rows, l' is irrelevant and can be eliminated.

The relevancy filter assumes that the initial literal set consists of relevant literals. Alternatively, the entire set $L \cup L_{New}$ can be considered as a unique literal set to be checked for the relevance of its elements; that variant of the relevancy filter was used in our experiments. That algorithm, called REDUCE, was initially developed by Gamberger and implemented in the ILLM (Inductive Learning by Logic Minimization) algorithm for inductive concept learning (Gamberger, 1995). The REDUCE algorithm (Lavrač et. al., 1996) first eliminates all the useless literals l_i (literals that are *false* for all $p \in P$ and literals that are *true* for all $n \in N$) and continues by the elimination of other irrelevant literals in turn.

9.2.7 Relevance of Features

The term *feature* is used to denote a positive literal, for example, $A_i = v$, $A_j \leq w$, $r(A_i, A_j)$. In the hypothesis language, the existence of one such feature implies the existence of two complementary literals: a positive and a negative literal. Suppose that we consider the feature *Color = black* and that the attribute *Color* has three possible values: *black*, *white*, *red*. Since each feature implies the existence of two literals, the necessary and sufficient condition that a feature can be eliminated as irrelevant is that both of its literals *Color = black* and *Color ≠ black* (i.e., $\neg(\text{Color} = \text{black})$) are irrelevant. This statement directly implies the procedure taken in our experiment. First we convert the starting feature vector to the corresponding literal vector which has twice as many elements. After that, we eliminate the irrelevant literals and, in the third step, we construct the reduced set of features which includes all the features which have at least one of their literals in the reduced literal vector.

It must be noted that direct detection of irrelevant features (without conversion to and from the literal form) is not possible except in the trivial case where two (or more) features have identical columns in the E_t table. Only in

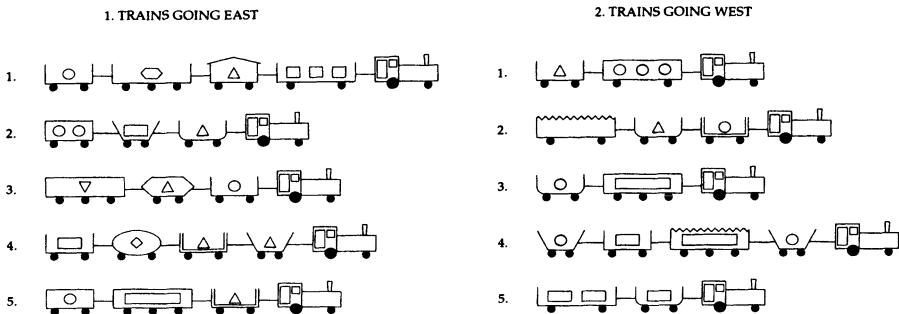


Figure 9.2 The original 10 train East-West challenge, by Michalski and Larson.

this case a feature f exists whose literals f and $\neg f$ cover both literals g and $\neg g$ of some other feature. In a general case if a literal of feature f covers some literal of feature g then the other literal of feature g is not covered by the other literal of feature f . But it can happen that this other literal of feature g is covered by a literal of some other feature h . This means that although there is no such feature f that covers both literals of feature g , feature g can still turn out to be irrelevant.

This analysis supports the approach, used in this study, in which the input to the learner are tuples of truth-values of positive and negative literals, rather than feature vectors which are used in most standard approaches to rule induction.

9.3 UTILITY STUDY: THE EAST-WEST CHALLENGE

Michie et al. issued a “challenge to the international computing community” to discover low size-complexity Prolog programs for classifying trains as Eastbound or Westbound (Michie et. al., 1994). The challenge was inspired by a problem posed by Michalski and Larson, published in 1977, where the task was to generate rules that will classify trains as East or Westbound. The original problem is illustrated in Figure 9.2.

The original challenge issued by Michie et al. included three separate tasks (Michie et. al., 1994). Michie later issued a second challenge, involving a fourth task. Our experiments described here involve the first and fourth tasks. The first task had 20 trains, 10 Eastbound and 10 Westbound, whereas the fourth task involved 24 trains, 12 Eastbound and 12 Westbound. The challenge in these tasks was to discover the simplest rule for distinguishing the Eastbound and Westbound trains.

For both tasks, the winner was decided by representing the rule as a Prolog program and measuring its size-complexity. The size-complexity of the Prolog program was calculated as the sum of the number of clause occurrences, the

number of term occurrences, and the number of atom occurrences. Note that performance on these two tasks was judged by size-complexity, not by accuracy on independent testing data (there were no independent testing data). All rules competing in the challenge were required to achieve 100% accuracy on the data.

9.3.1 RL-ICET

ICET is a cost-sensitive algorithm, a hybrid of a genetic algorithm and a decision tree induction algorithm, designed to generate low-cost decision trees (Turney 1996). The genetic algorithm is Grefenstette's GENESIS (Grefenstette, 1986) and the decision tree induction algorithm is Quinlan's C4.5 (Quinlan, 1992). ICET uses a two-tiered search strategy. On the bottom tier, C4.5 uses a TDIDT (Top Down Induction of Decision Trees) strategy to search through the space of decision trees. On the top tier, GENESIS uses a genetic algorithm to search through the space of biases.

ICET takes feature vectors as input and generates decision trees as output, using its C4.5 component. The C4.5 component of ICET has been modified from Quinlan's original design, so that its learning bias can be controlled by a vector of real-valued parameters, called a bias vector. The GENESIS component of ICET searches in the space of bias vectors for a bias that optimizes the performance of the C4.5 component, according to a given performance measure. ICET uses a performance measure that is sensitive to both the cost of features (the cost of acquiring information about an element in a feature vector) and the cost of classification errors (the cost of mistaken classifications made by the output decision tree).

Although ICET takes feature vectors as input and generates decision trees as output, the East-West Challenge involves input data in the form of Prolog relations, and output theories in the form of Prolog programs. For the East-West Challenge, ICET was extended to handle Prolog input. This algorithm is called RL-ICET (Relational Learning with ICET) (Turney 1996).

RL-ICET is similar to the LINUS learning system, since they both use a three-part learning strategy (Lavrač and Džeroski, 1994). First, a preprocessor translates the Prolog relations and predicates into a feature vector format. The preprocessor in RL-ICET was designed specially for the East-West Challenge, whereas LINUS has a general-purpose preprocessor. Second, an attribute-value learner applies a decision tree induction algorithm (ICET) to the feature vectors. Each feature is assigned a cost, based on the size of the fragment of Prolog code that represents the corresponding predicate or relation. A decision tree that has a low cost corresponds (roughly) to a Prolog program that has a low size-complexity. When it searches for a low cost decision tree, ICET is in effect searching for a low size-complexity Prolog program. Third, a post-processor translates the decision tree into a Prolog program. Postprocessing with RL-ICET is done manually, whereas LINUS performs post-processing automatically.

9.3.2 Feature Construction in RL-ICET

Much of the success of RL-ICET in the East-West challenge tasks may be attributed to its preprocessor which translates the Prolog descriptions of the trains into a feature vector representation. The data about each train in the East-West challenge were represented using Prolog. For example, a selected train and the corresponding representation in Prolog clause form are shown in Figure 9.3.



```
eastbound([c(1, rectangle, short, not_double, none, 2, 1(circle, 1)),
          c(2, rectangle, long, not_double, none, 3, 1(hexagon, 1)),
          c(3, rectangle, short, not_double, peaked, 2, 1(triangle, 1)),
          c(4, rectangle, long, not_double, none, 2, 1(rectangle, 3))]).
```

Figure 9.3 The first train and its Prolog clause representation.

The relatively compact Prolog description was converted by a simple Prolog program into a feature vector format (tuples of truth-values of features) to be used for decision tree induction. This resulted in rather large feature vectors of 1199 elements. The large vectors were required to ensure that all the features that are potentially interesting for the final solution are made available for ICET.

What follows is a brief outline of the feature construction procedure that occurs in the preprocessing of the training set: We started with 28 predicates that apply to the cars in a train, such as *ellipse(C)*, which is true when the car C has an elliptical shape. For each of these 28 predicates, we defined a corresponding feature. All of the features were defined for whole trains, rather than single cars, since the problem is to classify trains. The feature *ellipse*, for example, has the value *true* when a given train has a car with an elliptical shape. Otherwise *ellipse* has the value *false*. We then defined features by forming all possible unordered pairs of the original 28 predicates. For example, the feature *ellipse_triangle_load* has the value *true* when a given train has a car with an elliptical shape that is carrying a triangle load, and *false* otherwise. Note that the features *ellipse* and *triangle_load* may have the value *true* for a given train while the feature *ellipse_triangle_load* has the value *false*, since *ellipse_triangle_load* only has the value *true* when the train has a car that is both elliptical and carrying a triangle load. Next we defined features by forming all possible ordered pairs of the original 28 predicates, using the relation *infront(T, C1, C2)*. For example, the feature *u_shaped_infront_peaked_roof* has the value *true* when the train has a U-shaped car in front of a car with a peaked roof, and *false* otherwise. Finally, we added 9 more predicates that apply to the train as a whole, such as *train_4*, which has the value *true* when

the train has exactly four cars. Thus a train is represented by a feature vector, where every feature has either the value *true* or the value *false*.

Each feature was assigned a cost, based on the complexity of the fragment of Prolog code required to represent the given feature. Recall that the complexity of a Prolog program is defined as a sum of the number of clause occurrences, the number of term occurrences and the number of atom occurrences. Table 9.2 shows some examples of constructed features and their costs.

Table 9.2 Examples of features and their costs.

Feature	Prolog Fragment	Cost (Complexity)
<i>ellipse</i>	<i>has_car(T, C), ellipse(C).</i>	5
<i>short_closed</i>	<i>has_car(T, C), short(C), closed(C).</i>	7
<i>train_4</i>	<i>len1(T, 4).</i>	3
<i>train_hexagon</i>	<i>has_load1(T, hexagon).</i>	3
<i>ellipse_peaked_roof</i>	<i>has_car(T, C), ellipse(C), arg(5, C, peaked).</i>	9
<i>u_shaped_no_load</i>	<i>has_car(T, C), u_shaped(C), has_load(C, 0).</i>	8
<i>rectangle_load_infront_jagged_roof</i>	<i>infront(T, C1, C2), has_load0(C1, rectangle), arg(5, C2, jagged).</i>	11

The feature vector for a train does not capture all the information that is in the original Prolog representation. For example, we could also define features by combining all possible unordered triples of the 28 predicates. However, these features would likely be less useful, since they are so specific that they will only rarely have the value *true*. If the target concept should happen to be a triple of predicates, it could be closely approximated by the conjunction of the three pairs of predicates that are subsets of the triple.

This kind of translation to feature vector representation could be applied to many other types of structured objects. For example, consider the problem of classifying a set of documents. The keywords in a document are analogous to the cars in a train. The distance between keywords or the order of keywords in a document may be useful when classifying the document, just as the *infront* relation may be useful when classifying trains.

9.3.3 Experimental Setting and Results of Experiments

The objective of the experiments was to show the utility of the literal elimination algorithm REDUCE. The idea of the proposed approach is to allow the generation of a large number of different features, which will surely include all significant ones, and then, prior to the use of an inductive learner, eliminate all irrelevant features in order to keep the computation as effective as possible.

Two experiments were performed separately for the 20 and 24 trains problems. In both experiments, the RL-ICET preprocessor was used to generate the appropriate features and to transform the training examples into a feature vector format. This resulted in two training sets of 20 and 24 examples each, described by 1199 features.

In order to apply the REDUCE algorithm we first converted the starting feature vector of 1199 elements to the corresponding literal vector which has twice as many elements, containing 1199 features generated by the RL-ICET preprocessor (positive literals) as well as their negated counterparts (1199 negative literals). After that, we eliminated the irrelevant literals and, in the third phase, constructed the reduced set of features which includes all the features which have at least one of their literals in the reduced literal set.

The experimental setup, designed to test the utility of REDUCE, was as follows. First, 10 runs of the ICET algorithm were performed on the set of training examples described with 1199 features. Second, 10 runs of ICET were performed on the training examples described with the reduced set of features selected by REDUCE. Ten runs are needed because of the stochastic nature of the ICET algorithm, which produces a different result each time it runs, assuming its random number seed is changed. The average results of 10 runs of RL-ICET are compared with respect to the costs of decision trees and execution time.

In the initial experiments with RL-ICET, the performance was measured by the cost of decision trees induced by ICET, as well as the complexity of Prolog programs after the RL-ICET transformation of decision trees into the Prolog program form (Turney 1996). Here we skip the latter, since the transformation into the Prolog form is currently manual and sub-optimal, which means that a tree with the lowest cost found by ICET is not necessarily transformed into a Prolog program with the lowest complexity.

Results of the 20 Trains Experiment. The results of the 20 trains experiment are summarized in Table 9.3. With the 20 train data, REDUCE cuts the original set of 1199 features down to 86 features. In this way, the complexity of the learning problem is reduced to about 7% (86/1199) of the initial learning problem.

The efficiency of learning significantly increases: in the initial problem with 1199 features, the average time per run of RL-ICET is more than 2 hours (exactly 2:16:54), whereas in the reduced problem setting with 86 features the average time per run of RL-ICET is about 12 minutes (11:57). The difference between the two times is significant. This shows the utility of feature reduction for genetic algorithms which are typically demanding in terms of computation time.

The average cost of trees induced from the 86 feature set decreases, from $mean(c_{1199}) = 20$ to $mean(c_{86}) = 18.6$, but the difference is not statistically significant. The variance of the costs is also reduced, i.e., the costs of the

Table 9.3 Summary of results in the 20 trains East-West challenge.

Trial	86 features		Trial	1199 features	
	Time t_1	Cost c_1		Time t_2	Cost c_2
1	11:05	18	11	2:21:32	24
2	11:19	21	12	2:21:34	21
3	12:55	18	13	2:19:15	20
4	11:35	18	14	2:19:32	20
5	15:16	18	15	2:16:20	18
6	11:35	18	16	2:23:52	22
7	11:32	18	17	2:24:09	21
8	11:38	18	18	2:18:41	16
9	11:28	18	19	2:16:58	18
10	11:18	21	20	2:23:09	20
Sum	119:41	186	Sum	23:25:02	200
Mean	11:57	18.6	Mean	2:16:54	20

decision trees generated from 1199 features vary more than the costs of the trees generated from 86 features: $\text{var}(c_{1199}) = 5.1$ and $\text{var}(c_{86}) = 1.6$.

The decision tree reported in (Turney 1996) was rediscovered here in Trial 6 (with 86 features) and in Trial 15 (with 1199 features). This decision tree and the corresponding Prolog program are shown in Figure 9.4. This Prolog program is the best (lowest complexity) program that has been found by RL-ICET. It is interesting to note that RL-ICET could discover this tree even when the number of features was reduced from 1199 to 86.

```

short_closed = true: eastbound (8.0)
short_closed = false:
| train_4 = false: westbound (7.0)
| train_4 = true:
| | u_shaped = false: westbound (2.0)
| | u_shaped = true:
| | | train_circle = false: westbound (1.0)
| | | train_circle = true : eastbound (2.0)

eastbound(T) :-
    has_car(T, C),
    ((short(C), closed(C));
     (len1(T, 4), u_shaped(C), has_load1(T, circle))).
```

Figure 9.4 The best decision tree generated by RL-ICET (cost = 18 units) and the corresponding Prolog program (complexity = 19 units), for the 20 trains problem.

Results of the 24 Trains Experiment. In this experiment, REDUCE decreases the number of features from 1199 to 116. The complexity of the learning problem is reduced to about 10% (116/1199) of the initial problem.

Again, the efficiency of learning significantly increases. In the initial problem with 1199 features, the average time per run of RL-ICET is nearly two hours (exactly 1:55:05) whereas in the reduced problem setting with 116 features the average time per run of RL-ICET is about 14 minutes (14:12). The difference between the two times is significant. The difference between average decision tree costs also decreases, from $mean(c_{1199}) = 25.4$ to $mean(c_{116}) = 19.9$. The difference between $mean(c_{1199})$ and $mean(c_{116})$ is significant at the 99.99% confidence level, whereas the reduction in variance is not significant: $var(c_{1199}) = 5.2$ and $var(c_{116}) = 4.8$.

9.4 HANDLING NOISY DATA

In a noise-free setting we assume that the learning goal is to find a consistent and complete concept description. In this chapter, the choice of a simplified noise-free setting enabled a clear presentation of the notions underlying the relevancy filter. Real-life problems, however, involve a more complex setting, including noise handling. This section proposes an approach to noise handling by noisy example elimination in data preprocessing (Gamberger and Lavrač, 1996; Gamberger et. al., 1996).

Assume a propositional inductive task that starts from a set of training examples E that are part of some example domain D , $E \subset D$. Given E , and a set of literals L , assume that the training set E contains enough training examples to induce/identify a correct hypothesis H (correctness of H means that H is complete and consistent for all, including unseen, examples of the unknown target theory T . In other words, H denotes the same concept as T).

Let us define a noisy example as an example $e \in D$ which is inconsistent with the target theory T . Hence, for a noisy example e , the target theory T and the target hypothesis H are not correct.

Suppose that the set of examples E does not contain any noisy example. Suppose also that the set E is *saturated* which means that it is large enough to identify the correct hypothesis H . Let $E = \{e_1, \dots, e_m\}$, where $m = |E|$ is larger than a saturation threshold m_0 , $m \geq m_0$.

Let $q(E, L)$ represent the complexity of the simplest hypothesis that is complete and consistent with training examples in E and built of literals from L . Let us temporarily fix the set of literals L and study the complexity q only as a function of E : $q(E, L) = q(E)$.

If the set E is large enough, then by adding correct examples to E , the complexity of the simplest hypothesis that can be generated from the enlarged training set will not increase. In other words, if $m = |E| \geq m_0$, then for training examples e_{m+1}, e_{m+2}, \dots that are not included in the initial training set E it holds that:

$$q(E) = q(E \cup \{e_{m+1}\}) = q(E \cup \{e_{m+1}, e_{m+2}\}) = \dots$$

Hence, if $e \notin E$ is not noisy, it is consistent with the target theory and $q(E \cup \{e\}) = q(E)$. On the other hand, if e is a noisy example, it is inconsistent with T and $q(E \cup \{e\}) \geq q(E)$. The proof of the above statement can be found in (Gamberger and Lavrač, 1996).

This statement is the basis for the suggested noise elimination algorithm. The following approach is taken to answer the question whether the training set E contain noise. If it holds that

$$q(E) = q(E \setminus \{e\})$$

for all possible examples $e \in E$ then there is no noise in the dataset and the dataset is large enough for a correct hypothesis H to be induced. If this is not the case and an example e enables complexity reduction, i.e., $q(E) > q(E \setminus \{e\})$, then the example e is potentially incorrect. If there is more than one such example whose elimination leads to complexity reduction, it is advisable to eliminate the example that reduces the complexity the most. The elimination of ‘potentially noisy’ examples is an iterative process where one example is eliminated in each iteration until no elimination can result in further complexity reduction.

Based on the notion of p/n pairs, the noise elimination algorithm (Gamberger and Lavrač, 1996; Gamberger et. al., 1996) first heuristically evaluates the minimal number of literals needed to construct a complete and consistent hypothesis, and then detects in the training set the potentially noisy examples as those whose elimination will decrease the number of literals needed for hypothesis generation (Gamberger et. al., 1996).

9.5 NOISE AND RELEVANCE

The noise handling algorithm and the relevancy filter can be used iteratively. First, the relevancy filter is used to eliminate irrelevant features. Next, the noise handling algorithm eliminates potentially noisy examples from the training set. Example elimination may cause further literal elimination, since some literals may have been used only for the sake of covering some noisy examples. Having performed one or two steps of literal and noisy example elimination, a learner can be used for hypothesis construction. This can be either an algorithm capable of dealing with noise in the dataset, such as RL-ICET that employs C4.5, or alternatively, an algorithm without noise-handling capabilities.

9.6 RELATED WORK

The approach to feature construction presented in this study is based on the utilization of the available background knowledge about the domain. The approach is close to the approach taken in LINUS (Lavrač and Džeroski, 1994), but differs from most other approaches to constructive induction and predicate invention (see (Pagallo and Haussler, 1990; Wnek and Michalski 1991; Oliveira and Sangiovanni, 1992; Koller and Sahami, 1996) for constructive induction, and (Stahl, 1996) for an overview on predicate invention).

The problem of relevance was addressed in early research on inductive concept learning (Michalski, 1983). Basically one can say that all learners are concerned with the selection of ‘good’ literals to be used to construct the hypothesis. As shown in this volume, the problem of relevance has recently attracted much attention in the context of feature selection in attribute-value learning (Caruana and Freitag, 1994; John et. al., 1994; Skalak, 1994).

The aim of this study is to propose a filter that can be used to distinguish between a set of literals that are relevant for learning and a set of irrelevant literals that can be discarded before learning. Such filtering of irrelevant literals can thus be viewed as a part of preprocessing of the set of training examples. In this sense, the proposed approach to feature selection is close to (Oliveira and Sangiovanni, 1992).

9.7 CONCLUSION

The aim of this work is to contribute to a better understanding of the notion of relevance for inductive concept learning, in the context of constructive induction. The presented case study shows that the construction of appropriate features can be crucial for the success of learning, and that cost-sensitive elimination of irrelevant features can substantially improve the efficiency of learning and can reduce the costs of induced hypotheses.

Representation of training examples as tuples of truth-values of literals is typical both for genetic algorithms and for neural networks, and both learning approaches require substantial CPU time. The case study shows that the dimensionality of the problem can significantly decrease, resulting in a substantial improvement in terms of efficiency. Since RL-ICET originally required about two hours to process the 20 or 24 trains problems, it seemed that the RL-ICET approach to constructive induction did not scale up to larger problems, with hundreds or thousands of instances (Turney 1996). However, with REDUCE as a preprocessor, RL-ICET now requires under fifteen minutes for the 20 or 24 trains problems. REDUCE itself runs in seconds. It seems that the combination of REDUCE and RL-ICET can scale up to much larger problems than were previously practical for RL-ICET alone.

The case study confirms the potential of our approach for learning with genetic algorithms, and also its potential for learning with neural networks, where the elimination of irrelevant literals may decrease the computation time as well as facilitate the design of the network. In further work the usefulness of literal elimination could be studied also in the context of data preprocessing for learning by neural networks.

Another point made in this study is the consideration of literals versus features. It is useful to have negated literals as ingredients of the hypothesis language. This study points out the necessity of considering both positive and negative literals for elimination; this consideration may be useful also for the genetic algorithm and neural networks communities.

The case study deals with an exact, noise-free problem in which we assume that the learning goal is to find a consistent and complete concept description.

The choice of a simplified noise-free setting enabled a clear presentation of the notions underlying the implemented relevancy filter.

Real-life problems require a more complex setting, involving the use of our algorithm for noise detection and elimination. The proposed noise elimination algorithm and the relevancy filter can be used iteratively, in preprocessing of the dataset.

Acknowledgments

This research was supported by the Slovenian Ministry of Science and Technology, the Croatian Ministry of Science, the National Research Council of Canada, and the ESPRIT project 20237 Inductive Logic Programming 2. The authors are grateful to Donald Michie for stimulating this work. The authors are grateful to IEEE for the permission to reprint parts of the copyrighted article, appearing in *IEEE Expert*, 1998, and acknowledge the permission of Springer to reprint parts of the ISSEK workshop paper, published in CISM Vol. 382 Learning, Networks and Statistics.

References

- Caruana, R. and Freitag, D. (1994). Greedy attribute selection. In *Proceedings of the 11th International Conference on Machine Learning*, Morgan Kaufmann, pages 28–36.
- Gamberger, D. (1995). A minimization approach to propositional inductive learning. In *Proceedings of the 8th European Conference on Machine Learning (ECML-95)*, Springer, pages 151–160.
- Gamberger, D. and Lavrač, N. (1996). Noise detection and elimination applied to noise handling in a KRK chess endgame. In *Proc. of the 5th International Workshop on Inductive Logic Programming*, pages 59–75.
- Gamberger, D., Lavrač, N. and Džeroski, S. (1996). Noise elimination in inductive concept learning: A case study in medical diagnosis. In *Proc. 7th International Workshop on Algorithmic Learning Theory*, Springer, Berlin, pages 199–212.
- Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16, pages 122–128, 1986.
- John, G.H., Kohavi, R. and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning*, Morgan Kaufmann, pages 190–198.
- Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *Proc. of International Conference on Machine Learning*, Bari, pages 284–292.
- Lavrač, N. and Džeroski, D. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.
- Lavrač, N., Gamberger, D. and Turney, P. (1996). Cost-sensitive feature reduction applied to a hybrid genetic algorithm. In *Proceedings of the 7th International Workshop on Algorithmic Learning Theory (ALT-96)*, Springer, pages 127–134.

- Michalski, R.S. (1983). A theory and methodology of inductive learning. In: Michalski, R., Carbonell, J. and Mitchell, T. (eds.) *Machine Learning: An Artificial Intelligence Approach*, Tioga, pages 83–134.
- Michie, D., Muggleton, S., Page, D., and Srinivasan, A. (1994). To the international computing community: A new East-West challenge. Oxford University Computing Laboratory, Oxford. URL <ftp://ftp.ics.uci.edu/pub/machine-learning-databases/trains/> contains information on both the 20 and 24 train challenges.
- Oliveira, A.L., and Sangiovanni-Vincentelli, A. (1992). Constructive induction using a non-greedy strategy for feature selection. In *Proceedings of the International Conference on Machine Learning*, Aberdeen, pages 354–360.
- Pagallo, G. and Haussler, D. (1990) Boolean feature discovery in empirical learning. *Machine Learning*, 5, pages 71–99.
- Quinlan, J.R. (1992). C4.5: Programs for Machine Learning. Morgan Kaufmann.
- Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the 11th International Conference on Machine Learning*, Morgan Kaufmann, pages 293–301.
- Stahl, I. (1996) Predicate invention in inductive logic programming. In L. De Raedt (ed.) *Advances in Inductive Logic Programming*, IOS Press.
- Turney, P. (1996). Low size-complexity inductive logic programming: The East-West Challenge as a problem in cost-sensitive classification. In *Advances in Inductive Logic Programming*, IOS Press, pages 308–321.
- Wnek, J. and Michalski, R.S. (1991). Hypothesis-driven constructive induction in AQ17: A method and experiments. In *Proceedings of IJCAI-91 Workshop on Evaluating and Changing Representations in Machine Learning*, Sydney, pages 13–22.

III Feature Extraction

10 LEXICAL CONTEXTUAL RELATIONS FOR THE UNSUPERVISED DISCOVERY OF TEXTS FEATURES

Patrick Perrin and Fred Petry

Department of Computer Science
Tulane University, New Orleans, LA 70118
U.S.A.

perrin@eecs.tulane.edu

Abstract: This chapter studies the role of lexical contextual relations for the problem of systematically extracting the most relevant features characterizing texts in the narrative discourse form. Narrative texts have inherent structure dictated by language usage in generating them. We suggest that the relative distance between terms within a text gives sufficient information about its topic structure and its relevant content. We describe a model we developed for identifying major features in texts that were collected about some topics. Text features can be used, for example, to discover important events in psychiatric medical reports to direct the physician to certain recurrent and interesting aspects of some cases. The model utilizes the inherent structure of narrative discourse in the form of extracting context-dependent lexical relations. We propose and discuss possible measurements through experiments from applying the model to a database of psychiatric evaluation reports. We qualitatively demonstrate that a useful text structure and content can be systematically extracted by collocational lexical analysis without the need to encode any supplemental sources of knowledge.

10.1 INTRODUCTION

Since written texts are the richest repository of knowledge in almost all fields (e.g., Medical, Engineering, Science, Education, etc.), techniques and tools for the automatic acquisition of knowledge from texts is naturally intended to be

the most productive method of building knowledge bases. Whatever the techniques used for knowledge acquisition from texts (e.g., information retrieval or knowledge discovery), the major initial problem is to be able to extract systematically the most relevant content of the texts, since natural languages are ambiguous and infinite. Parsing only a single normal sentence can often lead to hundreds of possible parse trees rendering its interpretation a complex process requiring extra knowledge (common sense knowledge, pragmatics, domain knowledge, and so on). Although a lot of progress has been currently achieved, it is still difficult to extract the deep semantic structure of the class of sentences found in all documents, let alone the meaning of the document itself. In summary, current theories are not robust enough thus cannot be applied to large sets of unrestricted texts, and developing the necessary extra knowledge is out of reach for most problems (time and/or cost constraints) (Jacobs, 1992). We need then a rather simple structure that can be extracted from the texts in an automatic manner and at a reasonable cost, yet rich enough to allow for relevant discovery operations. Therefore, to be practical in a real-world environment, any knowledge discovery system has to overcome this problem and must attempt to guess at best what information each utterance of the corpus contains without relying uniquely on NLP parsing techniques. To do so, one can rely on statistics and information theory. The idea is simple and consists of statistically analyzing the whole corpus for lexical collocational frequencies, therefore deriving information about the distributional patterns of the language used. The model assumes then no prior extra knowledge and attempts to derive information based only on the properties of the corpus at hand. The basic assumption is to exploit the stochastic nature of natural languages: the choice of words used to express a particular meaning is not random but depends on the words that have been chosen so far in the near loci. The more frequently a given consecutive sequence of possibly non-adjacent words appears in the corpus, then this feature is characteristic of the corpus and might be of some importance. It is likely to convey some information that might be relevant and characteristic to the corpus because the repetition of information boosts its confidence in its truth. This is due to the fact that (1) terms co-occurring within a span of few words do so because they are somewhat loosely semantically related; (2) in the class of texts we consider, only one sense of a word is used in a given discourse; (3) any two contextual units consisting of mostly the same words are deemed to be about the same discourse theme.

10.1.1 Related Work

Several algorithms have been developed to make use of information on lexical occurrence, such as to automatically assign part-of-speech tags to new sentences from their a-priori statistical occurrence in a large corpus (Church et al., 1991), to recognize verb sub-categorization frames using syntactic local context (Manning, 1993), and for machine translation (Brown and *et al.*, 1990). Another approach to KD in textual databases is to study the corpus with respect to its concept distribution (Feldman and Dagan, 1995). Interest-

ing discoveries will be hinted at by unexpected distributions. But, concepts are not automatically acquired, which would require a manual construction of a concept hierarchy. The immediate drawbacks is the reliance on the tagged documents since manual tagging is expensive, therefore very few databases are tagged. It is even unrealistic to consider that very large ever growing textual databases will be tagged ever (Brill and Mooney, 1997; Cardie, 1997). Riloff considered the problem of automatically generating extraction patterns from untagged texts (Riloff, 1996). The approach is to generate an extraction pattern for each noun phrase of the corpora. Although this approach does not require annotated documents, it still needs the corpora to be somehow partitioned into relevant and irrelevant texts with respect to some specified subject of interest. In addition, it requires a traditional NLP component to parse the sentences to recognize the noun phrases, which is really a domain-specific pattern matching technique. Texts need not be annotated, but a domain specific concept node lexicon is necessary to be used by the sentence analyzer. The most relevant part is that she used relevance statistics (conditional probabilities) for each pattern in order to select the set of patterns that are the most characteristic of the set of labeled relevant texts. Her goal was not a discovery task, but a classification task with a specific goal: to have a certain number of patterns which can accurately extract relevant information from new texts. The approach is interesting because she notes that the relevance function worked well to identify the most relevant patterns. Lapalut studied the problem of knowledge acquisition from texts as a top-down hierarchical clustering task (Lapalut, 1995). He reports interesting findings: groups of words (he used pairs of words), identified by their statistical occurrence in sentences, have been identified as relevant semantic contexts. Lapalut goes further by clustering these classes of significant terms to reveal the structure in experts' interviews in a car accident domain. His goal is to learn a classifier that is able to recognize the discourse structure in the interviews and to relate them and gain more knowledge about the domain. This approach proved useful in revealing specific vocabulary and terminologies of a domain of interest. An interesting approach using discourse structure, but not directly intended in knowledge discovery tasks, is the one taken by Alterman and Bookman (Alterman and Bookman, 1990). They apply a knowledge intensive technique to interpret short texts and then plot the number of inferences that can be made against the clausal partition in the text. They are then able to segment the text by finding breaks in high concentrations of inferences, indicating an episode change in the discourse. Such breaking allows one to produce a summary of the text. Most work in extracting lexical information is done in automatically building lexicons based large set of texts reflecting the actual usage of the language, catching typical expressions (Soderland et al., 1995). A lot of work is also going on in text categorization, which is to index correctly texts (or part of texts) from a corpus with respect to their relevance to a given user's query (Lehnert et al., 1995).

10.2 COLLOCATIONAL EXPRESSIONS AS TEXT FEATURES

Texts can be defined as collections of words, implying both connected and unconnected strings. There are several aspects of semantic information one can obtain from texts; among them, (1) conceptual information concerning the possible (mutual) relationships holding between the meanings of lexical items (paradigmatic) and (2) collocational information concerning the meaning of lexical items in sequence (syntagmatic). Collocational analysis is not only a syntactic problem (i.e., a combination of lexical items), but also a semantic one. The objective is to identify significant collocational expressions among all possible ones in a text. To reveal interesting patterns in the data, one must first impose some structure to the corpus (or text). We use lexical collocational information and discourse structure information, in addition to a relative entropy measure (mutual information) to select the minimal set of collocational expressions representing the major topics of the text. Second, one must define interestingness to search for interesting textual patterns among all those that can be extracted. We used the definition of interestingness as related to the amount of surprise. The assumption is that readers usually find interesting the information that is the most surprising, since attention will be best retained and interest grows for whatever is not expected. Therefore, an interesting textual pattern is an informative pattern (the one with high amount of surprise with respect to what is expected). Any text in the declarative class of texts implies some context reflected by an inherent hierarchy of coherent contextual units, where all concepts form intrinsic relations (strong associations). This is called a text structure, which we define as a sequence of major topics (called themes), each being a set of relevant assertions. This structure has then two major dimensions which can be represented graphically. For example, Figure 10.1 depicts a typical psychiatric evaluation report. Each assertion is a true statement and is relevant to its overall theme. This is due to the fact that all statements in the text are assumed to be true and valid. Both themes and relevant assertions are extracted automatically from the text (Perrin and Petry, 1998).

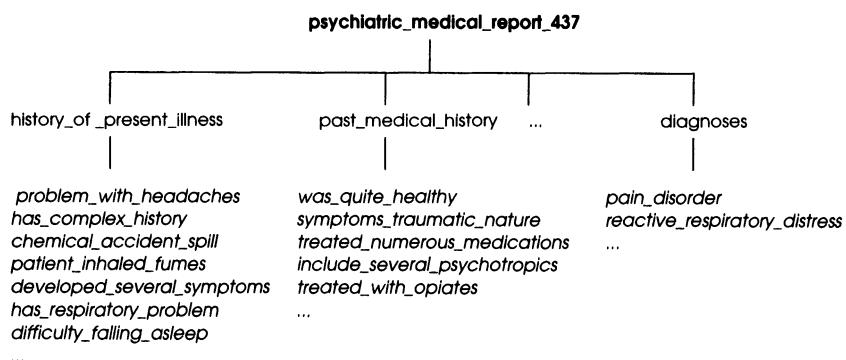


Figure 10.1 An example of a typical text structure

10.2.1 Textual Features

A text is a discourse with some intended meaning, expressed as a specific choice of sequences of words to form sentences, paragraphs and sections to structure and convey the intended meaning. This choice reflects the discourse structure. Work in cognitive psychology and psycholinguistics suggest that (1) natural languages are so constructed that they are asymmetrically hierarchical; (2) meaning is mainly carried by groups of 2 to 4 words; (3) not all words are linguistically significant – content words (nouns, verbs, adjectives and certain adverbs) convey the core of the meaning –; and (4) groups of content words are somewhat close to each other and usually span over more or less 5 words prior and after any word. A textual pattern is then best defined as a sequence of 2 to 3 consecutive, and possibly not adjacent, words within approximately 5 words. We refer textual patterns as collocational expressions to express the fact that they are groups of words related in meaning. For example *patient_depressed* and *fixation_trauma* are collocational expressions in the sentence:

“the patient describes feeling depressed with some fixation involving the trauma of her grandfather’s death”

We modeled two important assumptions derived from major findings from multiple works in psycholinguistics and linguistics. The first assumption is that the natural logical articulation of a declarative text reflects, to some acceptable extent, its intended meaning, or semantic structure. There actually exist various clues to find, in a straightforward manner, such natural (inherent) topic segmentation, mainly (1) by exploiting any explicit discourse structure, that is, by recognizing any pre-existing structure in the form of concise major sections headings (they usually partition the text into its various intended topics); and (2) by recognizing the fact that in declarative texts, words and constructs tend to be used with a consistent single intended meaning, and therefore the groups of sentences (paragraphs) containing mostly the same constructs, or words usage, are mostly likely to be about the same topics. They then form basic logical contextual units, which all together represent a linear segmentation of the text corresponding to its implicit discourse structure. The second assumption is to recognize the fact that particular combinations of content-words are believed to convey the core of the meaning, at least to reflect it to some extent.

10.2.2 Interesting Features are the Most Informative Ones

The measure of information is the key in information engineering. Intuitively, information is related to both meaning and surprise. Quantifying the meaning of single sentences is almost feasible for intelligent systems in AI according to developments in natural language understanding (NLU), but much more difficult, if even feasible, for naturally occurring long texts. Quantification of the amount of surprise has been studied for a long time in information theoretics (Shannon, 1948). Information has a special meaning in mathematics: it is based only on the surprise aspect because there is no mathematical representation for natural language meaning (Applebaum, 1996). Therefore, using information

content measures (entropy) based only on the surprise effect, one can rule out expected (ie, not surprising) assertions. Unfortunately, at the extreme, surprising but meaningless assertions will not be discarded, but we can assume that sort of pattern rarely occurs in the datasets we are using (Perrin, 1997). Nevertheless, it is important to remember that using entropy functions to measure interestingness actually measures only the surprise aspect and not the meaning. Even with such a limitation, we show that it is possible to use entropy functions to distinguish interesting textual patterns and to select the most relevant collocational expressions. The relation, or intra-collocation measure, between α words is expressed by the mutual information measure (Shannon, 1948):

$$I(\omega_1; \omega_2; \dots; \omega_\alpha) = \log_2 \frac{P(\omega_1, \omega_2, \dots, \omega_\alpha)}{P(\omega_1)P(\omega_2)\dots P(\omega_\alpha)}$$

and is also referred as an information gain, since the highest the information content, the more the gain in information. This idea assumes that the level of surprise is inversely related to the probability of guessing the correct label (e.g., relevant or not). It compares the probability of observing all words of the expression together with the probability of observing each separately. The larger I is the more interesting is the relationship between the α consecutive words (correlation). Table 10.1 shows the mutual information and frequency values for some hypothetical collocational expressions in a text, where $\alpha = 2$ and a corpus of 10 million words. This example shows that the words “*denied*” and

Table 10.1 Relevance of collocational expressions (example)

$I(\omega_1; \omega_2)$	$freq(\omega_1; \omega_2)$	$freq(\omega_1)$	$freq(\omega_2)$	ω_1	ω_2
7.64	30	10000	150	<i>denied</i>	<i>fever</i>
6.06	10	10000	150	<i>denied</i>	<i>smoking</i>
4.90	30	10000	1000	<i>denied</i>	<i>pain</i>

“*fever*” are strongly related (compared to the two other features) and therefore the feature “*denied fever*” carries significant information. Although the words *fever* and “*smoking*” have the same frequency of occurrence in the corpus, since the occurrence of “*denied fever*” is more frequent than the occurrence of “*denied smoking*”, it is more significant. Similarly, the occurrences in the corpus of both features “*denied fever*” and “*denied pain*” are equivalent, but since word “*pain*” appears more often than word “*fever*”, and it appears less often (in comparison) together with word “*denied*” than with word “*fever*”, and therefore the feature “*denied pain*” is a less informative feature, which is reflected by a lower mutual information value.

10.3 DISCOVERY OF CONTEXTUAL TEXT FEATURES

We developed and implemented a contextual model to automatically process a text expressed in natural language (Perrin and Petry, 1998). It relies only on contextual information given by the actual content of the text to extract automatically the most relevant constructs reflecting its major topic structure, and consequently its intended meaning. In (Perrin, 1997), we discuss in detail the various issues pertaining to this modeling that are summarized next.

10.3.1 *How to Represent the Discourse Topic Structure as well as the Relevant Assertions*

We resolve the knowledge representation issue in using a subset of a first-order logic (definite program clauses) language to describe the extracted textual assertions. Mainly, it is better suited for NL representation than the propositional form.

10.3.2 *How to Systematically Discover the Overall Discourse Topic Structure*

We developed an algorithm that makes use of explicit discourse structure, such as the major section headings in psychiatric reports. The objective is to extract an overall topic structure to partition the text into non-fixed size contextual units, cohesive in their content, and therefore to assign a contextual label to whatever facts that will be extracted from such contextual units. This is a labeling problem and the proposed approach is simple, can be systematic and does not require an oracle, which are all desired qualities in a knowledge discovery paradigm. This makes the approach robust.

10.3.3 *How to Systematically Discover a Set of Relevant Assertions*

We developed an algorithm that automatically extracts facts from the text and evaluates them for their relevance to the text. We use lexical collocational expressions for the facts and mutual information to reveal the most cohesive groups of content words. This approach is appealing because it provides a means to systematically take minimal measurements from any unrestricted text in the class in an unsupervised manner while ensuring the representation of all themes of the text. This is an unsupervised feature analysis problem, and especially a feature selection one, where the collocational expressions are the measurements and the entropy measure serves at reducing the dimensionality of the space (the number of features, or measurements) to render the discovery problem tractable. Each collocational expression is labeled by the theme of the contextual unit it was extracted from. The purpose is to attach a weight to each theme measuring its degree of cohesion (intra relationship). As humans do discard irrelevant information in cognitive tasks, we model this aspect by ranking the extracted themes by their degree of relevancy in the text they describe. The intra-collocation measure reflects the degree of coherence among the constituents of the collocational expression, that is, how good is the

collocational expression in describing the text. In many researches, the coefficient of relatedness between the terms of a collocational expression is computed by using statistical occurrence, for example: cosine, Jaccard, Dice's similarity functions (Chen and Lynch, 1992; Salton, 1989). Sperber and Wilson made an essential contribution to the research on relevance (Sperber and Wilson, 1986). They consider the psychological relevance of a proposition to a context. This is based on the assumption in which people have an intuition on the relevance issue – they can consistently distinguish relevant from irrelevant information. As Akman and Surav point out, such intuitions are not readily elicitable nor that it is possible to use them as evidence because the ordinary language notion of relevance comes along with an imprecise and ambiguous meaning (Akman and Surav, 1996). It has been found that relevance is relative to context, which renders the problem difficult because there is no way to know precisely which context someone has in mind at any given moment. Sperber and Wilson also mention degrees of relevance related to the fact that a piece of information might be more relevant to a certain context than another one. They introduced a mental processing effort to compare the relevance of various pieces of information. This measure consists of the length of the chain of reasoning as well as the amount of encyclopedic information involved in the process. They go on by noting that an assumption is relevant in a context to the extent that its contextual effects in this context are large; and it is irrelevant to the extent that the effort required to process it in this context is large. This framework has been used to interpret psychological relevance in relation to an information retrieval task (Harter, 1992). The intent of our model is to capture those findings (Perrin and Petry, 1998).

10.4 RESULTS AND DISCUSSION

From cognitive foundations, it has been reported that lexical relations actually represent a text content. This is a valid statement from a human cognitive stand point. In our model, we suggest that a relative entropy measure is appropriate to distinguish the most important collocational expressions (the minimal set of those expressions covering best each theme of the text). Naturally, the following question arises: do selected collocational expressions actually model lexical relations? That is, how good is the set of selected collocational expressions for a knowledge discovery task? How well does it represent the text content with respect to its discourse topic structure? Recall that our objective is to mine texts for regularities assumed to convey inherent interesting knowledge. With this goal in mind, we developed qualitative experiments to test the model in the following directions. It is important to note the non-statistical qualitative nature of these experiments. They are intended as a validation of the model and not for its proof. Appropriateness of the collocational expressions to their corresponding text is a subjective notion difficult to evaluate. We rely on average human judgment.

10.4.1 Data Used

We used a textual database of 557 psychiatric evaluation notes containing a total of 732968 words. Among these, 421254 are content words, with only 15390 different ones, suggesting a large repetition factor to describe psychiatric phenomena, even though the reports were generated by several different physicians. A psychiatric evaluation note is a medical report consisting of several pages of full narrative text transcribed from notes dictated during patient visits. The dataset corresponds to a mix of language usage and to various expertise on the same domain.

10.4.2 Minimum Theme Coverage

Table 10.2 Minimum themes coverage for report 557

<i>history_of_present_illness</i>	<i>past_medical_history</i>	<i>diagnosis</i>
coke_morphine_fentanyl	gonorrhea_chlamydia_herpes	narcissistic_borderline_traits
describe_waking_sweat	harrington_rod_placement	
downers_mentions_quaaludes	harrington_rod_vertebral	
hiv_test_unaware	mitral_valve_prolapse	
morning_awakening_appetite	rod_placement_vertebral	
morphine_fentanyl_patches	surgery_harrington_placement	
providing_emotional_support	venereal_disease_gonorrhea	
overtly_harming_mutilation		
quaaludes_coke_morphine		
quaaludes_coke_fentanyl		
retreating_sort_nonexistence		
receiving_elavil_making		
received_brief_detox		
shared_needles_knowledge		

Collocational expressions aim at describing the content of each contextual unit, or theme, of a text. Therefore, the selected collocational expressions should discriminate the definition of each theme, that is, they should be typical to each theme of the text they describe. In this experiment, we investigate whether collocational expressions, when selected by a minimum theme coverage method (Perrin, 1997), discriminate the themes of a text. It has been designed to show that the use of the minimal mutual information measure has an optimal discriminatory effect in selecting the collocational expressions to represent a text. Look at, for example, in Table 10.2 for the most relevant collocational expressions of a typical psychiatric evaluation report and which themes they describe. At a first glance, and leaving alone their deep meaning, it seems that the mutual information measure does just what we expected. It seems that the collocational expressions that were selected due to a high mutual information measure (i.e., $I(\text{colloc}_i, \text{theme}_j) \gg 0$) are actually good representative of

each theme of the text, with respect to a discriminative point of view. Each collocational expression is unique to a single theme; furthermore, there is no constituents of any of the collocational expressions appearing outside its theme. This suits the desirable discriminative text representation issue.

Table 10.3 describes the minimum theme coverage for the same report. It shows, for decreasing values of the mutual information content, how many collocational expressions cover each of the five themes text 557 contains, from the eight themes of the corpus. This table shows a minimum theme coverage for a mutual information content value of 15, resulting in the minimal set of 38 selected collocational expressions.

Table 10.3 Themes coverage for report 557

<i>MI value</i>	<i>theme 4</i>	<i>theme 1</i>	<i>theme 3</i>	<i>theme 2</i>	<i>theme 8</i>	<i>total</i>
16	0	0	0	0	0	0
15	1	27	1	8	1	38
14	2	70	4	17	5	98
13	10	216	12	25	16	279
12	13	432	39	40	36	560
11	16	661	64	66	50	857
10	18	903	89	88	68	1166
9	22	1080	104	104	76	1386
8	22	1164	109	117	79	1491

The minimal mutual information (MI) ensures an optimal selection of collocational expressions. If $MI \ll MI_{mini}$ then more collocational expressions are selected, resulting that make some of them will be present in more than one theme of the text and in a higher computational effort. A MI too high results in themes not being covered, and therefore, not being represented for the KD task. This is an undesirable loss of information. As one can see from Table 10.3, the lower the mutual information content value, the more collocational expressions are selected. Therefore the goal is to find the minimum one for which all themes of the text are guaranteed to be represented by at least one collocational expression. Figure 10.2 shows the cumulative frequency of the minimum mutual information value necessary to cover all the themes of a text for the whole corpus. Note that 65% of the 557 texts of the corpus have all their themes covered for a mutual information less than 12. Few texts require a large mutual information value to be able to minimally cover all their themes. Therefore, there is a few number of texts requiring a large number of collocational expressions.

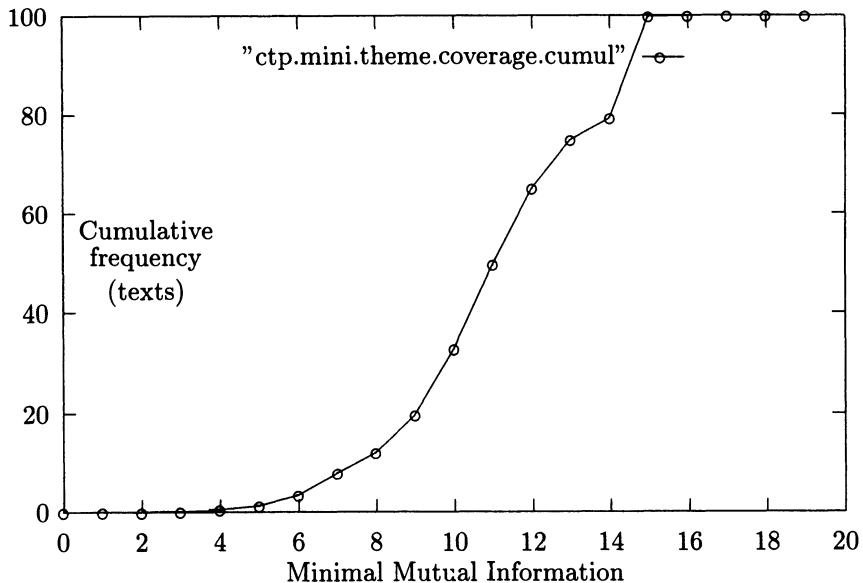


Figure 10.2 Cumulative frequency for minimum themes coverage in corpus, $\alpha = 3$ and $\beta = 4$

10.4.3 “Quality” of the Extracted Collocational Expressions

Tables 10.4 and 10.5 show the top 10 and the bottom 10 best collocational expressions for a typical psychiatric report. First, one can notice that the frequencies of the selected collocational expressions are very low within a text. Even single word frequencies are quite low, for example, *patient* appears 73 times in text 557. The mutual information discriminates each theme by not selecting collocational expressions when the frequencies of their constituents are high with respect to the text. For example, the collocational expressions containing the constituent *patient* may not be good discriminatory ones, due to the fact that the word *patient* tends to be over represented in this report. That is, it is surely present in many, if not all, sections of the text. The collocational expressions best discriminating a text’s themes are those found in one or two occurrences in the whole text, that is, only in the theme. For example, the collocational expression *patient.describes.anxiety* in the second table could have been an interesting collocational expressions (from its meaning), but it was not selected because it was not informative, that is, although it was present only once in the text, its constituents were spread all over the text. These two tables clearly show that the “best” collocational expressions are actually those conveying the most informative, or interesting, message, from a human interpretation point of view. For example, the collocational expression *intended.suicide.events*, from the first table, is much more “interesting” than *was.age.was* from the second table. This is an intuitive evaluation of

interestingness including the meaning aspect as discussed in a prior section on information and entropy. As previously mentioned, this is not quantifiable mathematically.

Table 10.4 Top 10 extracted collocational expressions

MI value	$f(word_1)$	$f(word_2)$	$f(word_3)$	c_i
15.0759	1	1	1	auditory_visual_hallucinations
15.0759	1	1	1	acts_rage_anger
15.0759	1	1	1	describe_waking_sweat
15.0759	1	1	1	hiv_test_unaware
15.0759	1	1	1	hospital_admissions_psychiatry
14.3828	2	1	1	attempts_run_away
14.3828	1	1	2	adolescence_dual_personality
14.3828	1	2	1	brief_hospitalization_detox
14.3828	1	2	1	disease_including_gonorrhea
14.3828	1	2	1	intended_suicide_events

Table 10.5 Bottom 10 extracted collocational expressions

MI value	$f(word_1)$	$f(word_2)$	$f(word_3)$	c_i
5.3873	73	13	17	patient_denies_having
5.32699	34	28	18	indicates_has_rather
5.24093	29	28	23	states_has_not
5.15068	8	73	35	symptoms_patient_describes
5.03289	9	73	35	currently_patient_describes
4.95652	73	34	10	patient_indicates_had
4.88557	73	5	73	patient_behavior_patient
4.69416	73	34	13	patient_indicates_anxiety
4.09462	73	35	23	patient_was_not
3.70376	73	34	35	patient_indicates_was

10.4.4 Role of the Collocational Expression Parameters

We aim at answering the following question: do α and β play a role for contextual text processing? In order to study the extraction of relevant information by examining collocates in a given span, we compared various combinations of the span length $\beta = \{2, 3, 4, 5\}$ with the number of collocates $\alpha = \{2, 3\}$. These values correspond to the cognitive findings we mentioned earlier, evaluated to approximately 5 words (forward and backward) the span in which a collocate to a given word will occur in a text. Considering more words for

either the collocational expressions or for the spans does not make sense, linguistically speaking, since correlated words tend to be used close to each others (short loci). Table 10.6 summarizes some statistics in extracting collocational expressions for the various combinations, also depicted in Figure 10.3. From

Table 10.6 Extraction of collocational expressions for various α and β

α	β	<i>total collocations</i>	<i>different collocations</i>	<i>max frequency</i>	$H_{\alpha,\beta}(X)$
2	2	396988	202873	1150	16.369
2	3	699367	371483	1136	17.377
2	4	948400	488465	1189	17.7574
2	5	1564748	783864	1487	18.3956
3	3	387319	325230	244	18.0172
3	4	799851	684171	283	19.128
3	5	2376222	2057282	312	20.7503

in this table, it is clear and predictable that the larger are α and β the more collocational expressions are extracted.

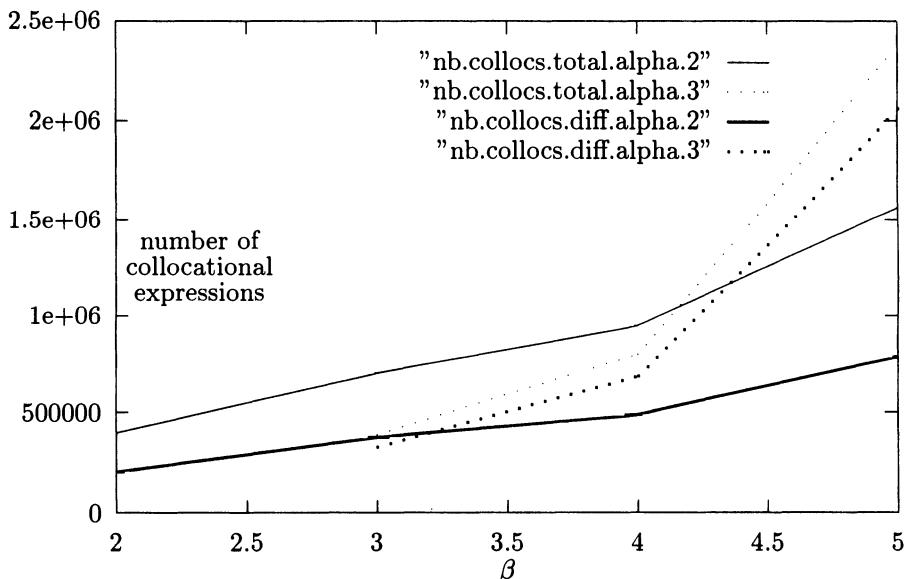


Figure 10.3 Extraction of collocational expressions for various α and β

Overall Entropy. In order to compare each combination α, β , we computed the entropy $H_{\alpha,\beta}(X)$ assuming that all collocational expressions are values of

a single discrete random variable X . Recall that the entropy is given by:

$$H_{\alpha,\beta}(X) = - \sum_{i=1}^n p_i \log_2(p_i)$$

where n is the total number of collocational expressions and p_i is the probability that X is the collocational expression i , which is simply defined by its frequency of occurrence in the corpus. Table 10.6 summarizes the entropies for combinations of α and β . The idea is to evaluate the mean amount of information for

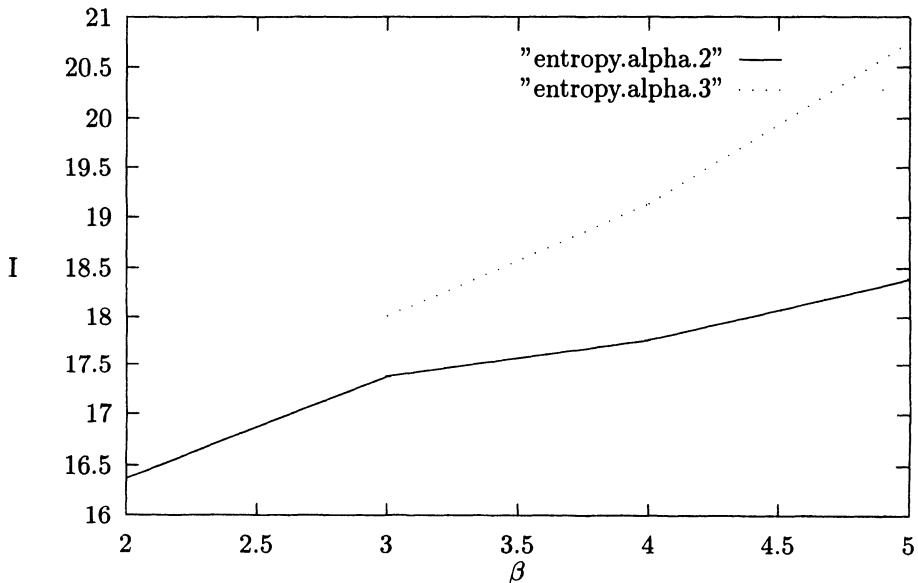


Figure 10.4 Entropy for various α and β

each combination and decide, with respect to this measurement, which one is more advantageous. The results suggest that the highest α and the highest β then the highest the gain of information. As expected, the combination $\alpha = 3$ and $\beta = 5$ seems to give the best results, with respect to the representation of the content of the corpus. The drawbacks are the larger number of collocations and therefore a higher computational effort (time). Table 10.7 shows the most frequent collocational expressions for $\alpha = 2$ and $\beta = 2$ for a typical psychiatric report. Table 10.8 is for the whole corpus. It is interesting to note that in both case, the most frequent constructs do not seem to have the most “interesting” meaning, or content. Since in most cases, the individual words are frequently used as well, the mutual information of those constructs will be very low.

10.5 CONCLUSION

We have suggested that the relative distance of terms within a text gives information about its topic structure and its relevant content. Furthermore, we

Table 10.7 Most frequent collocational expressions for $\alpha = 2$ and $\beta = 2$ for a typical report

frequency	expression
15	<i>patient.describes</i>
14	<i>patient.indicates</i>
9	<i>patient.states</i>
9	<i>substance.abuse</i>
8	<i>indicates.having</i>
7	<i>patient.was</i>
6	<i>patient.denies</i>
5	<i>denies.particular</i>
5	<i>describes.being</i>
5	<i>has.had</i>

Table 10.8 Most frequent collocational expressions for $\alpha = 2$ and $\beta = 2$ for the whole corpus

frequency	expression
1150	<i>year.old</i>
1072	<i>has.had</i>
973	<i>does.not</i>
869	<i>patient.was</i>
805	<i>did.not</i>
761	<i>patient.has</i>
476	<i>patient.reports</i>
471	<i>patient.describes</i>
459	<i>has.not</i>

suggested that this structure can be used to discover implicit knowledge embedded in the text, and therefore serves as a good candidate to represent effectively the text content for knowledge elicitation tasks. We demonstrated that a useful text structure and content can be systematically extracted by collocational lexical analysis without the need of any extra sources of knowledge. We have described algorithms to model autonomous extraction of relevant facts from a text and assign each with a theme label obtained from the automatic extraction of the topic structure discourse. The algorithm is fully implemented and makes no use of extra knowledge (e.g., lexicon, knowledge of the language, domain knowledge, or other inference mechanisms). The advantages of our approach are:

- Statistical methods to extract and sort (mutual information) collocational expressions are very attractive, mainly because they are simple and are domain independent – they only depend on the actual content of a text.
- There is no deep parsing of the texts, which reduces considerably the complexity of utilizing such information because of the fact that the texts are narrative and therefore contains jargon, non-well-formed sentences, etc. which is a real challenge for actual NLP parsers and grammars.

The limitations are:

- The proposed text content extraction algorithm is blind with respect to the semantics of the word correspondences it makes. It only gives an abstract result of the most relevant word associations a human reader would outline from the text, although these associations reflect the semantic context of natural language.
- This approach requires a large amount of computational effort to extract the collocational expressions and to evaluate the mutual information.

We have shown then an effective and “cheap” method to extract facts reflecting the content of the text (facts) and their automatic labeling reflecting their context without the need of an oracle or any manually constructed training sets, but only based on the text discourse structure and contextual information. It works for the class of text we have considered and is best suited for the intended use of this information, such as knowledge discovery tasks. This may not be suitable for tasks requiring deeper understanding using the actual meaning of each phrasal component. We proposed to use FOL to represent the text content for KD tasks for two main reasons: it has almost never been used in KDD research (which traditionally use the propositional language) and its expressive power allows to discover more interesting findings (Perrin, 1997). This approach, despite possible corrections, is still more efficient than extracting facts by hand and more robust than attempting to parse large sets of any long texts. These are necessary conditions for KDD techniques to be practical in real world applications. Finally, we reported results on the psychiatric database which have demonstrated the qualitative validation of the model.

References

- Akman, V. and Surav, M. (1996). Steps toward formalizing context. *AI Magazine*, 17(3):55–72.
- Allen, J. (1995). *Natural Language Understanding*. Benjamin Cummings, Redwood City, CA.
- Alterman, R. and Bookman, L. (1990). Some computational experiments in summarization. *Discourse processes*, 13:143–174.
- Applebaum, D. (1996). *Probability and information: an integrated approach*. Cambridge University Press, Cambridge, UK.
- Brill, E. and Mooney, R. J. (1997). An overview of empirical natural language processing. *AI Magazine*, 18(4):13–24.
- Brown, P. and et al., J. C. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.
- Cardie, C. (1997). Empirical methods in information extraction. *AI Magazine*, 18(4):65–79.
- Chen, H. and Lynch, K. J. (1992). Automatic construction of networks of concepts characterizing document databases. *IEEE Transactions on Systems, Man and Cybernetics*, 22(5):885–902.
- Church, K., Gale, W., Hanks, P., and Hindle, D. (1991). Using statistics in lexical analysis. In Zernick, U., editor, *Lexical acquisition: exploiting on-line resources to build a lexicon*, chapter 6, pages 115–164. LEA, Hillsdale, NJ.
- Feldman, R. and Dagan, I. (1995). Knowledge discovery in textual databases (kdt). In Fayyad, U. and Uthurusamy, R., editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 112–117.
- Harter, S. P. (1992). Psychological relevance and information science. *Journal of the American Society for Information Science*, 43:602–615.

- Jacobs, P. S., editor (1992). *Text-Based Intelligent Systems: current research and practice in information extraction*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- Jacobs, P. S. (1993). Using statistical methods to improve knowledge-based news categorization. *IEEE Expert*, 8(2):13–23.
- Lapalut, S. (1995). Text clustering to support knowledge acquisition from documents. Research Report 2639, INRIA.
- Lehnert, W., Soderland, S., Aronow, D., Feng, F., and Shmueli, A. (1995). Inductive text classification for medical applications. *Journal for Experimental and Theoretical Artificial Intelligence*, 7(1):271–302.
- Manning, C. (1993). Automatic acquisition of large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 235–242.
- Moulin, B. and Rousseau, D. (1991). Extracting logical knowledge from prescriptive texts in order to build deontic knowledge bases. In *Proceedings of the 6th Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pages 1–20.
- Moulin, B. and Rousseau, D. (1992). Automated knowledge acquisition from regulatory texts. *IEEE Expert*, 7(5):27–35.
- Perrin, P. (1997). *Contextual Representation and Learning for Unsupervised Knowledge Discovery in Texts*. PhD thesis, Computer Science Department, Tulane University, New Orleans, LA.
- Perrin, P. and Petry, F. (1994). Intelligent agents for scientific databases. In *Proceedings of the 5th Symposium on Computer Science, Merida, Mexico*.
- Perrin, P. and Petry, F. (1998). Contextual text representation for unsupervised knowledge discovery in texts. In *Proceedings of the Second Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-98), Melbourne, Australia, 15-17 April 1998*.
- Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In Press, A. P., editor, *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1044–1049, Menlo Park, CA.
- Salton, G. (1989). *Automatic text processing*. Addison-Wesley, Reading, MA.
- Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technology Journal*, 27:379–423, 623–656.
- Soderland, S., Fisher, D., Aseltine, J., and Lehnert, W. (1995). Crystal: inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, Montreal, Canada. AAAI, Morgan Kaufmann.
- Sperber, D. and Wilson, D. (1986). *Relevance: communication and cognition*. Basil Blackwell, Oxford, UK.

11 INTEGRATED FEATURE EXTRACTION USING ADAPTIVE WAVELETS

Yvette Mallet¹, Olivier de Vel², and Danny Coomans¹

¹Department of Mathematics and Statistics,
James Cook University,
Townsville 4811, Australia
{Yvette.Mallet|Danny.Coomans}@jcu.edu.au

²Department of Computer Science,
James Cook University,
Townsville 4811, Australia
olivier@cs.jcu.edu.au

Abstract: On-line quality control in the manufacturing and processing industry is increasingly being undertaken by analyzing collinear data such as spectra sampled by a spectrometer. Spectral data are very highly dimensional and are characterized by having highly correlated, localized structures. Wavelets are therefore most effective in extracting the important local features in spectra by reducing the number of variables whilst, at the same time, retaining as much information as possible and facilitating the automated analysis and interpretation of spectra. There are many kinds of wavelets which exist in the literature, but the fundamental problem to overcome is deciding which wavelet will produce the best results for a particular application. Rather than using an 'off-the-shelf' wavelet, an automated search is performed for the wavelet which optimizes specified multivariate modeling criteria. The spectral data analyzed in this chapter are of importance to the agricultural, pharmaceutical and mining industries as well as the environmental sciences.

11.1 INTRODUCTION

The popularity of spectroscopy has increased enormously over the past few decades. Such popularity can be attributed to the fact that spectroscopy pro-

vides a relatively efficient and non-destructive procedure for analyzing substances. This is of great benefit to research and quality control procedures in industry. Spectral data are obtained by spectrometers which measure the change in reflected or absorbed radiation which has been directed at some substance for hundreds of wavelengths, usually at regular intervals, where each wavelength can be considered equivalent to a variable or attribute. Such data are complex and extremely multivariate, consisting of many highly correlated variables or features. In most application areas, the important characteristics of a spectrum are identified by the localized peak patterns or global baseline trends. The extraction of such features which represent the morphology and/or shifts in spectral peaks are important for interpreting any variations in spectral characteristics which may be due to changes in the chemical composition of a substance. Process monitoring in the manufacturing and processing industry often requires, for example, the continual evaluation of the level of impurities in a given product by means of spectrometry. The introduction of a chemical impurity in a particular substance may be identified by local changes in the relative magnitudes of the peaks or by the introduction of shoulder peaks in some section of the spectrum.

Processing of spectral data sets often involves statistical modeling and multivariate data analysis. Depending on the problem at hand, techniques applied include: regression, dimensionality reduction, discriminant analysis, gradient analysis (ordination) or cluster analysis. There are several difficulties which arise from analyzing spectral data sets. One of the major problems is that the dimensionality (i.e. number of wavelengths or variables) is usually quite large, especially when compared to the number of available spectra. Consequently the estimation of parameters becomes highly variable and, in some instances, unobtainable due to numerical instabilities leading to a substantial performance degradation of the multivariate statistical model. A second problem is the existence of a high correlation structure in spectral data owing to the presence of a strong ordering in the features. Such problems are not limited to spectral data, but can also be found in more complex data sets such as image data.

There are some statistical methods which have evolved in recent years with the aim of combating problems associated with high dimensionality and high correlation structures. Such techniques are referred to as high-dimensional techniques and include regularized discriminant analysis (RDA) and penalized discriminant analysis (PDA) in classification or ridge regression in the modeling of spectral data (McLachlan, 1992)(Friedman, 1989)(Hastie et al., 1995). Conversely, techniques which begin to fail when the dimensionality becomes large when compared to the sample sizes are referred to as low-dimensional techniques. The high-dimensional methods generally allow for a more automated modeling procedure. Unfortunately though, most high-dimensional methods have evolved quite recently and are therefore not as well publicized or understood by the scientific or industrial community (Aeberhard et al., 1994). Also, it can be more difficult to apply high-dimensional techniques since they are generally not standard procedures in most mathematical or statistical toolbox

packages. Finally, most of these techniques provide few facilities for aiding the interpretation of the resulting multivariate prediction model. For these reasons, low-dimensional methods are often preferred.

Before low-dimensional statistical methods are applied, some form of feature extraction should be implemented prior to the analyses. Feature extraction is a mapping of a data vector X into a lower-dimensional feature vector Y (where $\text{length}(Y) \ll \text{length}(X)$). This mapping can be achieved by a linear or non-linear transformation, the transformation chosen so as to retain as much relevant information as possible, which is dependent on the data analysis technique advocated. Having fewer variables often means results can be obtained with reduced computational and economical expense. Another reason for extracting features may simply be that the features are more meaningful than the raw data thereby enhancing the interpretability of the data. Feature extraction can consist of two main components (see Figure 11.1): the variable transformation algorithm and the feature selection algorithm.

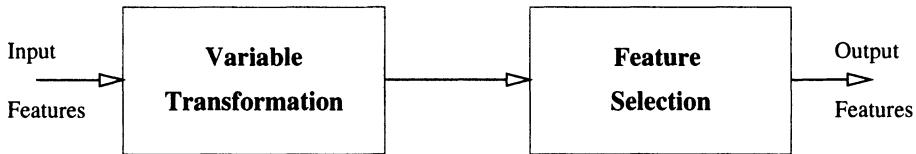


Figure 11.1 Feature extraction model.

Feature extraction (variable transformation and/or feature selection) can either be independent of, or integrated with, the statistical prediction model. An *integrated feature extraction* procedure makes use of the criterion function(s) of the statistical model (e.g. loss functions) whereas *independent feature extraction* uses criterion functions which are not directly related to the statistical prediction model (see Figure 11.2). An example *independent feature extraction* procedure is to perform a discriminant analysis on principal components. This procedure is independent because the principal components are obtained by maximizing variability which is independent of the assignment or allocation criterion which are more related (or integrated) with discriminant analysis.

The *integrated feature extraction* procedure which we consider, continually updates the features until some modeling criterion becomes optimal. For instance, if we were performing regression analysis on the spectra we may choose to find the features which optimize criterion based on residual sum of squares, or, a criterion for classification may involve misclassification rates. We propose a novel approach based on a concise and complete parameterization of the adaptive wavelet matrix which forms the basis of an integrated multivariate variable transformation (Turcajova and Kautsky, 1994). The actual features which we continually update are wavelet coefficients. The wavelet coefficients are produced from a wavelet which is called an adaptive or task-specific wavelet since it is ‘adapting’ to the current task. The adaptiveness is obtained

by searching for optimal wavelets trained on a particular spectral data set and using a given statistical model criterion function (e.g. cost function) that reflects the performance of the statistical model. We describe adaptive wavelets in the next section.

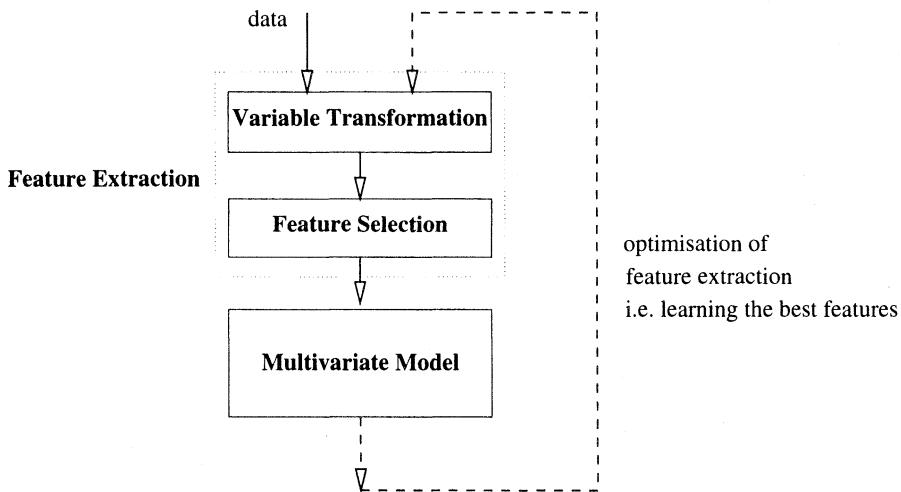


Figure 11.2 Integrated feature extraction.

Many variable transforms have been proposed for spectral data ranging from univariate to multivariate transformations involving all the variables of the spectrum. A multivariate technique commonly used is principal components analysis (PCA). PCA transforms the original variables into a new set of uncorrelated variables that are linear combinations of the original variables derived in decreasing order of variability. Of particular importance with spectral data is the order of the wavelengths. Unfortunately PCA does not take into account the ordering, and hence the correlational structure of the variables within a spectrum. Alternatively, the Fourier transform (FT) can be used to take into account the ordering of variables in a spectrum. However, the FT is a global transform and any localized changes in a spectrum will affect most, if not all, of the Fourier coefficients. To avoid such global effects and to better identify local absorption-wavelength effects, the wavelet decomposition can be used for variable transformation. Basically a wavelet decomposition is a basis function expansion into localized contributions labeled by scale and a position (wavelength) parameter. The wavelet coefficients are able to convey small scale effects in spectral characteristics and can, therefore, identify any variations in peak shape or position. The peaks need not necessarily be small-scale variations, but could be larger scale fluctuations such as the spectral baseline. Also, one useful property of the wavelet transform is its ability to back-transform the wavelet coefficients, thereby enabling an improved interpretation of the relevant (extracted) features.

11.2 WAVELET ANALYSIS OF SPECTRA

When dealing with spectral data we can represent each spectrum as a set of “building blocks” or basis functions. In the context of spectra, these basis functions should attempt to reflect as much as possible the different degrees of “scale” and localization in space (along the wavelength axis) that are inherent in a spectrum. That is, any global trends in the spectral characteristics such as slowly-varying baseline shifts or small-scale local trends such as the introduction of spectral peak shifts or peak broadening should be identified by different sets of wavelet coefficients.

Wavelets are small waves which oscillate above and below the horizontal axis and integrate to zero. Wavelets form a set of basis functions which can be used to represent functions from the class of square integral functions which have finite energy. The set of basis functions are derived by translating and dilating one basic wavelet, called a mother wavelet. The dilated and translated versions of the mother wavelet are called children wavelets. The coefficients in the expansion of the wavelet basis functions are calculated by the wavelet transform. These coefficients are referred to as wavelet coefficients.

We can introduce wavelets in different ways. For example, through multiresolution analysis (MRA), we can construct a variety of wavelets in a unified setting. MRA is convenient for deriving computationally-efficient lattice decomposition and reconstruction equations of the wavelet transform for infinite signals. The lattice decomposition equations are given by (Mallat, 1989):

$$f_{j-1,p}^{(0)} = \sum_{k \in \mathbb{Z}} f_{j,p}^{(0)} l_{k-mp} \quad \text{and} \quad f_{j-1,p}^{(s)} = \sum_{k \in \mathbb{Z}} f_{j,p}^{(0)} h_{k-mp}^{(s)}$$

for $s = 1, 2, \dots, m - 1$. The lattice reconstruction equation is formulated as:

$$f_{j,k}^{(0)} = \sum_{p \in \mathbb{Z}} f_{j-1,p}^{(0)} l_{k-mp} + \sum_{s=1}^{m-1} \sum_{p \in \mathbb{Z}} f_{j-1,p}^{(s)} h_{k-mp}^{(s)}$$

where $f_{j,k}$ are the wavelet coefficients, m is the dilation parameter, and l_k and $h_k^{(s)}$ are the sets of “low-pass” and “high-pass” filter coefficients, respectively, which are subject to certain regularity conditions (see later). The low-pass filter acts as a smoother, which produces a smoothed version of the data sequence which it is filtering. The $m - 1$ high-pass filters act as differencing operators, which produce the high frequency components of the signal which the low-pass filter did not capture. The multi-band decomposition scheme can be viewed as an unbalanced m -way tree, with each tree level corresponding to an iteration of the lattice decomposition equations (Mallet et al., 1997).

In practice, a value $j = j_0$ is chosen (the top level in the decomposition scheme) corresponding to the input signal or spectrum sequence. Also, since the input signal is usually finite (with length N), the sequence $\{f_{j_0,k}\}_{k \in \mathbb{Z}}$ is equated to the periodic extension of the input discrete signal. This gives rise to the discrete wavelet transform (DWT).

11.2.1 The Discrete Wavelet Transform (DWT)

Since the wavelet coefficients are the features which are eventually supplied to the statistical procedures, we now focus our attention on how the wavelet coefficients are calculated. The procedure for calculating the wavelet coefficients is called the discrete wavelet transform .

For convenience, we can rewrite the decomposition/reconstruction equations of the DWT as

$$\mathbf{f}_{j-1} = C_N \mathbf{f}_j^{(0)} \quad \text{and} \quad \mathbf{f}_j^{(0)} = C_N^T \mathbf{f}_{j-1}$$

where the vectors $\mathbf{f}_j^{(0)}$ and \mathbf{f}_{j-1} are mN -periodic

$$\mathbf{f}_j^{(0)} = (f_{j,0}^{(0)} f_{j,1}^{(0)} \cdots f_{j,mN-1}^{(0)})^T$$

$$\begin{aligned} \mathbf{f}_{j-1} &= [(f_{j-1,0}^{(0)} f_{j-1,1}^{(1)} \cdots f_{j-1,m-1}^{(m-1)}) (f_{j-1,1}^{(0)} f_{j-1,2}^{(1)} \cdots f_{j-1,N-1}^{(m-1)}) \cdots \\ &\quad (f_{j-1,N-1}^{(0)} \cdots f_{j-1,N-1}^{(m-1)})]^T \end{aligned}$$

C_N is a block circulant matrix generated by the wavelet matrix \mathbf{A} (where $q = N/m$) and has the special “banded” structure:

$$C_N = \begin{pmatrix} \mathbf{A}_0 & \mathbf{A}_1 & \dots & \mathbf{A}_{q-2} & \mathbf{A}_{q-1} & 0 \\ 0 & \mathbf{A}_0 & \dots & \mathbf{A}_{q-3} & \mathbf{A}_{q-2} & \mathbf{A}_{q-1} \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ \mathbf{A}_1 & \mathbf{A}_2 & \dots & \mathbf{A}_{q-1} & 0 & \mathbf{A}_0 \end{pmatrix}$$

where \mathbf{A} is composed of q sub-matrices \mathbf{A}_i (for $i = 0, 1, \dots, q - 1$) each of size $m \times m$ such that $\mathbf{A} = (\mathbf{A}_0 \mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_{q-1})$. The wavelet matrix \mathbf{A} denotes the matrix of filter coefficients, with the first row containing the low-pass coefficients and the remaining $m - 1$ rows, the sets of high-pass coefficients. Certain regularity conditions on the wavelet matrix \mathbf{A} must be satisfied if C_N is to be a banded orthogonal block circulant matrix:

- Shifted orthogonality condition: $\sum_{p \in \mathbb{Z}} \mathbf{A}_p \mathbf{A}_{k+p}^T = \delta_{0k} \mathbf{I} \quad k \in \mathbb{Z}$
- Basic regularity condition: $\sum_{p \in \mathbb{Z}} l_p = \sqrt{m}$
- The Lawton matrix $[M]_{ij} = \sum_k l_k l_{k+j-mi}$ must have a simple eigenvalue equal to 1.

In practice, “off-the-shelf” values for the low- and high-pass filter coefficients can be chosen from the literature. These are readily available more so for the situation when $m = 2$, for example the Daubechies wavelet family. We believe, that it can be advantageous to design your own task specific filter coefficients rather than using a predefined set. The problem we pose is to be able to describe a large class of orthogonal compactly-supported wavelets with a given length of support and design the wavelet matrix \mathbf{A} which optimizes some specified

modeling criterion relevant to a given multivariate prediction model, such as regression or discriminant analysis. Instead of optimizing over each element in \mathbf{A} , we make use of the factorized form (Turcanova and Kautsky, 1994) of a wavelet matrix and the regularity conditions placed therein to reduce the number of parameters to be optimized. It can be shown that it is possible to construct \mathbf{A} from a set of normalized vectors, denoted by $\mathbf{u}_1, \dots, \mathbf{u}_q$ and \mathbf{v} . Provided that the shifted orthogonality condition is satisfied, the wavelet matrix can also be written in the factorized form (Turcanova and Kautsky, 1994)

$$\mathbf{A} = \mathbf{Q} \square \mathbf{F}_1 \square \cdots \square \mathbf{F}_{q-1}. \quad (11.1)$$

The symbol \square denotes the “polynomial product” which is defined by

$$(\mathbf{B}_0 \ \mathbf{B}_1 \ \dots \ \mathbf{B}_{r-1}) \square (\mathbf{C}_0 \ \mathbf{C}_1 \ \dots \ \mathbf{C}_{s-1}) = (\mathbf{G}_0 \ \mathbf{G}_1 \ \dots \ \mathbf{G}_{r+s-2})$$

with

$$\mathbf{G}_i = \sum_k \mathbf{B}_k \mathbf{C}_{i-k}.$$

The factors $\mathbf{F}_i = (\mathbf{R}_i \ \mathbf{I} - \mathbf{R}_i)$ where \mathbf{R}_i is a projection matrix and $\mathbf{Q} = \sum_i \mathbf{A}_i$ is an orthogonal matrix. If, for example, $m = 4$ and $q = 3$ then $\mathbf{A} = (\mathbf{A}_0 \ \mathbf{A}_1 \ \mathbf{A}_2)$ with each \mathbf{A}_j having size 4×4 and \mathbf{A} having size $m \times (m \times q) = 12$.

We have

$$\begin{aligned} \mathbf{A} &= \mathbf{Q} \square \mathbf{F}_1 \square \mathbf{F}_2 \\ &= \mathbf{Q} \square (\mathbf{R}_1 \ \mathbf{I} - \mathbf{R}_1) \square (\mathbf{R}_2 \ \mathbf{I} - \mathbf{R}_2) \\ &= [\mathbf{Q}\mathbf{R}_1\mathbf{R}_2 \ \mathbf{Q}(\mathbf{R}_1 - 2\mathbf{R}_1\mathbf{R}_2 + \mathbf{R}_2) \ \mathbf{Q}(\mathbf{I} - \mathbf{R}_1)(\mathbf{I} - \mathbf{R}_2)] \\ &\equiv (\mathbf{A}_0 \ \mathbf{A}_1 \ \mathbf{A}_2). \end{aligned}$$

Essentially, we strive for representations of \mathbf{Q} and each projection matrix \mathbf{R}_i (for $i = 1, \dots, q-1$). First consider the representation of \mathbf{Q} .

An orthogonal matrix may be expressed as $\mathbf{I} - 2\mathbf{v}\mathbf{v}^T$ where \mathbf{v} is a normalized vector. The basic regularity condition however, places a constraint on the first row of \mathbf{Q} . This condition is equivalent to setting the first row of \mathbf{Q} to $1/\sqrt{m} \mathbf{1}$ where $\mathbf{1}$ denotes the vector $(1, 1, \dots, 1)$. The remaining $m-1$ rows are calculated ensuring the orthogonality of \mathbf{Q} is maintained. This is satisfied if the last $m-1$ rows are calculated by $(\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{T.D}$ where \mathbf{T} is an upper triangular matrix with diagonal elements $T_{ii} = i-m$ and off-diagonal elements equal to 1. The matrix \mathbf{D} normalizes the rows of \mathbf{T} . The $m \times m$ orthogonal matrix \mathbf{Q} is partitioned as follows,

$$\mathbf{Q} = \begin{pmatrix} 1/\sqrt{m} \mathbf{1} \\ (\mathbf{I} - 2\mathbf{v}\mathbf{v}^T)\mathbf{T.D} \end{pmatrix} \quad (11.2)$$

Now consider the projection matrices. A symmetric projection matrix of rank ρ can be written $\mathbf{R} = \mathbf{U}\mathbf{U}^T$ where $\mathbf{U}_{m \times \rho}$ is a matrix with orthonormal

columns. For the wavelet matrix to be non-redundant, the ranks of the projection matrices must form a monotonically increasing sequence (Turcanova and Kautsky, 1994), that is the $\text{rank}(\mathbf{R}_1) \leq \text{rank}(\mathbf{R}_2) \leq \dots \leq \text{rank}(\mathbf{R}_q)$. For simplicity, we restrict the ranks of each projector matrix to be 1, and so,

$$\mathbf{R}_i = \mathbf{u}_i \mathbf{u}_i^T \quad (11.3)$$

where $\mathbf{u}_i^T \mathbf{u}_i = 1$.

We can update the vectors \mathbf{v} and \mathbf{u}_i by using a search procedure based on minimizing some application-dependent cost/criterion function. Initial values for these vectors are generally chosen to be random values. Example cost functions for discriminant analysis (see below) include misclassification rate, entropy, Wilks' Λ and the quadratic probability measure. For regression, a typical cost measure is the R-squared measure. Figure 11.3 outlines the basic steps in the construction of the adaptive wavelet. More details are given in (Mallet et al., 1997).

Construct-Adaptive-Wavelet

- [1] Initialise vectors \mathbf{v} and \mathbf{u}_i ;
- [2] Construct orthogonal matrix \mathbf{Q} ;
- [3] Construct projection matrices \mathbf{R}_i ($i = 1, 2, \dots, q - 1$);
- [4] Construct factors $\mathbf{F}_i = (\mathbf{R}_i \mathbf{I} - \mathbf{R}_i)$ ($i = 1, 2, \dots, q - 1$);
- [5] Set $\mathbf{A} = \mathbf{Q} \square \mathbf{F}_1 \square \dots \square \mathbf{F}_{q-1}$.

Figure 11.3 Basic steps for the construction of the adaptive wavelet.

11.2.2 Subset Selection of the Wavelet Coefficients

Since there are just as many wavelet coefficients as there are wavelengths in the original spectra, it is necessary to perform a subset selection on the wavelet coefficients. This can be achieved by, for example, using a best-basis algorithm which minimizes some information cost function or by selecting a set of wavelet coefficients from a predetermined band that gives the lowest cost value for the multivariate task at hand (Saito and Coifman, 1994)(Mallet et al., 1997). In our implementation we choose to select a single band of wavelet coefficients from a single level in the DWT. These coefficients (for each of the spectra) are then supplied to the statistical procedure. The modeling criterion for optimizing the wavelet matrix is also based on the same coefficients. Figure 11.4 shows a schematic diagram of the learning process involved in extracting the optimal features for a given cost criterion and multivariate model. Note that, in the case of implementing cross-validation for small data sets, an additional external loop would need to be incorporated.

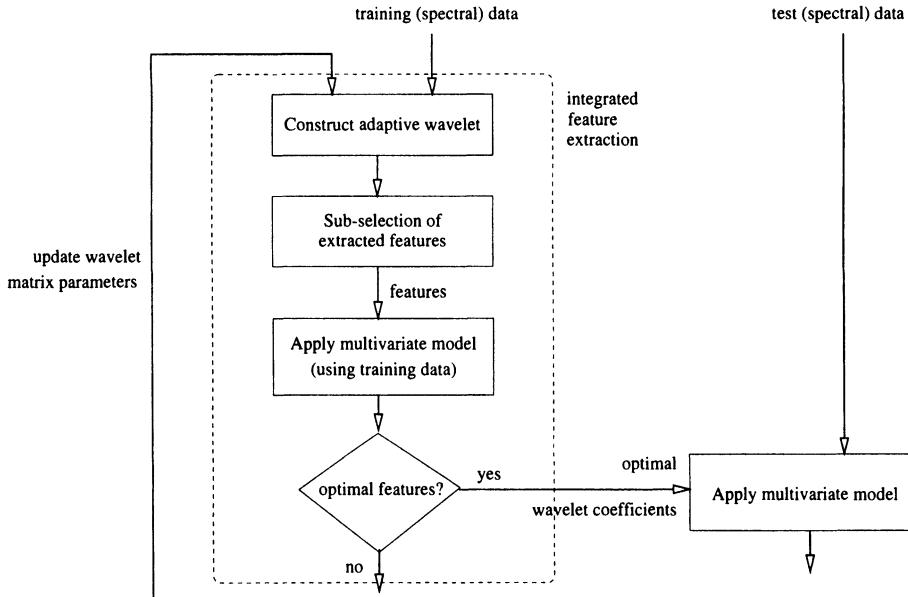


Figure 11.4 Learning the optimal features using adaptive wavelets.

11.3 MULTIVARIATE PREDICTION MODELS

Regression analysis and classification (or pattern recognition) techniques are often applied for monitoring quality control procedures in industry. Both regression and classification methods share similar goals which is to predict values of some response. For example, calibration in quality control can be performed using regression analysis and the determination of product defects in the manufacturing industry can be undertaken with classification algorithms. With regression analysis, this response variable may take on continuous values. With classification, the response variables are discrete values which represent class categories.

11.3.1 Regression

Let \mathbf{X} denote the $n \times d'$ matrix of features whose rows are the individual cases, and let $\mathbf{y} = (y_1, y_2, \dots, y_n)^T$ ($y_i \in \Re$) denote some response vector of size $n \times 1$ where n is the number of cases or training spectra. It is convenient if the response and feature matrix is centered so that the columns sum to zero. Let $\bar{\mathbf{X}}$ be the $n \times d'$ data matrix whose columns have been centered and $\bar{\mathbf{y}}$ be the centered response vector. The multiple linear regression (MLR) model estimates the response by

$$\hat{\mathbf{y}} = \bar{\mathbf{X}}\mathbf{b}$$

where $\hat{\mathbf{y}}$ denotes the predicted response vector, and \mathbf{b} are the regression coefficients calculated by

$$\mathbf{b} = (\bar{\mathbf{X}}^T \bar{\mathbf{X}})^{-1} \bar{\mathbf{X}}^T \bar{\mathbf{y}}.$$

Multiple linear regression is considered to be a low-dimensional technique, However, least-squares is unstable if the \mathbf{x} data are highly correlated which is referred to as the multi-collinearity problem.

A popular technique used in high-dimensional settings is partial least-squares (PLS). Here, the predictive variables are linearly combined to form latent predictive variables. The latent variables are determined from the predictor variables in a sequential manner such that each new latent variable has maximal covariance with the response and is uncorrelated with previously constructed latent variables (Geladi and Kowalski, 1986). PLS inherently performs a type of feature extraction as the number of latent variables is often much less than the number of dimensions.

The cost function used in finding the optimal set of wavelet coefficients is the leave-one-out cross-validated R-squared measure, R_{CV}^2 , which is calculated as:

$$R_{CV}^2 = 1 - \frac{\text{PReSS}}{\text{TSS}}$$

where PReSS and TSS are the predictive residual sum of squares and the total sum of squares, respectively.

11.3.2 Classification

Consider the classification problem of assigning an object \mathbf{x}_i which is from the i th row of the feature matrix \mathbf{X} , into one of K predefined classes. In linear discriminant analysis we can apply the Bayes rule (referred to as Bayes linear discriminant analysis, BLDA): assign data object \mathbf{x}_i to the class $c \in 1, \dots, K$, that minimizes the linear classification score

$$\text{cs}_c(\mathbf{x}_i) = (\mathbf{x}_i - \bar{\mathbf{x}}_c)^T \mathbf{S}^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}_c) - 2 \ln P(c) \quad \text{for } c = 1, \dots, K$$

where $P(c)$ is the prior probability that \mathbf{x}_i is from class c , $\bar{\mathbf{x}}_c$ is the mean vector of class c and \mathbf{S} is the pooled covariance matrix. If there are n_c data objects or spectra from class c such that $n = \sum_{c=1}^K n_c$ is the sample size of the data, then the prior probability, $P(c)$ could be estimated by $P(c) = n_c/n$. If $\mathbf{x}_{i(c)}$ indicates that \mathbf{x}_i is from class c , then and $\bar{\mathbf{x}}_c = 1/n_c \sum_{i=1}^{n_c} \mathbf{x}_{i(c)}$. The pooled covariance matrix is calculated by

$$\mathbf{S} = \frac{1}{n} \sum_{c=1}^K n_c \mathbf{S}_c$$

where

$$\mathbf{S}_c = \frac{\sum_{i=1}^{n_c} (\mathbf{x}_{i(c)} - \bar{\mathbf{x}}_c)(\mathbf{x}_{i(c)} - \bar{\mathbf{x}}_c)^T}{n_c}$$

is the covariance matrix of class c .

The modeling criteria optimized by the integrated adaptive wavelet-based feature extraction was a cross-validated quadratic probability measure (Coomans and Broeckaert, 1986).

11.4 THE DATA SETS

Four spectral data sets were used for evaluating the performance of the classification procedures and two data set are used for regression. All data sets had 512 variables. A summary of each of the classification data sets is presented in Table 11.1 and the regression data sets are described in Table 11.2. Sample spectra from the mineralogical data set are depicted in Figure 11.5.

Table 11.1 Description of the spectral data sets used for classification.

<i>Data Set</i>		<i>Class 1</i>	<i>Class 2</i>	<i>Class 3</i>	<i>Class 4</i>	<i>Class 5</i>	<i>Total</i>
Seagrass	Train	55	55	55	-	-	165
	Test	34	34	34	-	-	102
Mineral	Train	20	20	20	20	20	100
	Test	20	20	20	20	20	100
Paraxylene	Train	25	25	25	-	-	75
	Test	25	25	25	-	-	75
Butanol	Train	21	27	-	-	-	48
	Test	21	26	-	-	-	47

Table 11.2 Description of the spectral data sets used for regression.

<i>Data Set</i>	<i>Train</i>	<i>Test</i>	<i>Responses</i>
Sugar	100	89	brix , fibre
Wheat	60	40	protein

11.5 CLASSIFICATION APPLICATIONS

As well as implementing the adaptive wavelet feature extraction procedure, other low and high-dimensional classification schemes are applied. The low-dimensional classifiers are BLDA and flexible discriminant analysis (FDA). FDA is a nonlinear classifier which selects variables from the original data (Hastie et al., 1994). The BLDA classifier is used in two separate experiments, both using extracted features – one with a set of forward step-wise

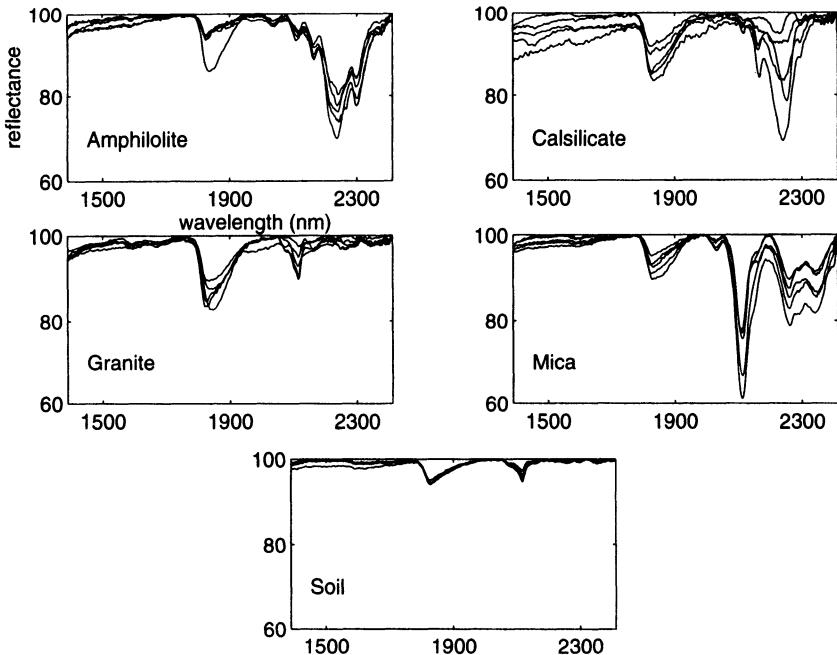


Figure 11.5 Five sample spectra from the mineral data.

selected variables as input features, the other with the adaptive wavelet coefficients as its input features (this technique is abbreviated as AWBLDA). The high-dimensional classifiers used are PDA and RDA. The correct classification rates are shown in Table 11.3.

Table 11.3 Correct classification rates (%)

<i>Data Set</i>		<i>BLDA</i>	<i>AWBLDA</i>	<i>FDA</i>	<i>PDA</i>	<i>RDA</i>
Seagrass	Train	100	100	98.2	97.0	99.4
	Test	97.1	100	99.0	95.1	99.0
Mineral	Train	99	99	100	100	100
	Test	85	99	95	100	95
Paraxylene	Train	97.3	94.7	100	86.7	100
	Test	73.3	84	86.7	81.3	100
Butanol	Train	89.6	91.7	75	43.7	87.5
	Test	89.4	83.0	70.2	43.8	87.2

It can be seen that the classification performance of BLDA has been greatly enhanced by using the wavelet coefficients (AWBLDA), with the exception of

the butanol data set (in which case BLDA does better than even the high-dimensional classifiers). Also, results for AWBLDA are consistently high and compare well with the existing techniques.

A useful property of the wavelet transform is its ability to back-transform the wavelet coefficients. If some of the coefficients in the wavelet transform are set to zero and the data is back-transformed (up to the root level of the tree), the reconstructed data highlights the information in the spectra which has been represented by the wavelet coefficients which were not set to zero. The procedure for setting the coefficients to zero is a form of hard thresholding. We can further extend the classification procedure by classifying the thresholded and reconstructed data and comparing its performance with the raw wavelet coefficients.

We now investigate how the classification performance of the discriminant techniques BLDA, FDA, RDA and PDA is affected when the reconstructed data is used as input to these procedures. The classification rates are displayed in Table 11.4. The boldface type indicates that the results on the reconstructed data (for the test set) are higher than the classification rates obtained from the original data.

When comparing the classification performance of BLDA and FDA based on the original data, and on the reconstructed data, we note that, in most cases, more favorable results are produced when classification is based on the reconstructed data. For the high-dimensional classifiers however, fewer advantages seem to be associated with the application of reconstructed data. The decrease in performance of PDA using the reconstructed data is possibly due to the fact that PDA is perhaps more suited to highly correlated data, and in the reconstruction process, much of the correlation structure has been removed. The reason why RDA has not shown any improvement is not so clear. We note that, with the exception of the paraxylene data, the results for RDA on the reconstructed data did not deteriorate as much as those for PDA. Perhaps this indicates that RDA is able to utilize much the same information from the original data as from the reconstructed data.

11.6 REGRESSION APPLICATIONS

Both multiple linear regression (MLR) and partial least-squares regression (PLS) were used for comparative analysis. Multiple linear regression was used in two separate experiments, both using extracted features – one with a set of forward step-wise selected variables as input features, the other with the adaptive wavelet coefficients as its input features (AWMLR). Table 11.5 shows the R-squared values obtained for the training and testing data for each of the methods.

AWMLR performs well in all cases by either outperforming normal MLR and PLS or by coming a close second. Furthermore, AWMLR does not overfit as much as the other techniques due to the smaller difference between the test and training results.

Table 11.4 Correct classification rates (%) for the reconstructed data.

<i>Data Set</i>		<i>BLDA</i>	<i>FDA</i>	<i>PDA</i>	<i>RDA</i>
Seagrass	Train	100	100	85.5	100
	Test	99.0	100	83.3	100
Mineral	Train	100	100	99	99
	Test	92	98	97	93
Paraxylene	Train	94.7	94.7	73.3	94.7
	Test	82.7	84.0	65.3	84
Butanol	Train	87.5	85.4	43.8	85.4
	Test	87.2	87.2	44.7	85.1

Table 11.5 R-squared values.

<i>Data Set</i>		<i>AWMLR</i>	<i>MLR</i>	<i>PLS</i>
Brix	Train	0.975	0.980	0.977
	Test	0.971	0.963	0.971
Fibre	Train	0.869	0.905	0.898
	Test	0.818	0.819	0.805
Protein	Train	0.978	0.989	0.977
	Test	0.836	0.775	0.814

As with the results for classification, we decided to determine if the R-squared values could be improved by performing MLR on the reconstructed spectra (see Table 11.6). A noticeable feature in Table 11.6 is the consistent improvements to the prediction of protein when the regression is based on the reconstructed data. The prediction of fibre however achieves better performances when the spectra is in its original form.

11.7 FUTURE DIRECTIONS

Future developments will concentrate on investigating better and faster adaptive wavelet search algorithms, investigating the impact of the wavelet decomposition scheme (e.g. depth and number of bands) on performance, and obtaining better heuristics for the subset selection of wavelet coefficients.

Table 11.6 R-squared values for the reconstructed data.

<i>Data Set</i>		<i>MLR</i>	<i>PLS</i>
Brix	Train	0.975	0.975
	Test	0.971	0.971
Fibre	Train	0.713	0.745
	Test	0.591	0.607
Protein	Train	0.978	0.978
	Test	0.837	0.834

References

- Aeberhard, S., Coomans, D., and de Vel, O. (1994). "Comparative analysis of pattern classifiers in a high dimensional setting". *Pattern Recognition*, 24:1065–1077.
- Coomans, D. and Broeckaert, I. (1986). *Potential Pattern Recognition in Chemical and Medical Decision Making*. John Wiley.
- Friedman, J. (1989). "Regularized discriminant analysis". *Journal of the American Statistical Association*, 84:165–175.
- Geladi, P. and Kowalski, B. (1986). "Partial least-squares regression: A tutorial". *Analytica Chimica Acta*, 185:1–17.
- Hastie, T., Buja, A., and Tibshirani, R. (1995). "Penalized discriminant analysis". *Annals of Statistics*, 23:73–102.
- Hastie, T., Tibshirani, R., and Buja, A. (1994). "Flexible discriminant analysis by optimal scoring". *Journal of the American Statistical Association*, 89:1255–1270.
- Mallat, S. (1989). "A theory for multiresolution signal and decomposition: the wavelet representation". *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11:674–693.
- Mallet, Y., Coomans, D., and de Vel, O. (1997). "Classification using adaptive wavelets for feature extraction". *IEEE Trans. on Pattern Analysis and Machine Intelligence*. to appear.
- McLachlan, G. (1992). *Discriminant Analysis and Statistical Pattern Recognition*. John Wiley.
- Saito, N. and Coifman, R. (1994). *Local discriminant bases*. In *Mathematical Imaging: Wavelet applications in signal and image processing*. Eds A. Laine and M. Unser, SPIE 2303.
- Turcova, R. and Kautsky, J. (1994). "Shift products and factorizations of wavelet matrices". *Numerical Algorithms*, 8:27–45.

12 FEATURE EXTRACTION VIA NEURAL NETWORKS

Rudy Setiono and Huan Liu

Department of Information Systems & Computer Science
National University of Singapore
Singapore 119260

{rudys|liuh}@iscs.nus.edu.sg

Abstract: A method for feature extraction which makes use of feedforward neural networks with a single hidden layer is presented. The topology of the networks is determined by a network construction algorithm and a network pruning algorithm. Network construction is achieved by having just 1 hidden unit initially; additional units are added only when they are needed to improve the network predictive accuracy. Once a fully connected network has been constructed, irrelevant/redundant network connections are removed by pruning. The hidden unit activations of the pruned network are the features extracted from the original dataset. Using artificial datasets, we illustrate how the method works and interpret the extracted features in terms of the original attributes of the datasets. We also discuss how the feature extraction method can be used in conjunction with other learning algorithms such as decision tree methods to obtain robust and effective classifiers.

12.1 INTRODUCTION

Classification is a task to learn from a training dataset S in which each instance is represented by feature-values with a class label, and to predict class labels of the instances outside S . We look for hypotheses (based on which we make our prediction) that are not only consistent within S , but also generalize beyond S . Selective induction algorithms such as the classic decision tree induction methods (Breiman et al., 1984; Quinlan, 1986; Clark and Niblett, 1989) suffer from problems of replication and fragmentation (Matheus and Rendell, 1989; Pagallo and Haussler, 1990). It is also observed that some classification tasks are more difficult than others since their concepts are more difficult to learn. Concept variation (Rendell and Seshu, 1990) has been used to measure

the degree of difficulty of a concept. It provides an estimate of the probability that any two neighbor instances differ in class value, roughly measuring the amount of irregularity in the distribution of instances along the instance space. In (Vilalta et al., 1997), the authors showed that more difficult datasets have higher concept variation; and the fragmentation problem adversely affects predictive accuracy as concept variation of a dataset increases.

Different representations can help alleviate these problems or make the learning task easier. Among many, compound features (Pagallo and Hausler, 1990), constructive induction (Matheus and Rendell, 1989; Wnek and Michalski, 1994), multivariate decision trees (Murthy et al., 1993; Brodley and Utgoff, 1995), lazy decision trees (Friedman et al., 1996), rule sets (Quinlan, 1987), decision graphs (Kohavi, 1994; Oliveira, 1995), discretization of continuous data (Fayyad and Irani, 1993; Liu and Setiono, 1995) are some examples. In this chapter, we introduce a powerful tool - a neural network-based feature extraction method for generating novel features that compress the data. Data compression is achieved by having a single hidden layer feedforward neural network with fewer hidden units than input units.

Feature extraction is one kind of feature transformation which also includes feature construction, discretization, and subset selection (Liu and Motoda, 1998). It changes the data into a form that is simpler (or lower dimensions), easier to learn the target concept so learning algorithms can learn faster, and results in simpler hypotheses (e.g., smaller trees). Feature extraction takes in as input the original N features and generates as output M (where $M < N$) new features. The advantage of performing feature extraction is that it is a preprocessing process and works on data independently of a learning algorithm so that one can still use any learning algorithm one prefers, but have its performance boosted.

An often used feature transformation method is principle component analysis (PCA). The basic idea is to calculate the eigenvalues of the covariance matrix of the data and select those eigenvectors whose values are greater than a threshold (e.g., 0.9) to form a transformation matrix with M (the number of new features) rows and N (the number of original features) columns. The big disadvantage of PCA is that it does not take into account the class information, though it is available. By discarding the eigenvectors with eigenvalues less than the threshold, PCA cannot preserve the input information as much as we wish.

Three-layer feedforward neural networks have been widely applied as tools for supervised pattern classification. In this chapter, we discuss how they can be used for feature extraction. We obtain a network with the minimal number of hidden units and with connections only from the relevant input units to the hidden units by applying a network construction algorithm and a network pruning algorithm. The extracted features from the dataset take the hidden unit activation values of the patterns in the set. If desired, they can be fed as input into a classification method that build decision trees to obtain a set of decision rules such that a user may enjoy the high accuracy of the neural network and the explicitness of the classification rules of the decision tree.

We outline now the contents of this chapter. In Section 2, we describe how feature extraction via neural network is achieved. The two main ingredients in finding a suitable network, a neural network construction algorithm and a neural network pruning algorithm are outlined briefly. In Section 3, we illustrate the effectiveness of the proposed feature extraction method by applying it to two artificial datasets and the iris dataset. We present in details the new features extracted by the trained networks and show why they can be expected to improve the predictive accuracy of classifiers such as C4.5 (Quinlan, 1993) or k-nearest neighbor. Section 4 provides experimental evidence for the effectiveness of the proposed method in extracting useful features. The results of the experiments indicate that the extracted features are of lower dimensions compared to the original data and that they can be used as input for other learning algorithms to obtain robust classifiers with high predictive accuracy. Section 5 shows a decision tree built using the extracted features. This tree is much smaller than the tree generated using the original attributes of the data. Section 6 concludes this chapter.

A word about our notation. The vector x in n -dimensional space \mathbb{R}^n denotes a sample in the original attribute space. The vector $y \in \mathbb{R}^h$ denotes the extracted feature for pattern x . Normally, we expect h to be much smaller than n . Associated with each pattern x^i is a class target $t \in [0, 1]^c$. For a binary classification problem, we set c to 1. For a given network with a single hidden layer, w_ℓ^m denotes the weight of the connection from input unit ℓ to hidden unit m , while v_p^m denotes the weight of the connection from hidden unit m to output unit p . The extracted features for pattern x by a network with h hidden units are the hidden unit activation values for this pattern. It is denoted by an h -dimensional vector y which is computed as follows:

$$y = \delta(W^t x)$$

where W is the $n \times h$ weight matrix containing all the weights of input to hidden unit connections, δ is the hyperbolic tangent function which is applied to each of the h components of $W^t x$. For the matrix W , W^t is its transpose and $W_{i,j}$ is its element on the i -th row and j -th column. Two squashing functions are used: the sigmoid function $\sigma(\xi) = 1/(1 + e^{-\xi})$ for the activation of the output units and the hyperbolic tangent function $\delta(\xi) = (e^\xi - e^{-\xi})/(e^\xi + e^{-\xi})$ for the activation of the hidden units.

12.2 FEATURE EXTRACTION VIA NEURAL NETWORKS

Given a set S of k patterns in an n -dimensional space where each pattern belongs to a known class C , $C \in \{1, 2, \dots, c\}$, we obtain a three-layer feedforward neural network with n input units and c output units that maps the input patterns into their classes with a satisfactory accuracy rate. Pattern x^i of class C is assigned a c -dimensional class target t^i with $t_C^i = 1$ and $t_j^i = 0$ for all $j \neq C$. The algorithm for building the neural network is the maximum likelihood neural network construction algorithm that has been shown to be very effective (Setiono, 1995). The outline of this algorithm is as follows:

1. Set the initial number of hidden units h to 1.
2. Initialize the connection weights of the network randomly.
3. Find a set of weights such that it minimizes the error function:

$$\begin{aligned}
 F(w, v) = & - \sum_{i=1}^k \sum_{p=1}^c (t_p^i \log S_p^i + (1 - t_p^i) \log(1 - S_p^i)) + \\
 & \epsilon_1 \left(\sum_{m=1}^h \sum_{\ell=1}^n \frac{\beta(w_\ell^m)^2}{1 + \beta(w_\ell^m)^2} + \sum_{m=1}^h \sum_{p=1}^c \frac{\beta(v_p^m)^2}{1 + \beta(v_p^m)^2} \right) + \\
 & \epsilon_2 \left(\sum_{m=1}^h \sum_{\ell=1}^n (w_\ell^m)^2 + \sum_{m=1}^h \sum_{p=1}^c (v_p^m)^2 \right)
 \end{aligned} \tag{12.1}$$

where $\epsilon_1, \epsilon_2, \beta$ are positive parameters and S_p^i is the predicted output:

$$S_p^i = \sigma \left(\sum_{m=1}^h \delta((x^i)^t w^m) v_p^m \right)$$

4. If the network meets the required accuracy rate then stop.
5. Add one hidden unit to the hidden layer. The connection weights from the input units to the new hidden unit are set randomly, while the weights from this hidden units to the output units are set to 0.
6. Set $h = h + 1$ and go to step 3.

In the error function (12.1), we have appended a penalty term to the usual cross entropy error function as once the network has been constructed, irrelevant/redundant connections of the networks will be removed by pruning. In a problem with a large number of attributes, it is very likely that many of these input attributes are not relevant to the classification of the patterns. Adding the penalty term to the error function and minimizing the resulting augmented function during network construction help identify weights that can be eliminated. Connection weights from the irrelevant input units to the hidden units will have small magnitude and these connections can be eliminated. The benefits that can be gained by removing these connections are twofold. First, because of less overfitting, the resulting network will generally have better generalization ability, i.e., it can be expected to predict patterns not in the training dataset with higher accuracy. Second, it is easier to interpret the extracted features and ultimately, the classification process, when there are less connections from the input units the hidden units.

Our pruning algorithm removes network connections based on their magnitudes (Setiono, 1997b). After the connections are removed, the network is retrained. Pruning is terminated when the removal of a connection causes the

accuracy of the network to drop below the specified rate. When it is not possible to determine a suitable accuracy rate for network training and pruning, a subset of the training data may be set aside and used as the cross-validation set. The accuracy of the network on this set is then monitored to determine when to stop training and pruning.

The method for feature extraction using neural network can be outlined as follows:

1. Construct and prune a three-layer neural network.
2. Let W be a matrix with n rows and h columns of the network connection weights from the input units to the hidden units.
3. For each network connection from input unit ℓ to hidden unit m that has been pruned, we set $W_{\ell,m}$ to zero.
4. Given a pattern $x \in \mathbb{R}^n$, the constructed feature for this pattern is an h dimensional vector y :

$$y_j = \delta[(W^t x)_j], \quad j = 1, 2, \dots, h.$$

12.3 ILLUSTRATIVE EXAMPLES

Two artificial problems are selected to illustrate how features are extracted using neural networks and what kind of features are extracted for the problems. The problems are DNF9a and CNF9a defined in (Vilalta et al., 1997):

DNF9a: $x_2x_3 + \bar{x}_2x_3x_7 + x_2\bar{x}_3x_8 + x_2\bar{x}_7\bar{x}_8 + \bar{x}_3x_7x_8$.

CNF9a: $(x_4 + x_6)(\bar{x}_4 + \bar{x}_5 + x_7)$.

Both problems have 9 features x_1, \dots, x_9 . They are chosen because the extracted features are 2-dimensional, which make it possible to obtain a scattered plot of the transformed data. For each problem, we can recover the concept by constructing a network using 50 patterns for training, and 50 patterns for cross-validation. These patterns were randomly selected from the 512 unique binary vectors in \mathbb{R}^9 .

Example 1. DNF9a.

The network for this problem is shown in Figure 12.1. It has only 2 hidden units and only the inputs that correspond to the relevant attributes x_2, x_3, x_7 , and x_8 remain. We also show the plot of the hidden unit activations for all 512 patterns.

We can see from the scattered plot that patterns with very low hidden unit 1 activation values (close to -1) or with very low hidden unit 2 activation values (around -0.38) are not DNF9a. What are these patterns? The activation at hidden unit 1 is determined by x_2 and x_7 . Since both the connections from these inputs are positive, a very small activation value can be obtained only when $x_2 = x_7 = 0$. For the second hidden unit, the connection from x_7 is negative,

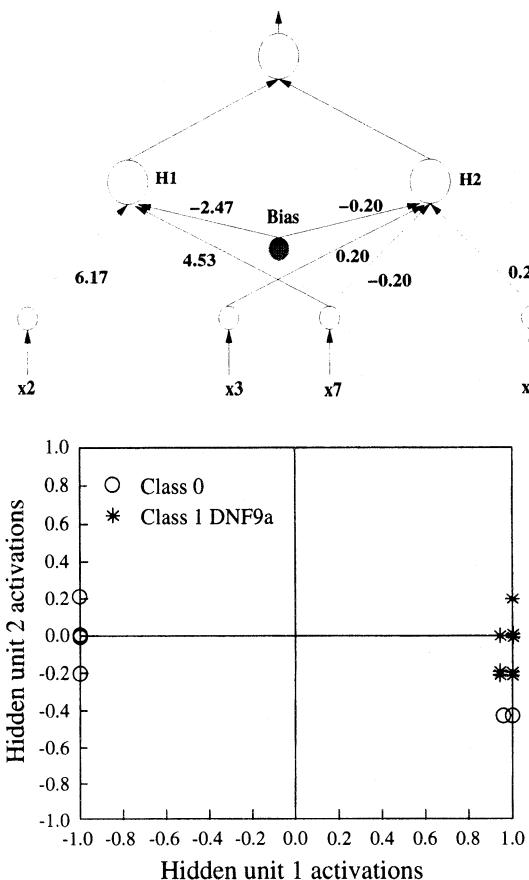


Figure 12.1 The neural network for DNF9a (Top). The transformed patterns in 2 dimensional feature space (Bottom).

while the connections from x_3 and x_8 are positive. Here, we can conclude that the minimum activation value can be obtained for a pattern with $x_3 = 0, x_7 = 1$ and $x_8 = 0$. Hence, for this problem, we can recover the original concept by inspecting the 2 dimensional transformed features of the patterns. We conclude that a pattern is DNF9a if it satisfies the condition $\neg(\bar{x}_2\bar{x}_7 + \bar{x}_3x_7\bar{x}_8)$.

Example 2. CNF9a.

We also constructed a network using 50 randomly selected patterns for training and another 50 patterns for cross-validation. Figure 12.2 shows the pruned network and the scattered plot of the hidden unit activation values. As in the previous example, these values make it easy for us to distinguish between CNF9a patterns and those that are not. From the plot, it is evident that a

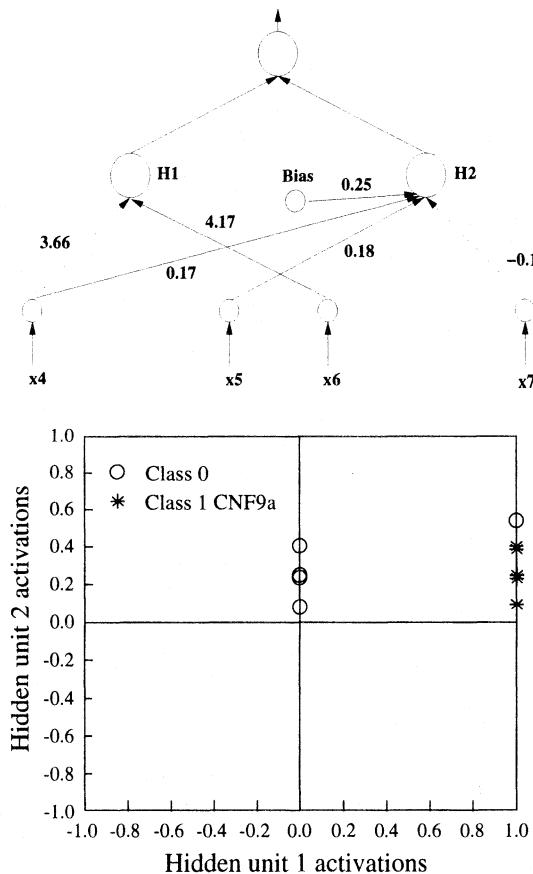


Figure 12.2 The neural network for CNF9a (Top). The transformed patterns in 2 dimensional feature space (Bottom).

pattern is not CNF9a if its hidden unit 1 activation value is 0 or if its hidden unit 2 activation value is largest. An activation value of a pattern at hidden unit 1 is 0 if and only if $x_4 = x_6 = 0$, while an activation at hidden unit 2 is largest if and only if $x_4 = 1, x_5 = 1$ and $x_7 = 0$. We conclude that a pattern is CNF9a if this condition is satisfied: $\neg(\bar{x}_4\bar{x}_6 + x_4x_5\bar{x}_7)$.

Example 3. Iris.

This is a classic pattern classification problem introduced by R.A. Fisher (Fisher, 1936). The dataset contains 50 patterns of each of the 3 classes: Iris setosa, Iris versicolor, and Iris virginica. The attributes are sepal-length (A_1), sepal-width (A_2), petal-length (A_3), and petal-width (A_4). This example is a showcase that the feature extraction method proposed here can also be applied to continuous data. We want to investigate what features can be extracted by a simple net-

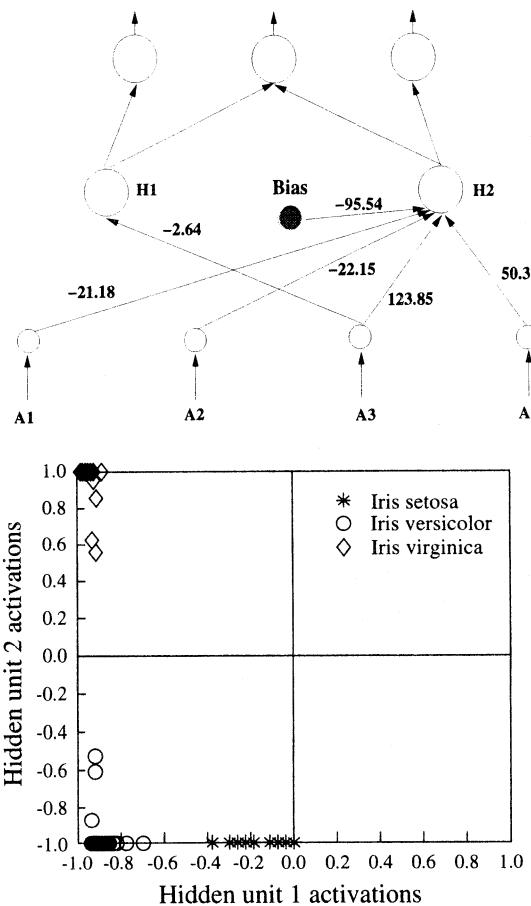


Figure 12.3 The neural network for Iris classification (Top). The transformed patterns in 2 dimensional feature space (Bottom).

work with high accuracy. Hence, the neural network shown in Figure 12.3 was constructed using all the data. Before training was started, all input attribute values were normalized such that their values were in the interval $[0, 1]$. It correctly classifies all but one of the 150 patterns in the dataset. The plot of the hidden activations of the patterns is also shown. This plot shows how the original patterns are gathered together in terms of their class values in the 2 dimensional transformed space. Patterns of Iris virginica form a cluster at the upper left hand corner, those of Iris versicolor at the lower left hand corner, while those of Iris setosa having hidden unit 1 activation values between -0.40 and 0 and hidden unit 2 activation of 0 form the third cluster.

Table 12.1 The dimensionality of features extracted and accuracy of the neural networks.

Problem	Accuracy (%)	Extracted features		
		Min.	Max.	Average
DNF9a	100.0	2	2	2.0
DNF9b	100.0	2	2	2.0
DNF12a	99.0	3	4	3.4
DNF12b	99.6	3	5	3.7
CNF9a	100.0	2	2	2.0
CNF9b	100.0	2	3	2.8
CNF12a	100.0	3	3	3.0
CNF12b	99.2	5	7	5.9
MAJ9a	100.0	1	1	1.0
MAJ9b	100.0	1	1	1.0
MAJ12a	100.0	1	1	1.0
MAJ12b	100.0	1	1	1.0
PAR9a	100.0	2	2	2.0
PAR9b	98.6	3	4	3.2
PAR12a	100.0	3	4	3.5
PAR12b	99.3	4	7	4.5
MUX9	99.6	3	5	4.0
MUX12	99.5	7	8	7.2

12.4 EMPIRICAL STUDY AND ANALYSIS

In addition to the three domains used for illustration purpose in the previous section, we have also carried out experiments on the other artificial problems given in (Vilalta et al., 1997) (see Appendix). We report in this section our experimental results of these problems.

For each problem, 10-fold cross-validation runs were conducted. For a problem with n binary features, all 2^n patterns were generated. In each of the 10 runs, 80 % of the patterns were used for neural network training, 10 % for cross-validation, and the remaining 10 % for testing. The patterns in the cross-validation set were used to determine when neural network construction and pruning should be terminated. The average accuracy on the testing set and the average number of hidden units of the 10 networks are computed. The results are summarized in Table 12.1. The table also shows the minimum and maximum numbers of hidden units in the 10 networks. These figures correspond to the smallest and largest dimensionality of the features extracted.

The dimensionality of the extracted features ranges from 1 to 8. We make the following observations on the figures in Table 12.1:

- The networks extracted 2 dimensional features for DNF9a and DNF9b. For the two larger DNF12a and DNF12b problems, the dimensions of the extracted features are as low as 3. A closer inspection of all 20 pruned networks trained on the two DNF12 datasets reveals that networks with 4 or 5 hidden units predict with 100 % accuracy. A new feature is defined by a combination of up to 6 original binary features.
- For CNF9a and CNF12a, the numbers of hidden units are as expected, they are always 2 and 3, respectively. For CNF9b, 2 of the 10 pruned networks have only 2 hidden units. For CNF12b, networks with as few as 5 hidden units have 100 % accuracy.
- The patterns of the majority problems (MAJ9a, MAJ9b, MAJ12a, MAJ12b) are linearly separable. This is obvious from their definitions. All networks trained on these patterns have only one hidden unit. However, a decision tree algorithm with a single-feature test at every node such as C4.5 cannot be expected to perform well (Vilalta et al., 1997). It would generate trees with many nodes and poor generalization ability. If the decision tree method is applied to the extracted features instead of to the original features of any of the four problems, a tree with only 2 leave nodes will be generated. Each leave node contains patterns of only 1 class.
- The sufficient numbers of hidden units to correctly classify all patterns of PAR9a and PAR12a are 2 and 3, respectively as expected (Setiono, 1997a). PAR9b requires 4 hidden units, while PAR12b 5 hidden units. Those networks with less than the required hidden units do not achieve 100 % predictive accuracy.
- One of the 10 networks trained on MUX9 dataset has 3 hidden units, this is the only one whose predictive accuracy is not 100 %. For MUX12, only network with 8 hidden units can correctly classify all 1024 patterns. Because of the randomness in the division of the patterns into training, cross-validating and testing sets, sometimes the accuracy of a constructed network can be as high as 100 % on the cross-validating set, while in fact the correct concept has not been learned. This is the reason why sometimes the construction algorithm stops prematurely with a network that does not give perfect predictive accuracy.

12.5 CNF9A REVISITED FOR DECISION TREE INDUCTION

We revisit one of the illustrated examples CNF9a and build decision trees using original and extracted features, shown in Figures 12.4 and 12.5. From left to right, branches in Figure 12.4 contain 128, 32, 32, 64, 128, 32, 32, and 64

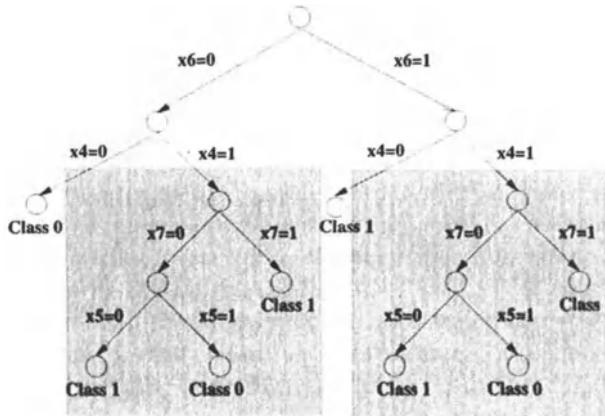


Figure 12.4 A decision tree built by C4.5 using original features.

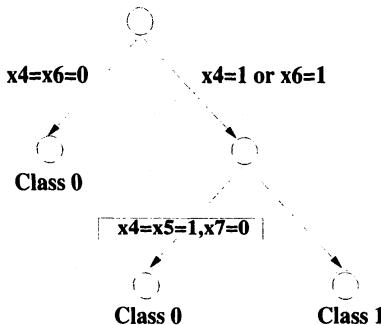


Figure 12.5 A decision tree built using extracted features.

patterns, respectively; branches in Figure 12.5 have 128, 64, and 320 patterns, respectively. There are 192 patterns that are not satisfying CNF9a in both trees. We can observe the replication problem in the tree (shaded branches) in Figure 12.4 which has many branches and four leaf nodes contain as few as 32 patterns and their parent node has only 64 patterns, while for the tree in Figure 12.5, there are only three branches, the least number of patterns in a leaf node is 64, which is not big, but its parent node where a decision test is done has 384 patterns. What's even better is that there is no trace of the replication problem. In other words, every decision test in the tree of Figure 12.5 is statistically more reliable.

12.6 SUMMARY

We have introduced a feature extraction algorithm using neural networks. The essence of the algorithm is to train a highly accurate network with a necessary number of hidden units, to further prune it without sacrificing its accuracy, and to use the remaining hidden units as extracted features. Many issues are considered, such as avoiding overfitting, better estimating error rates, and pruning connections. The empirical study demonstrates that the algorithm can handle both binary and continuous data. It can effectively handle the difficult problems such as DNF, CNF, MAJ, PAR, and MUX. In all cases, the number of extracted features is less than the number of original features. As shown in illustrative examples, the patterns of extracted features are gathered together in terms of their class values and the classification problems using the extracted features are much simpler. If desired, it is clear that a simple decision tree can be built on the extracted features for the three examples, although each node is more complicated now if it is rewritten in original features. In the meantime, we should also expect high accuracy due to the clear boundaries of patterns of different classes in the new space.

Appendix: Definitions for artificial problems

Refer to Table 12.A.1 for definitions for artificial problems used in (Vilalta et al., 1997).

References

- Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software.
- Brodley, C. and Utgoff, P. (1995). Multivariate decision trees. *Machine Learning*, 19:45–77.
- Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283.
- Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann Publishers, Inc.
- Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(2):179–188.
- Friedman, J., Kohavi, R., and Yun, Y. (1996). Lazy decision trees. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 717–724.
- Kohavi, R. (1994). Bottom-up induction of oblivious read-once decision graphs. In *Machine Learning: ECML-97*, pages 154–169. Springer-Verlag.
- Liu, H. and Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers.

Table 12.A.1 Definitions for artificial problems

Let	$X = (x_1, x_2, \dots, x_n)$, address(x_1, x_2, \dots, x_m) = $2^0 x_1 + 2^1 x_2 + \dots + 2^{m-1} x_m$
Definitions:	
DNF9a	$x_2 x_3 + \bar{x}_2 x_3 x_7 + x_2 \bar{x}_3 x_8 + x_2 \bar{x}_7 \bar{x}_8 + \bar{x}_3 x_7 x_8$
DNF9b	$x_1 x_2 x_3 + \bar{x}_1 \bar{x}_2 + x_7 x_8 x_9 + \bar{x}_7 \bar{x}_9$
DNF12a	$x_1 x_2 x_9 x_{12} + x_1 x_2 \bar{x}_9 x_{11} + \bar{x}_1 x_2 x_5 x_9 + \bar{x}_1 \bar{x}_2 x_8 \bar{x}_9$
DNF12b	$x_1 x_2 \bar{x}_7 \bar{x}_8 + x_1 x_2 x_{11} x_{12} + x_1 \bar{x}_2 \bar{x}_{11} x_{12} +$ $\bar{x}_1 x_2 x_{11} \bar{x}_{12} + \bar{x}_1 \bar{x}_2 \bar{x}_{11} \bar{x}_{12} + x_7 x_8 x_{11} x_{12}$
CNF9a	$(x_4 + x_6)(\bar{x}_4 + \bar{x}_5 + x_7)$
CNF9b	$(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_5)(x_1 + x_2 + \bar{x}_3)$
CNF12a	$(x_8 + x_9)(x_5 + x_6 + x_7)(x_2 + \bar{x}_6 + \bar{x}_8)$
CNF12b	$(x_1 + x_2 + x_6)(\bar{x}_1 + \bar{x}_2 + x_7)(x_9 + x_{10} + x_{12})$ $(\bar{x}_6 + \bar{x}_7 + x_8)(x_9 + \bar{x}_{10} + \bar{x}_{11})(\bar{x}_7 + \bar{x}_8 + \bar{x}_9)$
MAJ9a	$\{X (\sum_{i=1}^9 x_i) \geq 3\}$
MAJ9b	$\{X (\sum_{i=1}^9 x_i) \geq 6\}$
MAJ12a	$\{X (\sum_{i=1}^{12} x_i) \geq 4\}$
MAJ12b	$\{X (\sum_{i=1}^{12} x_i) \geq 8\}$
PAR9a	$\{X (\sum_{i=1}^3 x_i) \bmod 2 > 0\}$
PAR9b	$\{X (\sum_{i=1}^6 x_i) \bmod 2 > 0\}$
PAR12a	$\{X (\sum_{i=1}^4 x_i) \bmod 2 > 0\}$
PAR12b	$\{X (\sum_{i=1}^8 x_i) \bmod 2 > 0\}$
MUX9	$\{X x_{(\text{address}(x_1, x_2)+3)} = 1\}$
MUX12	$\{X x_{(\text{address}(x_1, x_2, x_3)+4)} = 1\}$

- Liu, H. and Setiono, R. (1995). Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*.
- Matheus, C. and Rendell, L. (1989). Constructive induction on decision trees. In *Proceedings of IJCAI*, pages 645–650.
- Murthy, S., Kasif, S., Salzberg, S., and Beigel, R. (1993). OC1: Randomized induction of oblique decision trees. In *Proceedings of AAAI Conference (AAAI'93)*, pages 322–327. AAAI Press / The MIT Press.
- Oliveira, A. (1995). Inferring reduced ordered decision graphs of minimum description length. In *Proceedings of the 12th International Conference on Machine Learning*, pages 421–429.
- Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99.
- Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

- Quinlan, J. (1987). Generating production rules from decision trees. In *Proceedings of 10th International Joint Conference on Artificial Intelligence*, pages 304–307. Morgan Kaufmann.
- Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
- Rendell, L. and Seshu, R. (1990). Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence*, 6:247–270.
- Setiono, R. (1995). A neural network construction algorithm which maximizes the likelihood function. *Connection Science*, 7(2):147–166.
- Setiono, R. (1997a). On the solution of the parity problem by a single hidden layer feedforward neural network. *Neurocomputing*, 6(3):225–235.
- Setiono, R. (1997b). A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204.
- Vilalta, R., Blix, G., and Rendell, L. (1997). Global data analysis and the fragmentation problem in decision tree induction. In van Someren, M. and Widmer, G., editors, *Machine Learning: ECML-97*, pages 312–326. Springer-Verlag.
- Wnek, J. and Michalski, R. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14.

13 USING LATTICE-BASED FRAMEWORK AS A TOOL FOR FEATURE EXTRACTION

Engelbert Mephu Nguifo and Patrick Njiwoua

C.R.I.L - Université d'Artois
Rue de l'Université SP-16
62307 Lens Cedex, FRANCE
{mephu|njiwoua}@crl.univ-artois.fr

Abstract: Preprocessing data is a means for improving the efficiency of concept learning systems. Feature transformation (FT) increasingly becomes a central problem in machine learning. Different FT approaches were intensively studied this last few years. These approaches wish to give a learner only those attributes that are relevant to the target concept. This chapter presents a process that extracts a set of new numerical features from the original set of boolean features through the use of an empirical mapping function. This mapping is based on the use of an entropy function to learn knowledge over the Galois semi-lattice construction of the initial set of objects. One advantage here is the reduction of the effect of possible irrelevant features. This process allows to design an Instance-based Learning system, IGLUE, which uses the Mahanalobis measure over the numerical features. The IGLUE system is compared with a well-known tree-based system, C4.5, an instance-based learner K*, and a lattice-based system LEGAL, in terms of classification accuracy and running time on some real-world and artificial data sets.

13.1 INTRODUCTION

Learning to classify objects is one of the most studied areas in machine learning (ML). Many ML systems have been developed in order to achieve this purpose. The most well-known ML system is ID3 (Quinlan, 1986), which is a decision tree-based system. It plays an increasingly important role in ML. The last decade has witnessed several variants of this system, among which C4.5 (Quin-

lan, 1993). However the time required to learn by these ML systems grows considerably with the number of features.

Meanwhile important works was done in order to illustrate the efficiency of preprocessing data in the area of statistics, pattern recognition as well as ML. ML systems use data analysis methods towards the search space to outperform both accuracy and complexity of their learning algorithm. These systems generally focus on selection or transformation of original features. Encouraging results of classification by feature projections (Akkus and Güvenir, 1996) and classification with overlapping feature intervals were obtained. Koller and Sahami (Koller and Sahami, 1996) give an approximate algorithm for optimal feature selection, which is untractable in practice.

Feature transformation (FT) relies also on the extraction of new features through the use of a mapping function. In a recent work, Zheng (Zheng, 1996) describes the effects of different types of new attributes (conjunctive, disjunctive, M-of-N, and X-of-N representations). These new attribute remains binary or nominal features since continuous-valued attributes are transformed into binary or nominal attributes by discretization.

In this chapter, we develop a novel constructive method for feature extraction that uses ML techniques to generate new relevant features. Our approach translates initial binary attributes into continuous-valued ones. This translation takes into account only relevant attributes. The feature extraction is achieved for different purposes:

1. It is necessary to reduce the effect of irrelevant attributes, by focusing our attention on the way objects are defined in the whole initial context. That is, the definition of an object should be analyzed not separately, but by considering other examples descriptions.
2. In the literature, there are different kinds of discretization methods that allow to transform continuous-valued features into binary or nominal ones. We propose a method to achieve the opposite process.
3. Binary features are often not well appropriated for instance-based learning (IBL), since the choice of a good similarity measure is more difficult with boolean features than with numerical features.

Many Instance-based learners have been developed such as the system K* (Cleary and Trigg, 1995). They have demonstrated excellent classification accuracy over a large range of domains. Instance-based learners classify an instance by comparing it to a set of pre-classified examples. There is a fundamental assumption that similar instances have similar classification. The main difficulty consist of choosing a distance function to define *similar instance*, and a classification function to define *similar classification*. Another difficulty arises from the lack of an integrated theoretical background. To reduce such limitations, IBL systems use a preprocess mechanism on initial data in order to select relevant features. This chapter presents a new algorithm for feature extraction that has been integrated in an ML technique to design the IBL system called IGLUE. The feature extraction algorithm consists of two steps.

First, it builds a join-semi-lattice of the initial context of objects and binary features. To do this, it uses the entropy function to select relevant nodes during the top-down lattice construction. To reduce lattice complexity, only relevant nodes at the top level are taken into account.

Second, all the initial examples are redescribed by the way they are concerned with each relevant node of the semi-lattice. The redescription of examples allows to suppress irrelevant features which are sometimes introduced in the initial description of examples. Irrelevant features are those which do not appear inside the semi-lattice nodes. This redescription transforms only relevant initial binary features into new numerical attributes, depending on how examples could appear or not in the relevant nodes. More precisely, we generate a new continuous-valued feature for each remaining original feature. The new feature value for an example is the number of relevant nodes containing both the example and the corresponding binary feature.

IGLUE is based on the nearest neighbor principle to classify unseen examples. The system applies a similarity measure, which is the Mahalanobis distance, between the redescription of unseen example and the redescription of examples of the training set in order to achieve the classification.

Experimental comparisons reported in this chapter show the effectiveness of IGLUE. Its computational complexity is lower than that of a lattice-based system LEGAL (Mephu Nguifo, 1994) while its predictive accuracy is as good as that of LEGAL on different well-known datasets. Results of lattice-based systems are better than that of the two other systems: decision tree-based C4.5 and instance-based K^* . However their running time requirements are more higher than that of these two other systems.

In Section 2, we review the preliminaries of concept lattices, and overviews ML related to concept lattices. Section 3 describes the feature extraction algorithm and section 4 presents the IGLUE system. Finally section 5 shows experimental comparison on different well-known datasets.

13.2 GALOIS LATTICE

Lattice theory gives rise to a new research domain termed *Formal Concept Analysis* (Wille, 1982) where concept lattice can be understood as a basic answer to two fundamental questions concerning an initial context, namely the question of an appropriate classification of the objects and the question about the dependencies between attributes.

Recently, Ganascia (Ganascia, 1993) introduced a formal model to describe Top-Down Induction System (TDIS), based on the use of the Galois connection. It is clearly shown that concept lattice is a general framework in which other learning TDIS strategies can be expressed.

LEGAL (Mephu Nguifo, 1994) is an empirical inductive learning system which builds only a join-semi-lattice as learned knowledge. It has been successfully applied on real problems coming from different domains such as biology (Vignal et al., 1996). Carpineto and Romano (Carpinetto and Romano, 1993) implemented an incremental algorithm to design a lattice conceptual clustering

system applied to browsing retrieval. Sahami (Sahami, 1995) described the learning system RULEARNER, which also builds all the concept lattice, and then uses heuristics to select pertinent concepts inside the lattice. Among all these lattice-based systems, LEGAL has the weak theoretical complexity since it uses two learning parameters to generate only a join-semi lattice.

Galois lattice is a mathematical framework that allows to build embedded classes from a set of objects. An overview of concept lattice notions follows.

- A *context* C is a triple (O, A, I) , where O , A are respectively sets of examples and attributes, and $I \subseteq O \times A$ is a binary relation between O and A . We define the two mappings $f : O \rightarrow A$ and $g : A \rightarrow O$ as follows : Given $o \in O$ and $a \in A$, $f(o) = \{a \in A; (o, a) \in I\}$ is the set of all attributes verified by o , $g(a) = \{o \in O; (o, a) \in I\}$ is the set of all objects that have attribute a .

f and g are respectively extended to $O_1 \subseteq O$ and $A_1 \subseteq A$.

- The pair (f, g) forms a *Galois connection* between the two power sets over O and A .
- The pair (O_1, A_1) is a *concept* if $f(O_1) = A_1$ and $g(A_1) = O_1$.
- O_1 (resp. A_1) is called the *extent* (resp. the *intent*) of the concept (O_1, A_1) .

To avoid ambiguity, a lattice concept is termed a *node*.

- An *order relation* (denoted \leq) is defined over the set of all nodes. For two nodes (O_1, A_1) and (O_2, A_2) , $(O_1, A_1) \leq (O_2, A_2) \iff O_1 \subseteq O_2$ and $A_1 \supseteq A_2$.
- With \leq , the system of all nodes of a context has the mathematical structure of a *complete lattice* (i.e. two nodes of the lattice have a common super-node and a common sub-node) and is called the *Galois lattice* $L(C)$ of C (see an example in Figure 13.1).

$O \setminus A$	a	b	c	d	e	Class
1	1	1		1		+
2			1	1		1
3	1	1	1	1		+
4			1		1	-
5	1			1		-

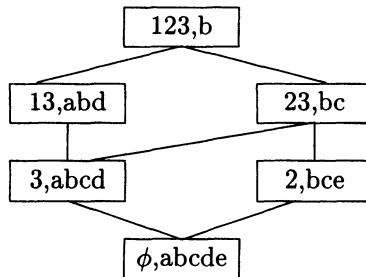


Figure 13.1 A matrix context and the complete Galois lattice of the positive context.

Given the binary matrix of the relation I between the set O of positive and negative examples ($O = O^+ \cup O^-$), and the set A of attributes, Figure 13.1 shows a matrix context $C = (O, A, I)$ and the complete Galois lattice of the context (O^+, A, I) . In this figure, a rectangle represents a concept node and lines between rectangles denote the top-down order relation. Refer to Figure 13.2 for the lattice's construction algorithm.

We define a *regularity* as a conjunction of attributes. For example, in Figure 13.1, $(\{1, 3\}, \{a, b, d\})$ is a node, and the associated regularity is $a \wedge b \wedge d$ or abd .

13.3 FEATURE EXTRACTION ALGORITHM

This section describes the feature extraction algorithm which consists of two steps. The first step builds the join-semi-lattice of the initial context, and selects relevant nodes. The second step generates new continuous-valued attributes and redescribes the training set of examples with these new features.

13.3.1 Join-semi-lattice Algorithm

The algorithm used here to generate the join-semi-lattice is somehow identical to that of LEGAL (Mephu Nguifo, 1994). However they differ by the heuristics used. While LEGAL selects relevant nodes by using two user-specified learning parameters which are respectively the minimum and maximum required percentage of positive and negative instances inside a node, our method uses the entropy function to select relevant nodes at the h-first levels of the join-semi-lattice. This function has one advantage over the empirical process of LEGAL, since it allows to select all significant nodes even if they contain few training instances. These instances can be lonely instances which have any similarity with other training instances.

```

Join-algo( O,A,I,h,Seuil1):
Begin
   $F \leftarrow (O^+, \{\})$ ;  $L \leftarrow \{\}$ ;  $P \leftarrow \{\}$ ;
  While  $\exists(O_k, A_k) \in F$ 
    Remove  $(O_k, A_k)$  from  $F$ ;
    Construct  $C_k$ , the set of sub-nodes of  $(O_k, A_k)$ ; (see Figure
    13.3).
    For each node  $(O_{k_i}, A_{k_i}) \in C_k$ 
      If  $Level(O_{k_i}, A_{k_i}) < h$  Then
        If  $Entrop(O_{k_i} \cup (g(A_{k_i}) \cap O^-), A_{k_i}) \leq Seuil1$  Then
          If  $(O_{k_i}, A_{k_i}) \notin L$  Then
             $P \leftarrow P \cup A_{k_i}$ ;
            Insert( $O_{k_i}, A_{k_i}, L$ );
          Else
            add an edge from  $(O_k, A_k)$  to  $(O_{k_i}, A_{k_i})$  in  $L$ ;
          Endif
        Endif
      Endif
    Endfor
  EndWhile
  Return( $P, L$ )
End
```

Figure 13.2 Pseudo-code of the join construction algorithm

```

Construction( A, Ok, Ak):
Begin
  S ← A - Ak, Ck ← {}
  For each attribute (column) a in S
    Maxi ← TRUE; Egal ← {a}
    For each column a' ≠ a
      If g(a) ∩ O+ ⊆ g(a') ∩ O+ Then
        Maxi ← FALSE; Stop
        If g(a) ∩ O+ = g(a') ∩ O+ Then Egal ← Egal ∪ {a'}
      Endif
    Endfor
    If Maxi = TRUE Then
      Ok,i ← Ok ∩ g(Egal)
      Ck ← Ck ∪ (Ok,i, Ak ∪ Egal )
    Endif
  Endfor
  Return (Ck)
End

```

Figure 13.3 Pseudo-code of the sub-nodes construction subroutine.

The join-semi-lattice algorithm (see Figure 13.2) uses two procedures (*Construct* and *Insert*) to first construct all sub-nodes of a given node and second insert a new one in the lattice.

The procedure *Level*(O₁, A₁) gives the level of a node inside the lattice. The maximum level, *h*, of the join-semi-lattice is an input parameter which can be specified by the user.

The procedure *Entrop*(O₁, A₁) consists in using the entropy function to evaluate the relevance of the node (O₁, A₁). Let p₁ = |O₁ ∩ O⁺| and n₁ = |O₁ ∩ O⁻|:

$$\text{Entrop}(O_1, A_1) = -\frac{p_1}{p_1 + n_1} \cdot \log_2\left(\frac{p_1}{p_1 + n_1}\right) - \frac{n_1}{p_1 + n_1} \cdot \log_2\left(\frac{n_1}{p_1 + n_1}\right).$$

The user could also specified one threshold (*Seuil1*) for the entropy function. Very often the ML system searches by iteration for the best value. Therefore a definition for valid and relevant node is derived:

- a node (O₁, A₁) is *relevant* if *Entrop*(O₁, A₁) ≤ *Seuil1*; its intent (and the regularity obtained from it) is also relevant.
- otherwise (O₁, A₁) is *valid* and the intent is valid too.

The upper bound for the size of the join-semi-lattice is $\sum_{i=1}^h C_{\min(n,m)}^i$, and the upper bound of the system's time complexity is $\sum_{i=1}^h C_{\min(n,m)}^i \cdot n \cdot |I|$; If $i \leq j$, $C_j^i = \frac{j!}{i!(j-i)!}$.

13.3.2 Feature Generation

Here we describe the process to generate new continuous-valued features. We use the set of relevant nodes of the join-semi-lattice for this purpose. Let A^* be the set of attributes of A which appears in at least one relevant node. If an attribute a_i never appears in one of the lattice nodes, then it is not relevant. Such attribute would not be used again by the system when it generates new features.

A feature a_k is more relevant than a_h if the number of nodes in which a_k appears is greater than the number of nodes containing a_h .

All training examples are redescribed by the way they interact with the set of relevant nodes. We associate to each attribute a_k of A^* , a new feature, d_k which is defined by the relation J .

Let D be the set of those new features, d_1, \dots, d_m , where m is the number of attributes of A^* .

Let $P_i \subseteq P$ be the set of relevant regularities which hold for the example o_i .

For an example o_i and a new feature $d_k \in D$, we define a new relation J between O and D , where $J(o_i, d_k)$ is the number of appearances of attribute a_k in all regularities $r \in P_i$, and thus, $0 \leq J(o_i, d_k) \leq |P|$. This relation is then extended to examples of the test set or unseen examples.

Each feature d_k is a quantitative variable and has a correspondence a_k in A^* . The number of new features is less equal to the number of original binary features ($|A^*| \leq |A|$, thus $|D| \leq |A|$). It is important to have a strong dependency between new features and the set of built regularities due to their future use for learning purpose. They are built by the scheme explained in Figure 13.4.

In the following, d_{ik} will refer to *the value of the feature d_k for the example o_i* .

Redescription(O, D):

Begin

$d_{ik} = 0$ for all examples o_i and all features $d_k \in D$

For each example o_i

 Find P_i , the set of relevant regularities that holds for o_i

For each regularity $r \in P_i$

For each attribute a_k of the regularity r

 Add 1 to d_{ik}

EndFor

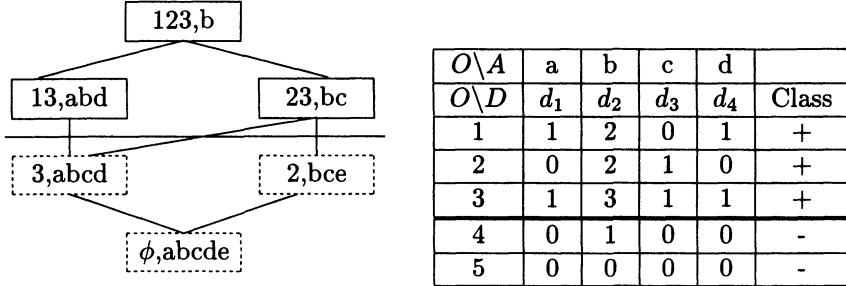
EndFor

EndFor

End

Figure 13.4 Instances redescription algorithm.

Figure 13.5 shows a join semi-lattice and its context redescription. In this context, the continuous-valued features are obtained after redescribing the pre-

**Figure 13.5** A join semi-lattice and its context redescription.

vious example context (see Figure 13.1). Solid rectangles denote relevant nodes while dashed rectangles are not generated by IGLUE since they are irrelevant. As the initial feature e doesn't appear in the built semi-lattice, e is an irrelevant attribute. The new generated features d_1, d_2, d_3 and d_4 respectively correspond to original features a, b, c and d .

13.4 THE IGLUE SYSTEM

Methods of numerical taxonomy express similarity as a numerical value, and they appear to implicitly assume that objects can be represented naturally in terms of continuously valued variables. The similarity between two objects is the value of a numeric function applied to the descriptions of the two objects. Here we are interested in classifying an unseen example.

Literature reports extensive studies on nearest neighbor (NN) algorithms for learning from examples. These methods often work as well as other sophisticated ML techniques (Salzberg, 1991). Among different similarity measures proposed in the literature (Salzberg, 1991), the most common metric is the Euclidean distance metric. However we choose to implement the Mahalanobis distance (Quinqueton and Sallatin, 1982):

$$s(o_i, o_j) = \sum_{1 \leq k \leq m} \frac{|d_{ik} - d_{jk}|}{\sqrt{d_{ik} + d_{jk}}}.$$

For all objects o_x belonging to the training set O or not, and for a given similarity threshold δ , we define $S(o_x)$ as the set of training objects that are most similar to o_x : $S(o_x) = S^+(o_x) + S^-(o_x)$, where $S^+(o_x)$ (resp. $S^-(o_x)$) is the set of similar training positive (resp. negative) examples. The threshold δ is determined by the system. $S(o_x) = \{o_j \in O | s(o_x, o_j) \leq \delta\}$.

- An unseen example o_x is a *positive instance* if its most similar training example is a positive one.
- Otherwise o_x is a *negative instance*, as its most similar training example is a negative one.

Table 13.1 Distance between instances.

<i>Objects</i> \ <i>Objects</i>	1	2	3	4	5
1	0.00	-	-	-	-
2	3.00	0.00	-	-	-
3	1.45	2.45	0.00	-	-
4	+M	+M	4.00	0.00	-
5	+M	+M	4.73	+M	0.00

Table 13.1 gives an example of distance values obtained when applying the Mahanalobis measure between instances redescribed in Figure 13.5. +M stands for an infinite distance between two instances. In this example, setting the δ to 3 will provide the correct label to each instance.

13.5 EXPERIMENTAL RESULTS

In this section, we present preliminary experiments designed to test our feature extraction algorithm over the new system IGLUE. We report a practical comparison in terms of complexity and prediction accuracy between LEGAL, IGLUE, C4.5 (unpruned version) and K^* . The time and space performances comparison have been carried out using the domain that was used to test these systems. We assembled some data sets (see Table 13.2) from the classification-learning problems available in the UC/Irvine Repository (Merz and Murphy, 1996). The discretization process applied here, for LEGAL and IGLUE, consists in creating a new attribute for each original feature value. C4.5 and K^* are tested with their default parameters values within the Waikato Environment for Knowledge Analysis, WEKA (Garner, 1995).

Table 13.2 A brief description of the used data sets.

<i>Data</i>	<i>Class</i>	<i>Attrib.</i>		<i>Examples</i>	
		<i>Orig.</i>	<i>Bin.</i>	<i>Train</i>	<i>Test</i>
Monks1	2	8	16	124	432
Monks2	2	8	16	169	432
Monks3	2	8	16	122	432
Breast-C (W)	2	11	91	699	CV
S. Soybean	4	36	136	47	CV
Votes	2	17	17	435	CV

13.5.1 Methodology

For each data set of the Monks problems, training and testing sets are given. We apply a five cross-validation method on the three other data sets to measure the accuracy of the three systems. Each time, we randomly select a subset to be used for the training part and the remaining data was used for test. In the case of data set with more than two classes (in this case small Soybean), each class was recognized against the others. The size of the partition used for training was less than 60% of the total number of examples. The method to select system parameters follows: a batch process has to choose for each system the parameters values which give quite good results on the learning set. These values are then used for test. The different system codes are executed on a Pentium 166 machine with $32M_o$ of RAM.

13.5.2 Results and Discussion

The Figure below and Table 13.3 present the percentage of relevant attributes and learning accuracy results obtained with IGLUE for different lattice's level.

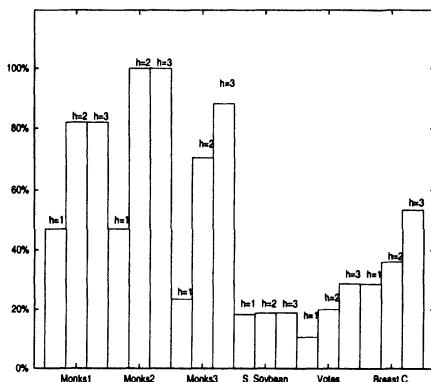


Figure 13.6a Used attributes.

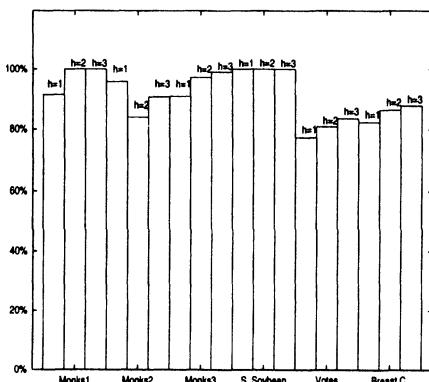


Figure 13.6b Learning accuracy.

Our first remark is that the number of relevant attributes and the learning accuracy increase with lattice's level except in the case of the Monks2 problem.

The increase of the number of relevant attributes is significant between the first and the second levels and weak after the second level. The increase of the learning accuracy is slight after the first level. So that, the default value of the lattice's level could be set to 1, which considerably reduces the system learning time and space complexities (see Table 13.4). This remark is generally confirmed by the test accuracy results (see Table 13.5).

Table 13.4 presents the required time to learn and test a concept on different data sets. Rows indicate which data set was used. For the considered data sets, lattice-based systems are slower than C4.5 and K^* , which illustrated the computational disadvantage of lattice algorithm.

Table 13.3 Percentage of used attributes (F) and learning accuracy (A) with IGLUE.

Data	$h=1$		$h=2$		$h=3$	
	F	A	F	A	F	A
Monks1	47.0	91.4	82.0	100.0	82.0	100.0
Monks2	47.0	95.8	100.0	84.0	100.0	90.7
Monks3	23.5	90.9	70.5	97.2	88.2	99.0
S.Soybean	18.3	100.0	18.8	100.0	18.9	100.0
Votes	10.6	77.4	20.0	80.9	28.7	83.6
Breast C.	28.9	82.4	36.1	86.4	53.4	87.8

However IGLUE is really faster than LEGAL on all problems, although it is based on a more complex approach that combines different learning strategies. This is due to the level threshold used by IGLUE when building the semi-lattice. When the value of this level increases, the time of IGLUE slightly increases but not exponentially as the theoretical complexity may indicate. This comes from the fact that the lattice construction also depends on the content of the binary table.

Table 13.4 Time spent (in seconds) on different data sets.

Data	LEGAL	IGLUE			C4.5	K*
		$h=1$	$h=2$	$h=3$		
Monks1	294.55	44.21	165.08	237.51	30.0	4.5
Monks2	4796.25	7.81	41.85	159.83	39.0	4.5
Monks3	4618.60	25.20	154.41	313.96	21.0	4.5
B. Cancer	29852.93	3169.23	4075.32	4110.28	141.7	74.2
S. soybean	136.56	96.17	107.53	131.55	35.4	31.1
Votes	10927.32	791.55	1371.80	2285.30	43.4	38.7

Results of classification accuracy obtained on the previous data sets are summarized in Table 13.5. This table shows that in all the considered problems except the small Soybean data set, lattice-based systems could be superior to other systems in terms of prediction accuracy. This result is more significant in the case of the Monks2 problem.

Among the two lattice-based systems, the performance of IGLUE is generally higher than that of LEGAL. Major differences between the two systems are reported on Monks2 and Breast-Cancer data sets. Since lonely examples are not generally taken into account by LEGAL when building the semi-lattice, including entropy function when generating new features allows to avoid such limitation.

Accuracy results of IGLUE will generally increase according to the level threshold while the system will still be able to build additional nodes that really encapsulate the behavior of examples descriptions, i.e. it identifies relevant nodes that contain significant subgroups of examples. A good illustration is given on Monks3 data set (which contains mislabelled examples). Thus the system accuracy increases between the first and second levels, but decreases between the second and third levels. In this special case the adding nodes seems to characterize the mislabelled examples.

Table 13.5 Accuracy results.

<i>Systems</i>	<i>Monks</i>			<i>Breast</i>	<i>Small</i>	<i>Votes</i>
	<i>M1</i>	<i>M2</i>	<i>M3</i>	<i>Cancer W.</i>	<i>Soybean</i>	
LEGAL	100.0	75.6	97.3	79.0	96.9	95.7
IGLUE (h=1)	94.4	94.4	80.5	97.7	99.0	97.7
IGLUE (h=2)	100.0	72.2	92.1	98.0	99.0	98.6
IGLUE (h=3)	100.0	77.3	87.9	98.2	99.0	98.6
C4.5	97.4	67.8	93.9	92.1	96.6	90.6
K*	89.7	58.9	85.7	95.1	100.0	92.0

When dealing with data with a little correlation between examples (this is the case in Monks2 data set) symbolic ML systems fail to learn. The best results reported in the literature for this problem are that of neural network methods. However, IGLUE is able to obtain significant results that are comparable to neural network techniques.

Although taking into account special cases do not guarantee better results, in practice this may be significant as it is the case with the considered data sets.

13.6 CONCLUSION

We have presented a new model for generating continuous-valued features from originally binary features. This model is based on the use of entropy function over the galois lattice framework to induce new features. This framework is the concisely largest search space when constructing regularities among examples. The entropy function allows to take into account lonely examples when building the semi-lattice. This model is a means of preprocessing data before using an appropriated ML system which should deal with continuous-valued features.

To analyze the efficiency of our model, an IBL system, namely IGLUE, has been developed. Our model generates new features on the basis of similarity between initial examples. Thus difficulty that comes out from the application of similarity measures for IBL systems are avoided by the redescription of initial data. A first implementation of IGLUE uses the Mahanalobis measure.

IGLUE is a multi-strategy learning system which combines inductive and IBL approaches.

This system has been compared in practice with the LEGAL system in terms of complexity and classification accuracy. Experimental comparisons reported in this chapter show the effectiveness of IGLUE. Its computational complexity is smaller than that of the lattice-based system LEGAL while its predictive accuracy is as good as that of LEGAL on different well-known data sets. Results of lattice-based systems are better than that of the two other systems: decision tree-based C4.5 and instance-based K^* . However their running time requirements are much more higher than that of the two other systems.

Results of experimentations performed to date and reported in the literature show that IGLUE outperforms other lattice-based systems such as *Rulearn* (Sahami, 1995) or *Galois* (Carpineto and Romano, 1993). This remark is indicated to show the soundness of our method, as we do not have any implementation of these systems.

IGLUE uses a selection function (entropy based) and a distance measure (Mahalanobis). A study to know in which way these two parameters influenced the results should be done. Our intuition is that the system depends mainly on the selection function. Therefore, IGLUE could benefit of applying Bagging (Breiman, 1996) and Boosting (Freund and Schapire, 1996) strategies between different selection functions (Maclin and Opitz, 1997).

IGLUE works with all the training set of examples which could be huge in practice. Ongoing research is dealing with this problem in order to select pertinent examples in the training set for the classification process. Our intention is to demonstrate the potential of Galois Lattice in Concept Learning. Experimentations will also be conducted on other publicly available data sets (Merz and Murphy, 1996).

Acknowledgments

Financial support for this work has been partially made available by the Ganymede project of the Contrat de Plan Etat Nord-Pas-de-Calais. One of the authors, Patrick Njiwoua, is partially supported by the Association Internationale pour le Développement Intégré de l'Afrique (AIDIA). The authors thank M. Rouxel and Assaf Chmeiss for English proofreading and the anonymous reviewers for their helpful comments.

References

- Akkus, A. and Güvenir, H. (1996). K Nearest Neighbor Classification on Feature Projections. In *Proceedings of the Thirteenth International Conference (ICML'96)*, pages 12–19, Bari, Italy.
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.
- Carpineto, C. and Romano, G. (1993). Galois: An order-theoretic approach to conceptual clustering. In *ICML-93 Proceedings*, pages 33–40, Amherst.

- Cleary, J. and Trigg, L. (1995). K*: An Instance-Based Learner Using an Entropic Distance Measure. In *Proceedings of Machine Learning Conference*, pages 108–114, Tahoe City, CA, USA.
- Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference (ICML' 96)*, pages 148–156, Bari, Italy.
- Ganascia, J. (1993). TDIS: an Algebraic Formalization. In *Proceedings of the 13th International Joint Conf. on Artificial Intelligence, IJCAI' 93*, volume 2, pages 1008–1013.
- Garner, S. (1995). WEKA: The Waikato Environment for Knowledge Analysis. In *Proc. New Zealand Computer Science Research Students Conference*, pages 57–64, University of Waikato, Hamilton, New Zealand.
- Koller, D. and Sahami, M. (1996). Toward Optimal Feature Selection. In *Proceedings of the Thirteenth International Conference (ICML' 96)*, pages 284–292, Bari, Italy.
- Maclin, R. and Opitz, D. (1997). An Empirical Evaluation of Bagging and Boosting. In *Proceedings of AAAI'97*, pages 546–551, Providence, USA.
- Mephu Nguifo, E. (1994). Galois Lattice: A framework for Concept Learning. Design, Evaluation and Refinement. In *Proc. of the sixth International Conf. on Tools with Artificial Intelligence*, pages 461–467, New Orleans, Louisiana, LA. IEEE Press.
- Merz, C. and Murphy, P. (1996). Uci repository of machine learning databases.
- Quinlan, J. (1986). Induction of Decisions Trees. In *Machine Learning*, volume 1, pages 81–106. T.M. Mitchell & al. eds.
- Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., Los Altos, California.
- Quinqueton, J. and Sallatin, J. (1982). Expansion and compression on binary data to build feature data. In *Proc. of Intl. Conf. on Pattern Recognition*.
- Sahami, M. (1995). Learning Classification Rules Using Lattices. In *Machine Learning: ECML-95*, pages 343–346, Heraclion, Crete, Greece. Nada Lavrac and Stefan Wrobel eds.
- Salzberg, S. (1991). Distance metrics for instance-based learning. In *Proc. of 6th Intl. Symp. Methodologies for Intelligent Systems*, pages 399–408.
- Vignal, L., d'Aubenton Carafa, Y., Lisacek, F., Mephu Nguifo, E., Rouze, P., Quinqueton, J., and Thermes, C. (1996). Exon prediction in eucaryotic genomes. *Biochimie*, 78(5):327–334.
- Wille, R. (1982). Restructuring Lattice Theory. In Rival, I., editor, *Symposium on Ordered Sets*, pages 445–470. University of Calgary, Boston.
- Zheng, Z. (1996). Effects of Different Types of New Attributes on Constructive Induction. In *Proceedings of 8th Intl. Conf. on Tools with AI, TAI-96*, pages 254–257.

14 CONSTRUCTIVE FUNCTION APPROXIMATION

Paul E. Utgoff and Doina Precup

Department of Computer Science

University of Massachusetts

Amherst, MA 01003

USA

{utgoff|dprecup}@cs.umass.edu

Abstract: The problem of automatically constructing features for use in a learned evaluation function is visited. Issues of feature overlap, independence, and coverage are addressed. Three algorithms are applied to two tasks, measuring the error in the approximated function as learning proceeds. The issues are discussed in the context of their apparent effects on the function approximation process.

14.1 INTRODUCTION

The use of an evaluation function has become a mainstay in building decision-making components of larger systems. Many approaches have been devised for learning such an evaluation function while the system performs its tasks. As learning progresses, the evaluation function presumably improves at estimating the value of each element that it evaluates. Thus, the learned evaluation function approximates the true function that one would like to have in hand. Of central interest is how to formulate this approximation improvement process in a way that attempts to optimize several important criteria. A critical part of this process is to manage the set of features that are used to express the approximation.

The context in which the evaluation function is used and trained is largely immaterial for the discussion here. Typically, one would expect that the evaluation function learning is embedded within a system that is gaining experience through actual or simulated use. We can safely ignore this outer embedding here. However, one must note that there can be important interactions be-

tween function approximation and behavior, when behavior depends on the current evaluation function. For example, in reinforcement learning, the evaluation function, also known as a value function, often represents the discounted reward that one can expect to receive when proceeding from a certain state. However, if one makes each decision in a greedy manner, by making the apparently optimal choice, then certain states (elements of the domain of the evaluation function) are seen more often than others. As the evaluation function changes over time, the probability distribution over the domain will likely change with it.

Various methods for balancing the need to explore suboptimal states and exploit optimal states have been studied at length in the reinforcement learning community. With respect to the function approximation process, the essence of the interaction between the current approximation and the behavior is that the target function to be approximated appears to be non-stationary. Thus, one must ensure that a function approximation process can track a changing target function, which means that all representable approximations must remain reachable during learning. The discussion focuses on the stationary function approximation problem, taking this proviso into account.

The function approximation problem considered here is the following: given a finite domain \mathbf{X} , where each $\mathbf{x} \in \mathbf{X}$ is a vector of Boolean values, and given a stream of point estimates (\mathbf{x}_i, v_i) , find a function $\hat{v} : \mathbf{X} \rightarrow \mathbb{R}$ that minimizes mean-squared error, learning time, and space consumption. The primary concern is to minimize error, but for a given level of error, one wants to use as few resources as possible. The Boolean domain admits many difficult functions. For example, the game of Checkers has 32 squares of no more than five possible values each, which can be encoded in a straightforward manner using 152 Boolean variables. More generally, a discrete variable with a value set of size n can be recoded as n propositional variables. It is also often practical to convert a real-valued variable to a set of discrete intervals, each of which can be converted to a Boolean variable. CMACs (Albus, 1981) and other discretization techniques have been used frequently to provide Boolean encodings for real-valued variables. There are many techniques that operate in the Real domain, but our interest here is restricted to the Boolean domain, which nevertheless contains many useful target functions that one would like to be able to learn.

14.2 THE NEED FOR FEATURES

It is well known that it is typically very difficult to find an accurate approximation directly in terms of the base-level Boolean variables. The functions to be learned over such a domain are typically highly irregular with respect to the models that are often considered. For small domains, a lookup table with one cell per domain element can work very well. However, for d Boolean variables, one needs a table of 2^d cells, so this approach does not scale well to large numbers of Boolean variables. One can think of a lookup table of this kind as an unbiased function approximator. However, even when one can afford the space for a table, one must visit each cell often enough to attain a good estimated

for the value of the function at that point (cell). Using a lookup table in this manner amounts to rote learning, which is devoid of generalization. Without a lookup table, how can one construct an approximation of high accuracy?

A common approach is to introduce an intermediate representation, specified as a set of features. In the discussion here, the term *feature* refers to a computable function of the base-level variables. In order to avoid confusion, the term *input feature*, frequently used in the literature, is replaced by *base level variable*. Given a vector of features \mathbf{f} , a domain element \mathbf{x} is mapped to a vector of feature values, and \hat{v} maps a vector of feature values to a real value. The function approximator needs to construct features to include in \mathbf{f} , and it needs to delete features from \mathbf{f} . Given a particular extant \mathbf{f} , the approximator needs to combine the feature values numerically so that \hat{v} is computable. The features of \mathbf{f} recode the base-level variables so that fitting a model to the instances as mapped by \mathbf{f} is easier to do.

Feature construction and feature selection are dual problems. In principle, one could construct an approximation in which all possible features were represented explicitly. Then one could either remove features selectively, or mark each feature as being ‘in’ or ‘out’. This would have the flavor of a feature selection problem. Alternatively, one could start with no features, and then subsequently construct any needed feature or delete any existing feature not currently needed. This would have the flavor of a feature construction problem. However, these are just two views of the same problem: deciding which features to use in the approximation. For simplicity, the feature construction view is adopted here.

14.3 FEATURE SPACES

There are many ways in which the computed feature values could be combined to compute the estimated value of the function for a domain element. Many forms of combination can be restated as a linear combination of a vector \mathbf{W} of real-valued weights and the vector of feature values, and this is the form that is discussed below. Although this form may appear to limit the space of describable evaluation functions, it is quite useful, and should be seen as providing a context in which to find a good set of features for representing the approximation of the target function. The function approximation problem becomes one of finding a good set of features, given that their values will be combined linearly with a weight vector to estimate the value of a domain element.

The domain of each feature is the set of base-level Boolean variables, in xxboolean variable and the range of each feature is the set $\{0, 1\}$, for Boolean features, or the interval $(0, 1)$ for sigmoid features. Given that sigmoid function values are close to zero or to one at all points, except near the transition, the range can be considered to be $\{0, 1\}$ for all the feature types under consideration here. For each feature, its value is 1 if the domain element is covered by the feature, and its value is 0 otherwise. This kind of feature is itself a nonlinear function over the Boolean domain. When the linear combination is

computed, a scalar is multiplied with the feature value, thus scaling it. Hence, each feature can be seen as describing a set of domain elements that share a common intrinsic property, contributing additively to the value of a domain element. The problem of finding good features becomes the problem of finding useful intrinsic properties of the domain elements.

Several issues arise with respect to the set of features in the approximation. First, are the features linearly independent? This has a direct impact on whether the weight updating algorithm can find a weight vector \mathbf{W} that globally minimizes the mean-squared error in the approximation. Without linear independence, a weight updating algorithm is likely to become trapped at a local minimum that is not a global minimum. Though becoming trapped at a local minimum may still produce a useful evaluation function, one would rather avoid such a situation.

Second, are the features orthogonal? Orthogonal features form a basis for the instance space. Consequently, the weights of such features adjust independently of each other, and learning becomes significantly faster. Given the range of the features described above, the orthogonality condition implies the restriction that features should not overlap. When two features overlap, and point estimates are received (serially) from within the domain of each feature individually, and from within the intersection of the two features, then the weight corrections for the two features interact. While this is often workable, learning is slowed compared to updating weights for two features that do not overlap. This slowing is compounded as the number of overlapping features increases. For d distinct Boolean overlapping features, there are 2^d distinct regions of the domain. The number of distinct regions, and the amount of overlap in each have a large effect on the number of weight adjustments that are needed to minimize the function error, given those features.

Third, how large is the domain of each feature? This size has an effect on how quickly the weight that corresponds to that feature will be adjusted. This is because the weight associated with the feature is adjusted when the feature covers the observed domain element, and left alone otherwise (by virtue of having value 0). For the moment, assume that every domain element of the function is equally likely to appear in the observed point estimates. Then a feature with a larger domain is more likely to have its weight adjusted than a feature with a smaller domain. The feature with the smaller domain will probably take longer to reach its final value, given the influence of the other features. While the uniform distribution over domain elements is unlikely, one would still expect to see a large number of the domain elements, and hence notice this effect.

Finally, what sets of domain elements can a feature be defined to cover? Thinking of the function domain as a Boolean hypercube, one can define a feature as a hyperplane through the cube, with domain elements on the positive side of the plane being covered by the feature, and the rest being not covered. Any feature defined by a hyperplane that is either orthogonal or parallel to all the axes is called an *axis-parallel* feature. Any feature that is defined by a

hyperplane, whether or not it is axis-parallel, is called an *oblique* feature. One could devise other methods to specify a particular set of domain elements for a feature definition, but no such additional methods are considered here.

14.4 COMPARISON OF THREE FUNCTION APPROXIMATION ALGORITHMS

In order to gain some insight into how different algorithms for constructing features affect learning, three such algorithms are compared here on two stationary function approximation problems. The primary concern is how quickly mean-squared error falls as training points are received and used to update the approximation in a serial manner. A secondary concern is the size of the approximation. How many features are there, and how complex is each one? In each case, the number of bits needed to encode the approximation is estimated, providing a common basis for the size comparison.

14.4.1 DNC

Ash's (1989) Dynamic Node Creation (DNC) adds a new feature in the form of sigmoid function of a linear combination of the base level variables. The sigmoid function serves the purpose of providing a threshold, but it is differentiable, which helps the feature weight adjustment process. A feature covers those domain elements for which the feature has a value near one, and excludes those elements for which the feature has a value near zero. These features are also known as the hidden units of an artificial feed-forward network. For the problem considered here, the network output value is a linear combination of the feature values.

Each feature is defined by an oblique boundary. The features also overlap. DNC uses gradient descent to propagate the error in the approximation to the individual features, and again to the weights of each feature. Adjusting the weights of the linear combination that defines the feature corresponds to changing the definition of that feature, which can also be seen as deleting one feature and adding another. The features that are created are dependent on the output weights. As a result, the error surface for the function being approximated has local minima, in which the function approximation algorithm can become trapped.

When the mean-squared error in the approximation asymptotes at too high a level, the algorithm adds a new feature. In our implementation, the weights within a new feature that correspond to the base-level variables are initialized to 0. Therefore, each new feature value is initially 0.5 on the threshold of the sigmoid feature. The weight for this feature that is used in the linear combination of the approximation is also initialized to 0, which means that the new feature makes no initial contribution to the approximation. This has the nice property that adding a new feature has no immediate effect, and the new feature comes to contribute to the approximation smoothly over time as it is tuned in. Note that the weight initializations to 0 are different from Ash's use

of small random values. Random values are not needed here, and the algorithm produces repeatable results for the same stream of observations.

14.4.2 ELF

The ELF algorithm (Utgoff, 1996) constructs each feature as a Boolean function of the base-level variables. The Boolean function is represented as a pattern, and it covers only those domain elements that match the pattern. Each component of a pattern has either the value ‘#’ or the value ‘0’. A ‘#’ matches either of the possible values of the corresponding base-level variable, while a ‘0’ in the pattern matches only a ‘0’ (False) value. The pattern of all ‘#’ covers every domain element because the pattern matches any domain element at every component. The pattern of all ‘0’ covers the one element in which all the base-level variables have value ‘0’. One pattern is strictly more general than another if and only if it covers all the domain elements covered by the other.

This pattern language is somewhat unusual because it can specify that a base-level Boolean variable must have value ‘0’, but not that it must have value ‘1’. Nevertheless, the pattern language is sufficient to describe any evaluation function over the domain. Consider a trivial domain of one Boolean variable. The pair of features ‘#’ and ‘0’ is sufficient because the feature ‘0’ covers just one element, and the feature ‘#’ covers both elements. One can assign a weight to the ‘#’ feature that is correct for the ‘1’ element, and one can assign a weight to the ‘0’ feature so that it is correct for the ‘0’ instance, given that the ‘#’ feature also covers it, and has its own weight. This argument extends to any number of Boolean variables. There is no need for a ‘1’ in this pattern language. The boundaries of an ELF feature are axis-parallel.

When a training point is presented to the ELF algorithm, it takes two steps. First it updates the adjustable weights of the approximation, and then it updates the features in f as necessary. The weights are updated using the Widrow-Hoff rule (Widrow & Hoff, 1960) for iteratively minimizing the mean-squared error of the evaluation function. One computes an amount by which to alter the weights of those features that matched the instance. The update rule takes this form here because the features evaluate individually to 0 or 1. The amount by which to adjust each weight is the error divided by the number of features that matched, multiplied by a stepsize parameter α that is fixed at 0.1.

Initially, the function approximation consists of the one most general feature, with a weight of 0. This initial approximation returns 0.0 for every element. As point-estimates are observed, the single weight of the single feature is adjusted in an attempt to minimize the mean-squared error. This amounts to trying to fit the points with a constant function. By itself, this would not be very good, but one can gather useful information while this vain attempt at fitting is taking place, and then use that information to add a useful feature that will help subsequent attempts.

While ELF is attempting to adjust the feature weights, initially just the one weight, it accumulates the error that is associated with each bit (base-level Boolean variable) being on (true) in an instance. With each feature, ELF

maintains a separate vector of bit-errors that are accumulated by attributing the amount of correction that has been applied to the feature weight to each of the '#'s in the feature pattern whose corresponding base-level variable was true. This information is of central importance to the procedure for updating the features, but plays no role in updating the feature weights. This is much like taking an X-ray picture. One bombards the X space with error, which paints a picture as defined by the accumulated bit errors. These errors provide very useful information about those subsets of the domain to which one would like to be able to assign different values.

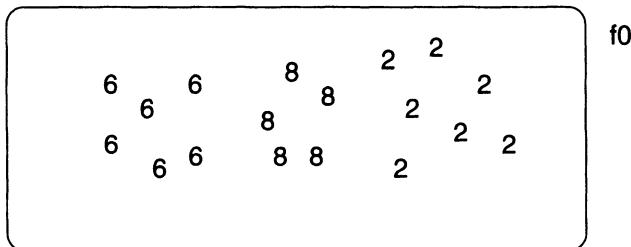


Figure 14.1 Target Values

Consider a simple illustration. Imagine that the space of domain elements is as depicted by the Venn diagram shown in Figure 14.1. The points are not shown, but some of their target values are. Some points should evaluate to 6, while others should evaluate to 8, while still others should evaluate to 2. As the weight of the most general feature f_0 is repeatedly adjusted, suppose that it settles to the value 5. Then the error associated with each of these points would be as shown in Figure 14.2.

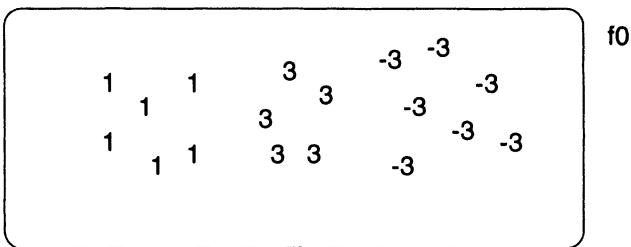
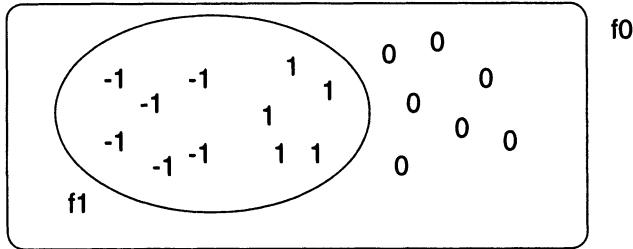
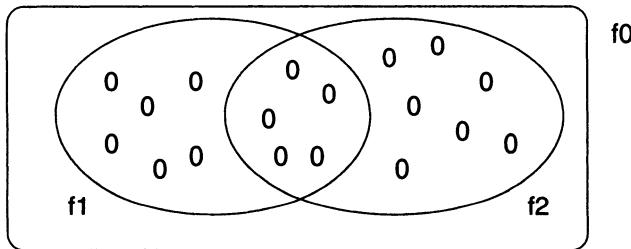


Figure 14.2 Error with One Feature

Assume that a new feature f_1 has been constructed that covers the 6s and the 8s. Through subsequent point-wise error adjustments, suppose that the most general feature f_0 takes on a weight of 2, and the new feature f_1 takes on a weight of 5. Now the new X-ray paints the error picture shown in Figure 14.3. These errors are smaller than before.

**Figure 14.3** Error with Two Features

Finally, assume that a new feature f_2 has been constructed that covers the 8s and the 2s. Through more point-wise error adjustments, suppose that f_0 takes on a weight of 0, f_1 takes on a weight of 6, and f_2 takes on a weight of 2. Now a new X-ray would be transparent, as shown in Figure 14.4 because the error is uniformly 0.

**Figure 14.4** Error with Three Features

To accomplish assigning one value to some instances and not others, one constructs a new feature that covers just those instances. This is done by identifying the feature that is least able to reduce the error that is attributed to it. In the initial approximation, there is the single most general feature that is trying to fit all the points with a constant value. Even after other features come into existence, the problem is still the same with respect to any one feature. The other features simply transform the values of the instances they cover. Every feature tries to fit the points it covers with a constant function. The feature contributes value 0 for those instances it does not cover, and the value of its weight for all others. One needs to find the feature that is fitting its points least well, given all the other features.

To identify the feature that fits its points least well, given the other features, one looks for the feature with the greatest spread between the maximum and minimum bit-wise error. This value is called the *warp* of a feature, because the bitwise errors indicate a desired direction in which to move the weight of the feature. The feature is not actually warped, as it has only its single scalar

weight, but the bitwise errors can be seen as stresses on the feature weight related to different segments of the domain.

To change the set of features, ELF identifies the feature with the largest warp value, then identifies the ‘#’ in the feature whose accumulated error is most different from the mean accumulated error for that feature. ELF then makes a copy of the feature, changes the ‘#’ to a ‘0’ in the copy, initializes the weight of the new feature to 0, and adds it to the set of features in \mathbf{f} . By initializing the weight to 0, adding the new feature has no immediate effect on the approximated function. However, subsequent weight adjustments will use the new feature like any other, moving its weight to a more useful value.

Every feature definition can be reached in principle, even though the only method for revising a feature definition is to specialize it. Whenever a new feature is added, the accumulated bit errors and several other bookkeeping variables are reset for all the features. However, the feature weights \mathbf{w} are left untouched. In addition, it would be quite useless to allow two or more features with identical patterns, so ELF prevents this by not considering any specialization that would produce a duplicate pattern.

The algorithm deletes a feature whose weight has been near to 0 for an extended length of time. This is accomplished by considering features for deletion only at the same time that one might specialize a feature. For a feature that has both its minimum and maximum observed weight near 0, the feature is deleted. However, the most general feature is never deleted. This is critical, so that any feature definition remains potentially reachable through specialization. Deletion is not an important part of the algorithm. It is included to weed out useless features for the sake of efficiency. The ELF algorithm would work without deletion, though slightly less efficiently. One would need to omit this deletion mechanism to provide a guarantee that the algorithm will converge to 0.0 error in the limit.

Because ELF specializes bits that are associated with the largest errors, those features that need high magnitude weights tend to be identified earliest. This has the desirable effect of reducing error early in the learning process. As features are found that reduce error, the new errors that emerge tend to be related to features that have smaller magnitude weights. The effect is to keep reducing residual error. There is no random element of the algorithm itself. Given the same stream of observed point-wise errors, the algorithm will produce the same result every time, which is highly desirable in terms of understandability and reproducibility. The algorithm decides when and where to add new features, based on need.

To illustrate ELF, the algorithm was embedded in a program that also includes a target function v represented separately from the approximated function \hat{v} . The target is in the same form that ELF uses for its approximation. The program repeatedly samples a point at random from \mathbf{X} , evaluates it with the target v and then delivers the point estimate to ELF. This loop continues to execute until a decaying average of the absolute errors drops below 10^{-6} .

Table 14.1 Artificial 4¹⁰ Task

Target v	ELF \hat{v}	
	Weight	Feature
-93.9527350	fffffedffff	fffffedffff
-76.8096050	feeefedde	feeefedde
74.4897331	ffef79dfffe	ffef79dfffe
-74.4897259	ffef79dcfe	ffef79dcfe
74.0239411	fefbefffffb	fefbefffffb
-74.0239410	fefbefffffab	fefbefffffab
45.6317710	edfef7efdf	edfef7efdf
-30.6775867	fffffff	fffffff
14.2437743	bfbfdfe6de	bfbfdfe6de
-10.0314592	ff3f3ff7fd	ff3f3ff7fd
10.0314589	ff3fcff7fd	ff3fcff7fd
1.9227299	fffffdffffe	fffffdffffe

The target v and the final approximation \hat{v} are shown in Table 14.1, though 31 features all with magnitudes below 10^{-4} have been omitted. A total of 63 features were created, with 20 of them deleted, leaving 43 total features, 12 of which appear in the figure. Each feature is shown as its weight and its pattern coded in hexadecimal, where '#' is coded as bit-value 1 and '0' is coded as bit-value 0. The target function consists of 256 regions over a domain of 1,048,576 elements. Comparing the features of \hat{v} with those of v , one sees that ELF found an approximation in terms of the intrinsic properties in the target. There are three cases of a pair of features that collectively match a single feature in the target. For example, **ffef79dfffe** and **ffef79dcfe** in the approximation collectively represent **ffef79d3fe** in the target.

A decaying average of the mean squared error is plotted in Figure 14.5, with the number of training points shown in tens of thousands. The error diminishes quickly, with revision of f continuing long after the squared error has attained a low value. Each dot along the curve shows when some feature was (copied and) specialized by one bit (changing a '#' to a '0'). Each square shows when some feature was deleted.

The components of the evaluation function (the feature weights) are shown in Figure 14.6. The x-axis represents the number of training elements observed (in tens of thousands), as it does for the previous Figure 14.5. One can see how the weight of each feature changes over time. Dots along the same curve indicate that the feature has spawned multiple specializations, and not that the feature itself has become very specialized. After a feature has been copied and specialized, the weights of all the features continue to adjust through subsequent training.

One sees that upon specialization, feature weights 'travel' until a new set of weights that minimize the squared error are attained. Often, many weights

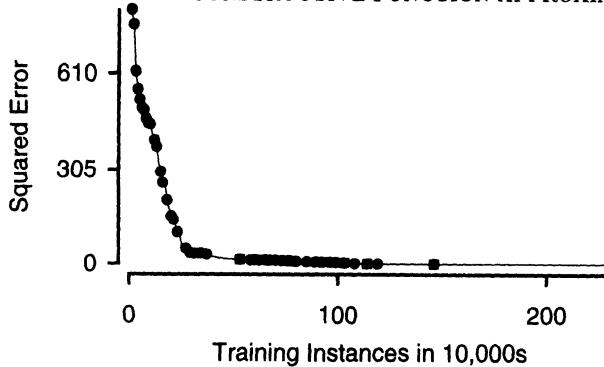


Figure 14.5 Squared Error for Artificial Task

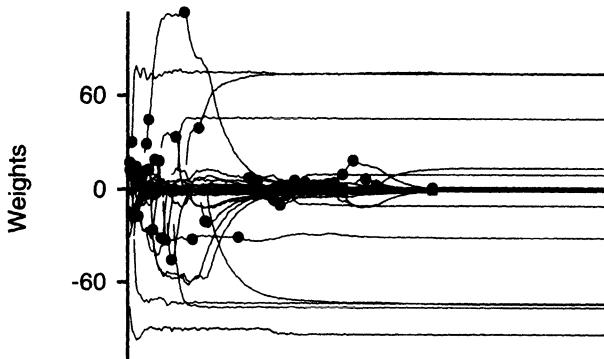


Figure 14.6 Feature Weights for Artificial Task

change at one time, due to sympathetic movement of overlapping features. When a feature is (copied and) specialized, the new specialized feature has an initial weight of 0. These new features are clearly visible in the graph. By looking at the dots on the curves, one can find the new feature that is born at that time.

One can observe two characteristics in the figure. First, the features with larger magnitude weights tend to be found earlier than those with smaller magnitude weights. Second, specialization often occurs repeatedly on the same feature. One can see several consecutive specializations occurring on the same feature, by finding a succession of feature creations. One can speculate that as the feature begins to cover instances that share an intrinsic property, the bit errors are more clearly attributable to certain bits, making those features least adequate under the stress measure described above.

14.4.3 RT

The RT algorithm is somewhat similar to the ELF algorithm. One major difference is that RT instead constructs features that do not overlap. This is done by using a slightly different pattern language. A ‘#’ matches either True or False in the corresponding Boolean variable. A ‘0’ matches only False, and a ‘1’ matches only True. Presence of a ‘1’ in the pattern language differs from ELF. The set of features is initialized with the most general feature. When it becomes time to specialize a feature, two new features are constructed, and the parent feature is deleted. In one of the new features, the pattern is that of the parent, except that the identified ‘#’ has been changed to a ‘0’. Similarly, in the other new feature, the ‘#’ has been changed to a ‘1’. The weights for the two new features are initialized to that of the parent. These two features that have replaced the parent are now free to have their weights adjusted independently. Indeed, every element of the domain is always covered by exactly one feature. This is good for orthogonality, but it loses the idea of identifying intrinsic properties.

This strategy for modifying the set of features used in the approximation can be seen as a method for building a variable-resolution lookup table. Variable resolution methods for discretizing continuous variables have been used successfully in reinforcement learning tasks (Chapman & Kaelbling, 1991; Moore & Atkeson, 1995). This strategy can also be seen as forming a regression tree whose leaf values adjust continually to observed errors at that leaf (Breiman, Friedman, Olshen & Stone, 1984).

Because the features do not overlap, the RT algorithm cannot produce duplicate features. Because there are no feature interactions, the bit-errors for each feature are not cleared when a feature is split into two, except that the bit-errors for the two new features are initialized to 0.0.

14.4.4 TicTacToe

The TicTacToe function is defined over the 4,520 states that are reachable during legal play. Every position is expressed from the point of view of the player that is about to move. A position in which a win has been achieved or is attainable has value 10.0, a draw has value 0.0, and a position that is lost has value -10.0. The fact that the function comes from a sequential problem solving task is irrelevant here because it is treated simply as a stationary function approximation problem. This is done by uniformly sampling the 4,520 points (with known target values) repeatedly in an offline manner.

The concern is how various aspects of the feature construction process affect error reduction in the approximation. A regression tree based on minimizing an information criterion contains 1,467 leaves, which suggests an upper bound on the number of features needed for the approximation. As represented here, there are 27 base level variables for the TicTacToe function. This is because each of the nine squares can have one of three possible values. More compact codings are possible, but this one is straightforward.

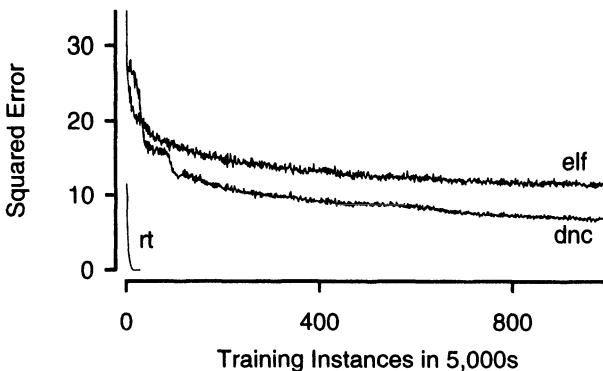


Figure 14.7 TicTacToe Error Graph

The number of features constructed by each algorithm, and the number of bits necessary to encode each feature are quite different. DNC constructed a function with 37 features. The representation for each feature contains 27 weights and there is also one more weight for the final linear combination. Assuming that each weight can be represented as a 32-bit floating point, the approximated function can be encoded in $(27 \times 32 + 32) \times 37 = 33,152$ bits.

ELF constructed 282 features, and deleted 26, for a net total of 256 features. Each feature can be encoded using one bit for each base level variable, and a real-valued weight. Using the same convention as above, the approximated function can be encoded in $(27 \times 1 + 32) \times 256 = 15,104$ bits.

The function constructed by RT was much larger, consisting of 5,245 features. Features are similar in size to those of ELF, except that a pattern for RT requires twice as many bits as a pattern for ELF. The encoding for the RT approximation takes $(27 \times 2 + 32) \times 5,245 = 451,070$ bits. Note also that there are 5,245 features over a domain in which only 4,520 different states (points) can occur. Clearly RT is overspecializing, which means the criterion for when to split a feature is too aggressive.

14.4.5 Eight Puzzle

The eight-puzzle is a sliding tile puzzle that contains eight square tiles in a flat space that could accommodate nine tiles in a 3x3 arrangement. By having just eight tiles, there is an empty location, into which one can slide one of the adjacent tiles. There are 181,440 states, including a single goal state consisting of all tiles in row order (the tiles are numbered 1-8), with the empty location in the lower right corner. The longest path to this goal state requires 31 steps, providing 32 different values in the range of the target function, which contains 25,077 local maxima (states from which every decision is equally good) and one minimum. A regression tree based on minimizing variance of its blocks contains 114,114 leaves. This function is apparently difficult to compress.

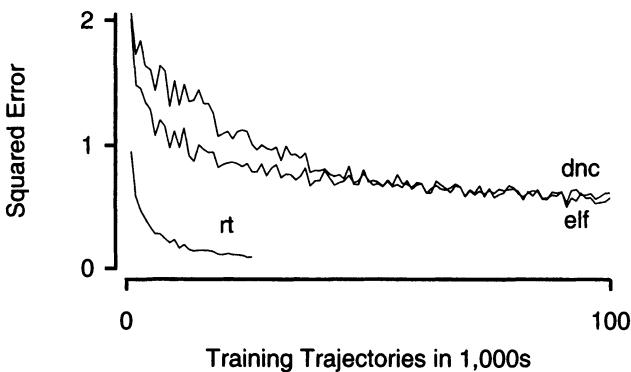


Figure 14.8 8-puzzle Error Graph

The number of base-level variables for this function is 81, because each of the nine locations can have one of nine possible values. More compact encodings are possible, but this one is straightforward. DNC constructs a function of 15 features, which needs $(81 \times 32 + 32) \times 15 = 39,360$ bits for the encoding. ELF constructs 222 features and deletes 25, for a net total of 197 features. This approximation can be encoded in $(81 \times 1 + 32) \times 197 = 22,261$ bits. RT constructs a function of 14,065 features, and the encoding requires $(81 \times 2 + 32) \times 14,065 = 2,728,610$ bits.

14.4.6 Discussion

The three algorithms were each run on both tasks. Within each problem, the sequence of sampled points was identical for each algorithm. For the TicTacToe function, the reachable points in the domain were sampled randomly according to a uniform probability distribution. For the 8-puzzle function, the points were sampled from optimal trajectories, with the correct target value being used for each point. In this formulation, the 8-puzzle is a stationary function approximation task, but with a non-uniform probability distribution over the domain. Only the starting point of each trajectory is sampled randomly and uniformly from the domain. These differences in the sampling distributions (for TicTacToe versus 8-puzzle) occur due to the way these applications were implemented previously, not because of a deliberate design choice made here.

For TicTacToe, Figure 14.7 shows that DNC and ELF become slow at reducing the error. Both algorithms produce features that overlap. DNC has features that are not linearly independent, so an additional cause of the slowed rate of error reduction may be due to reaching a local minimum with the current feature set. ELF has features whose coverage is individually decreasing as points are seen. Although one can expect ELF to converge to 0.0 error in the limit, the feature overlaps and shrinking feature coverage work against that, causing the rate of error reduction to dwindle. DNC suffers the same fate, due to feature overlap and linear dependence. DNC does a better job of error

reduction, suggesting that oblique features are more useful than axis-parallel features for this function. The same figure shows that the RT algorithm reduces error very quickly. It does however create a large number of features, which slows execution. When each of the runs was halted, DNC had 37 features, ELF had 256 features, and RT had 5,245 features.

For the 8-puzzle, Figure 14.8 shows that all three algorithms reduce the approximation error quite quickly. RT reduces error most quickly, but creates 14,065 features. DNC and ELF come much closer to RT as learning progresses, but still trail. However, each of DNC and ELF use many fewer features than RT in the process. One can surmise that DNC and ELF have each identified intrinsic properties. When halted, DNC had 15 features, and ELF had 197 features. Given the similar accuracies of DNC and ELF, it appears that oblique features have little apparent advantage over axis-parallel features for this particular function. As was observed for the TicTacToe function, the rate at which DNC and ELF reduce error becomes very slow.

One real-world task that we have studied is the instruction scheduling task. The goal is to learn a greedy evaluation function for picking the instruction that should be scheduled next for a basic block during the code-generation phase of code compilation. The task is known to be hard, and it is becoming increasingly important in the context of highly pipelined architectures. It was possible to learn to produce instruction schedules with running times within 10% of optimal on a DEC Alpha architecture. The function that needs to be learned in this case is much simpler than either of the tasks chosen as illustrations above.

14.5 RELATED WORK

A variety of constructive methods have been devised for classification problems. Several algorithms have been designed for constructing networks of thresholded logic units, by adding boundaries that correct for misclassified examples (Parekh, Yang & Honavar, 1997). It should be possible to extend these techniques to function approximation problems.

A meiosis network (Hanson, 1990) is a feed-forward network in which the variance of each weight is maintained. For a hidden unit (feature) that has one or more weights of high variance, the unit is split into two. The input weights that define the feature, and the output weight for the linear combination are altered so that the two units are moved away from their means in opposite directions.

Wynne-Jones (1992) presents an approach called *node splitting* that detects and attempts to repair an inadequate hidden layer (set of features) of a feed-forward artificial neural network. The system detects when the hyperplane of a hidden unit (weights of an oblique feature) is oscillating, indicating that the unit is being pushed in conflicting directions in feature space. Such a unit is split into two units, and the weights are set so that the units are moved apart from each other along an advantageous axis. Although this approach sometimes works well, Wynne-Jones observes that the units often work back toward each other instead of diverging.

Fahlman and Lebiere's (1990) cascade correlation method constructs a new hidden unit (feature) and freezes its defining weights. The original input variables and the newly constructed unit become the input variables for the next layer. Thus, one adds a new feature and a new layer of mapping at the same time. The algorithm alternates between adding a new unit/layer, and adjusting the weights for the output units. The algorithm has produced good results when applied to classification tasks.

Fritzke (1993) has developed 'growing-cell-structures', an improvement to self-organizing feature maps. The algorithm builds an approximation in the form of an interpolator, adding refinement and precision when and where needed.

The RT algorithm is highly similar to Chapman and Kaelbling's (1991) G algorithm. The G algorithm splits the domain on Boolean variables that are deemed relevant to the function value by a statistical test. Chapman and Kaelbling note that the G algorithm will fail when bits are irrelevant in isolation, but relevant collectively. The RT algorithm will continue to split the space whenever the error for a feature is too high. Chapman and Kaelbling suggest that perhaps good features should be orthogonal, and therefore individually relevant.

14.5.1 Conclusions

There are several issues to consider when building a function approximator. The accuracy of the approximation is typically of paramount importance, though it is worth noting that when using an approximation to implement a decision-making component, one can tolerate error in the approximation when it does not lead to errors in decision making.

With regard to oblique or axis-parallel features, one can see here, and in many other examples, that oblique features can be very useful in building the approximation. Given that the set of axis-parallel features is properly contained in the set of oblique features, it would seem obvious to use oblique features. However, current methods for finding such features do not guarantee that those features are linearly independent, which is a significant disadvantage.

Whether the features overlap has an important effect on the rate at which error is reduced. The number of distinct regions over the domain grows exponentially with the number of features. One can conclude that feature overlap is detrimental. On the other hand, if one is attempting to identify intrinsic properties, then overlap is necessary. When overlap is precluded, the features become orthogonal, and can be adjusted independently. One can reduce error more quickly by avoiding overlap, but at the expense of not finding intrinsic properties, and at the expense of requiring more total features. It may well be that one should first find useful intrinsic properties (as with DNC or ELF), and only then proceed to partition the resulting function (as with RT).

In order for these algorithms to be useful for reinforcement learning techniques, they should be able to track concept drift. DNC and ELF theoretically have this property. Further investigation is needed in order to establish how

well they behave in practice. RT commits to a partition of the domain, so it is less suitable for non-stationary tasks. It would track the drift, but with a potentially rigid and therefore poor partition of the domain.

Much more exploration of constructive function approximation methods is needed. It would be useful to design a constructive method that produces oblique features that are linearly independent. It would be useful if the amount of feature overlap could be controlled dynamically, so that slowed learning and creating a large number of features can be balanced automatically. None of the algorithms discussed here or elsewhere truly solves the feature construction problem. A better understanding of the tradeoffs is needed, as well as better control of these tradeoffs in the constructive function approximation algorithms that are to be used in practice.

References

- Albus, J. S. (1981). *Brain, behaviour and robotics*.
- Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science*, 1, 365-375.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.
- Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparison. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 726-731). Sidney, Australia: Morgan Kaufmann.
- Fahlman, S. E., & Lebiere, C. (1990). The cascade correlation architecture. *Advances in Neural Information Processing Systems*, 2, 524-532.
- Fritzke, B. (1993). Kohonen feature maps and growing cell structures: A performance comparison. *Advances in Neural Information Processing Systems*, 5, 123-130.
- Hanson, S. J. (1990). Meiosis networks. *Advances in Neural Information Processing Systems*, 2, 533-541.
- Moore, A. W., & Atkeson, C. G. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 199-233.
- Parekh, R., Yang, J., & Honavar, V. (1997). *Constructive neural network learning algorithms for multi-category real-valued pattern classification*, Iowa State University, Computer Science.
- Utgoff, P. E. (1996). *ELF: An evaluation function learner that constructs its own features*, (Technical Report 96-65), Amherst, MA: University of Massachusetts, Department of Computer Science.
- Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *Convention Records of the Western Conference of the Institute for Radio Engineers* (pp. 96-104).
- Wynne-Jones, M. (1992). Node splitting: A constructive algorithm for feed-forward neural networks. *Advances in Neural Information Processing Systems* (pp. 1072-1079).

IV Feature Construction

15 A COMPARISON OF CONSTRUCTING DIFFERENT TYPES OF NEW FEATURE FOR DECISION TREE LEARNING

Zijian Zheng

School of Computing and Mathematics

Deakin University

Geelong, Victoria 3217

Australia

zijian@deakin.edu.au

Abstract: This chapter studies the effects on decision tree learning of constructing four types of new feature (conjunctive, disjunctive, M-of-N, and X-of-N representations). To reduce effects of other factors such as tree learning methods, new feature search strategies, search starting points, evaluation functions, and stopping criteria, a single tree learning algorithm is developed. With different option settings, it can construct four different types of new feature, but all other factors are fixed. The study reveals that conjunctive and disjunctive representations have very similar performance in terms of prediction accuracy and theory complexity on a variety of concepts, even on DNF and CNF concepts that are usually thought to be suited only to one of the two kinds of representation. In addition, the study demonstrates that the stronger representation power of M-of-N than conjunction and disjunction and the stronger representation power of X-of-N than these three types of new feature can be reflected in the performance of decision tree learning in terms of higher prediction accuracy and lower theory complexity.

15.1 INTRODUCTION

Conventional inductive learning systems, called *selective induction*, learn theories by selecting task-supplied features. When these features are not appropriate for describing theories, the performance of selective induction systems

in terms of prediction accuracy and theory complexity is poor. One possible solution to this problem is *constructive induction* (Michalski, 1980). This type of inductive system transforms the original instance space into a more appropriate space by constructing *new features*. Task-supplied features are called *primitive features* in contrast to new features.

Many constructive induction systems have been developed. They perform differently in different domains. Different systems use different constructive operators, different theory description languages, different strategies such as a data-driven strategy, a hypothesis-driven strategy, and a knowledge-driven strategy (Wnek and Michalski, 1994), different new feature evaluation functions, different new feature search methods, and so on. Constructive operators play an important role in constructive induction. This is worth investigating. However, all other factors may also affect the performance of a learning system. Each of them may, either positively or negatively, contribute to performance differences between different algorithms.

Most constructive induction systems such as FRINGE (Pagallo, 1990) and LFC (Ragavan and Rendell, 1993) use conjunction or disjunction as constructive operators. That is, the constructed features are conjunctions or disjunctions of other features. A few systems use other constructive operators, for example, M-of-N (Murphy and Pazzani, 1991), "feature counting operators" (Michalski, 1980; Bloedorn et al., 1993), and X-of-N (Zheng, 1995a). A comparison of two constructive decision tree learning systems showed that the one constructing conjunctions as new features outperformed that constructing disjunctions for DNF concepts, and vice versa for CNF concepts (Pagallo, 1990). Murphy and Pazzani (1991) demonstrated that constructing M-of-N representations can improve the performance of decision tree learning. Zheng (1995a) showed that X-of-N representations are more representationally powerful than conjunctive, disjunctive, and M-of-N representations. Nevertheless, all the comparisons involved in these studies use different learning algorithms.

To explore the effects on constructive induction, more specifically on decision tree learning, of conjunctive, disjunctive, M-of-N, and X-of-N representations as new features, we compare them by using a single constructive decision tree learning algorithm. It constructs one of these four types of new feature when the corresponding option setting is specified. By doing so, all other factors including those mentioned above that might affect the performance of a constructive induction algorithm are fixed. Effects on the performance of algorithms introduced due to differences of these factors are removed.

We will show that, with this kind of algorithm, conjunctions and disjunctions perform very similarly on different types of concept, even on DNF and CNF concepts. We will also illustrate that M-of-N is more representationally powerful than conjunction and disjunction, while X-of-N is the most powerful among them. The algorithm with X-of-N representations as new nominal features performs well in both artificial and real-world domains under investigation.

The following section describes the four types of new feature. Section 15.3 presents the learning algorithm developed for the comparison study. Sec-

tion 15.4, then, experimentally compares the four types of new feature in a set of artificial and real-world domains, followed by discussions in Section 15.5 and related work in Section 15.6. The final section concludes with future work.

15.2 FOUR TYPES OF NEW FEATURE

All the four types of new feature (conjunctive, disjunctive, M-of-N, and X-of-N representations) studied here are based on a set of conditions (feature-value pairs). Note that primitive continuous-valued features are transformed into binary or nominal features by discretization before learning occurs. The conjunctive representations and disjunctive representations are conventional logical conjunctions and disjunctions of conditions respectively. M-of-N representations are at-least M-of-Ns. That is, given a set of conditions, an M-of-N representation answers whether at least M of conditions in the set are true. An X-of-N representation states how many conditions in its set are true for a given example. A definition of X-of-N and M-of-N representations is given below.

Definition

Let $\{F_i \mid 1 \leq i \leq \text{MaxFea}\}$ be the set of features of a domain, and for each F_i , $\{V_{ij} \mid 1 \leq j \leq \text{MaxFeaVal}_i\}$ be its value set, where MaxFea is the number of features, and MaxFeaVal_i is the number of different values of F_i .

- An **X-of-N** is composed of a set of feature-value pairs, denoted as:

$$X\text{-of-}\{FV_k \mid FV_k \text{ is a feature-value pair denoted as } "F_i = V_{ij}"\},$$

$$1 \leq k \leq N_+, N \leq N_+, 1 \leq N \leq \text{MaxFea} \quad \}.$$
The value of an X-of-N representation can be any number between 0 and N . Given an instance, its value is x iff x of the FV_k are true.
- An **M-of-N** is composed of a set of feature-value pairs and a number M , denoted as:

$$M\text{-of-}\{FV_k \mid FV_k \text{ is a feature-value pair denoted as } "F_i = V_{ij}"\},$$

$$1 \leq k \leq N_+, N \leq N_+, 1 \leq N \leq \text{MaxFea}, 1 \leq M \leq N \quad \}.$$
The value of an M-of-N representation is either true or false. Given an instance, its value is true iff at least M of the FV_k are true; false otherwise.

N_+ is the number of feature-value pairs in a representation, called the size of the representation, and N is the number of different features that appear in the representation. A feature-value pair FV_k ($F_i = V_{ij}$) is true for an instance iff feature F_i of the instance has the value V_{ij} .

The M-of-N representation is a special case of the X-of-N representation, while the conjunctive and disjunctive representations are two special cases of the M-of-N representation. Conjunction and disjunction can be thought as N-of-N and 1-of-N respectively.

For logical problems, a conjunction is the negation of a disjunction which has the same size as the conjunction, and vice versa. Therefore, for decision tree learning, the representation power of conjunctions and disjunctions should be the same, even on DNF and CNF concepts that are intuitively thought to be

suited only to conjunctions and disjunctions respectively. For example, given a DNF concept: $(A \wedge B) \vee (C \wedge D) \vee (E \wedge F)$, its decision tree representation with conjunctions as tests contains three decision nodes: $(A \wedge B)$, $(C \wedge D)$, and $(E \wedge F)$. Its CNF representation, $(A \vee C \vee E) \wedge (A \vee C \vee F) \wedge (A \vee D \vee E) \wedge (A \vee D \vee F) \wedge (B \vee C \vee E) \wedge (B \vee C \vee F) \wedge (B \vee D \vee E) \wedge (B \vee D \vee F)$, is very complex, but its negation has a concise CNF form (the conjunction of three disjunctions). Therefore, its tree representation with disjunctions as tests can also contain only three decision nodes: $(\bar{A} \vee \bar{B})$, $(\bar{C} \vee \bar{D})$, and $(\bar{E} \vee \bar{F})$. The tree corresponds to $(\bar{A} \vee \bar{B}) \wedge (\bar{C} \vee \bar{D}) \wedge (\bar{E} \vee \bar{F})$. The arguments above are for logical concepts with binary features and two classes. If learning tasks have more than two classes and nominal features, the situation is complex. Although this is worth exploring further, we will show later that the performance of conjunctions and disjunctions, as new features for decision tree learning, is quite similar in the real-world domains studied. These domains involve nominal features and multiple classes.

In this study, we consider X-of-N representations as nominal features (Zheng, 1995a), although they can also be treated as continuous-valued features (Zheng, 1995b). X-of-N can directly represent all the following types of concept: conjunction (with or without internal disjunction), disjunction (with or without internal disjunction), at-least M-of-N, at-most M-of-N, exactly M-of-N, even parity and odd parity, and possible combinations of the above six types of concept (Zheng, 1995a). Figure 15.1 gives an example of the X-of-N representation's usage in the real-world. It shows a part of a decision tree in the Cleveland Heart Disease domain. The root is an X-of-N representation. SubtreeA and subtreeB contain some other X-of-N representations.

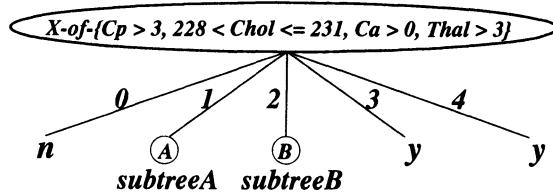


Figure 15.1 A tree with nominal X-of-Ns in the Cleveland Heart Disease domain

One disadvantage of X-of-N representations as nominal features is the fragmentation problem. When X-of-N representations with large values for N are used as tests for decision trees, they quickly split training sets into a large number of small subsets. This makes it harder to find further good tests for subtrees. However, this situation only occurs when training sets are relatively small and target concepts are complex in the sense that a few X-of-N representations are not enough. If target concepts are not complex or huge data sets are available, it is not a problem.

Two possible approaches to alleviating the fragmentation problem are subsetting (Quinlan, 1993; Zheng, 1995a) and subranging. For an X-of-N representation, subsetting merges its values into several subsets by carrying out search.

When building a decision tree, one branch is created for each subset, rather than each value of the X-of-N. Subranging is similar to subsetting except that it takes account of the ordering information of the values of an X-of-N representation. It merges only adjacent values of an X-of-N.

15.3 A SINGLE ALGORITHM FOR CREATING FOUR DIFFERENT TYPES OF NEW FEATURE

We use the most well-known algorithm C4.5 (Quinlan, 1993) for learning decision trees. The search method employed for constructing new features is the simple but widely used greedy search. Here, we do not intend to show that greedy search is the most appropriate search method for constructing new features. It is adopted because it works reasonably well, and does not favor any specific new feature type.

The algorithm borrows the control structure, tree learning method, new feature construction strategy, new feature search method, evaluation function, and so on from the XofN algorithm (Zheng, 1995a). If the option is set to construct X-of-N representations, this algorithm behaves exactly the same as the XofN algorithm. Choosing the XofN algorithm as the basis of the algorithm for the comparison study in this chapter is based on the soundness and effectiveness of XofN which are established in Zheng (1995a). It has been shown that XofN performs reasonably well compared with several other constructive and selective induction algorithms in a variety of artificial domains and real-world domains under consideration in terms of higher prediction accuracy and lower theory complexity.

The details of the XofN algorithm can be found in Zheng (1995a). Here, we only describe the main idea and a few specific points about the learning algorithm used in this chapter. This algorithm is referred to as CONJ, DISJ, MofN, and XofN when constructing conjunctive, disjunctive, M-of-N, and X-of-N representations respectively.

The test at each decision node of a tree is derived from either a primitive feature or a new feature (one of the four types). The algorithm adopts the data-driven strategy for constructing new features. When building decision trees, it generates at each decision node one new feature based on primitive features using the local training set. If the new feature constructed at a node is better than all primitive features and all previously created new features, the algorithm uses it as the test for the node; otherwise discards it and uses the best of the primitive features and previously constructed new features.

To construct a new feature, the algorithm performs a greedy search in the instance space defined by primitive features. The search starts from an empty set. At each search step, it either adds one possible feature-value pair or deletes one possible feature-value pair. When searching for a new feature at each decision node, XofN treats a set of feature-value pairs as a nominal X-of-N feature; CONJ treats it as a conjunction of the feature-value pairs, while DISJ treats it as a disjunction. To find an M-of-N representation, for each set of

feature-value pairs, MoFN uses the number that gives an M-of-N with the highest value of the evaluation function as M .

For comparing and selecting new features, a combination of information gain ratio and coding cost is used as the evaluation function (Zheng, 1995a). The idea is to accept a new feature with a higher gain ratio if its coding cost is not higher, or a new feature with the same gain ratio but with a lower coding cost. The search stops when the maximum possible size of representations is reached or no better new feature has been found in five consecutive search steps. Note that this default setting is arbitrary. There is no special reason for selecting “five”.

To deal with primitive continuous-valued features, the algorithm uses a simple discretization method, although some better, but more complex, methods such as Catlett (1991), Fayyad and Irani (1993), and Van de Merckt (1993) can be used. It runs C4.5 once using primitive features to obtain cut points on continuous-valued features. These continuous-valued features, then, are discretized using the cut points. For solving the fragmentation problem of nominal X-of-N representations, subranging is used. After an X-of-N is created, the best subranges of its values are found and used to form a test. The XoFN algorithm with subranging is denoted as XoFN(r). Note that subranging does not affect the behavior of CONJ, DISJ, and MoFN because they construct binary features.

15.4 EXPERIMENTS

We use experiments in a set of artificial and real-world domains to evaluate the performance of C4.5, CONJ, DISJ, MoFN, XoFN, and XoFN(r). We focus on prediction accuracy and theory complexity. The theory complexity, here, is the modified tree size (Zheng, 1995a). It is the sum of the sizes of all the nodes, including leaves, of the tree rather than the number of decision nodes or all the nodes of the tree. The size of a leaf is 1. The size of a decision node is 1 for a univariate tree, and is the number of attribute-value pairs, or conditions, in the test of the node for a multivariate tree. Univariate trees refers to decision trees with tests in their decision nodes containing single primitive attributes. Selective induction algorithms generate this type of tree. Multivariate trees have decision nodes with tests containing more than one primitive attribute. Constructive induction algorithms create this kind of tree. The modified tree size is a fair measure of the theory complexity when comparisons involve both selective and constructive induction algorithms. Computational requirements will be briefly addressed at the end of this section.

15.4.1 Experimental Domains and Methods

Table 15.1 summarizes the characteristics of a set of artificial logical domains (Pagallo, 1990). Each CNF is the dual concept of the corresponding DNF. This is done by changing AND to OR, and changing OR to AND. For CNF concepts, columns “No. of terms” and “Term length” give the number of disjunctions and disjunction length respectively. These domains cover a variety

of well-studied artificial logical concepts in the machine learning community: randomly generated boolean concepts including DNF and CNF concepts, multiplexor concepts, parity concepts, and majority concepts. We use the same experimental method given in Pagallo (1990). For each experiment, a training set and a test set are independently drawn from a uniform distribution. The size of the test sets is 2000. The sizes of training sets, given in Table 15.1, are the result of a VC dimension analysis (Vapnik and Chervonenkis, 1971) for finding the number of examples which would suffice for an ideal learning algorithm to create a consistent hypothesis with an error rate less than 10% on any test set (Pagallo, 1990). For each concept, the experiments are repeated ten times using different training and test sets.

Table 15.1 Characteristics of artificial logical domains

Domain name	Description	No. of terms	Term length			No. of att.	Training set size
			Min.	Max.	Average		
DNF1	random DNF	9	5	6	5.8	80	3292
DNF2	random DNF	8	4	7	4.5	40	2185
DNF3	random DNF	6	4	7	5.5	32	1650
DNF4	random DNF	10	3	5	4.1	64	2640
CNF1	random CNF	9	5	6	5.8	80	3292
CNF2	random CNF	8	4	7	4.5	40	2185
CNF3	random CNF	6	4	7	5.5	32	1650
CNF4	random CNF	10	3	5	4.1	64	2640
MX6	6-multiplexor	4	3	3	3.0	16	720
MX11	11-multiplexor	8	4	4	4.0	32	1600
Parity4	4-parity	8	4	4	4.0	16	1280
Parity5	5-parity	16	5	5	5.0	32	4000
Maj11	11-majority	462	6	6	6.0	32	3000
Maj13	13-majority	1716	7	7	7.0	64	3000

In addition to the fourteen artificial logical domains, ten domains from the UCI repository of machine learning databases (Merz and Murphy, 1997) are used, in which M-of-N-like concepts are expected to be found (Spackman, 1988). They consist of five medical domains (Cleveland Heart Disease, Hepatitis, Liver Disorders, Pima Indians Diabetes, Wisconsin Breast Cancer), one molecular biology domain (Promoters), three linguistics domains (Nettalk(Phoneme), Nettalk(Stress), Nettalk(Letter)), and one game domain (Tic-Tac-Toe). For the three Nettalk domains, we use the 1000 most common English words containing 5438 letters. In each of these ten domains, a 10-fold cross-validation (Breiman, Friedman, Olshen, and Stone, 1984) is conducted on the entire data set, and all algorithms are run on the same partitions. Table 15.2 gives a brief summary of the ten domains, including the dataset size, the number of binary (B), nominal (N), continuous-valued (C) attributes, the total (T) number of

Table 15.2 Description of ten domains from UCI

Domain name	Size	No. of Attributes				No. of Classes	Default Accuracy (%)
		B	N	C	T		
Cleveland Heart Disease	303	0	0	13	13	2	54.1
Hepatitis	155	13	0	6	19	2	79.4
Liver Disorders	345	0	0	6	6	2	58.0
Diabetes	768	0	0	8	8	2	65.1
Wisconsin Breast Cancer	699	0	0	9	9	2	65.5
Promoters	106	0	57	0	57	2	50.0
Nettalk(Phoneme)	5438	0	7	0	7	52	18.7
Nettalk(Stress)	5438	0	7	0	7	5	40.1
Nettalk(Letter)	5438	0	7	0	7	163	11.2
Tic-Tac-Toe	958	0	9	0	9	2	65.3

attributes, the number of classes, and the default accuracy (the relative frequency of the most common class). The ten domains cover the spectrum of properties such as dataset size, attribute types and numbers, the number of different nominal attribute values, and the number of classes.

In all the experiments reported here, all the algorithms are run with their default option settings. To compare the accuracies of two algorithms, a two-tailed pairwise t-test is conducted. A difference is considered as significant if the significance level of the t-test is better than 0.05. In tables summarizing accuracies, boldface (italic) font indicates that an algorithm is significantly better (worse) than C4.5.

15.4.2 Results

Tables 15.3 and 15.4 show that all the problems except MX6 are hard for the selective induction algorithm C4.5. In these domains, all constructive induction algorithms achieve significant improvement over C4.5 on both accuracy and complexity except that XofN does not work well on the DNF, CNF, and MX11 concepts. In these domains, XofN suffers from the fragmentation problem. However, XofN with subranging learns very accurate and concise decision trees in these domains. In addition, CONJ and DISJ learn more accurate trees than C4.5 in the Maj11 and Maj13 domains, but the theory complexities of the trees learned by CONJ and DISJ in these two domains are similar to those learned by C4.5. It is even the case that DISJ generates significantly more complex trees than C4.5 in the Maj13 domain. The reason is that the DNF and CNF representations of Maj11 and Maj13 are very complex.

The performance of CONJ and DISJ is very similar in almost all these artificial domains, even in the DNF and CNF domains. This indicates that constructing conjunctions as new features has similar performance to construct-

Table 15.3 Accuracies (%) in the artificial logical domains

<i>Domain</i>	<i>C4.5</i>	<i>CONJ</i>	<i>DISJ</i>	<i>MofN</i>	<i>XofN</i>	<i>XofN(r)</i>
DNF1	87.2	96.7	100.0	100.0	86.1	97.9
DNF2	90.7	99.5	99.5	100.0	87.7	99.6
DNF3	93.8	99.8	99.8	100.0	93.1	99.1
DNF4	74.0	99.2	99.9	100.0	74.5	99.5
CNF1	86.9	99.8	98.2	100.0	84.6	99.5
CNF2	90.6	99.4	99.3	99.6	87.1	99.4
CNF3	93.3	99.7	99.8	100.0	93.2	99.6
CNF4	72.9	99.7	99.5	100.0	73.5	100.0
MX6	100.0	100.0	100.0	100.0	100.0	100.0
MX11	97.2	100.0	100.0	100.0	96.6	100.0
Parity4	67.5	100.0	100.0	100.0	100.0	100.0
Parity5	52.2	79.4	78.1	95.5	100.0	100.0
Maj11	82.9	92.0	92.1	100.0	100.0	100.0
Maj13	76.3	83.4	84.5	100.0	100.0	100.0

ing disjunctions as new features for decision tree learning on logical concepts. DNF1 and CNF1 are two exceptions. CONJ obtains a significantly lower accuracy and a significantly higher theory complexity than DISJ in DNF1, and vice versa in CNF1. It is worth mentioning that no accuracy difference between CONJ and DISJ is significant in the other domains.

However, this is not because conjunction and disjunction, as new features for tree learning, have different expressive power in the DNF1 and CNF1 domains. Actually, the differences come from the implementation of the search used by CONJ and DISJ. CONJ and DISJ greedily search for a set of feature-value pairs to form a conjunction and a disjunction respectively. The search starts from a feature-value pair consisting of the first feature and its first value of a learning problem. In the experiment, the feature values “0, 1” are presented for every feature of each artificial domain. DNF1 has a term $x_5x_{28}x_{38}x_{72}x_{74}x_{76}$, for example. It is easier for DISJ to find $((x_5 = 0) \vee (x_{28} = 0) \vee (x_{38} = 0) \vee (x_{72} = 0) \vee (x_{74} = 0) \vee (x_{76} = 0))$ than for CONJ to find $((x_5 = 1) \wedge (x_{28} = 1) \wedge (x_{38} = 1) \wedge (x_{72} = 1) \wedge (x_{74} = 1) \wedge (x_{76} = 1))$ because $(x_i = 0)$ appears earlier than $(x_i = 1)$. These two new features are equal for building tree representations of DNF1. The difference between them when they are used as tests at a decision node is that the two branches of the node are reversed. The situations are the same for other terms of DNF1, and the reverse for disjunctions of CNF1. Therefore, DISJ is more likely to build a good tree than CONJ in the DNF1 domain, and vice versa in the CNF1 domain, but only as a side-effect of the implementation of the heuristic search strategy employed. Note that the differences in accuracy between CONJ and DISJ in DNF1 and CNF1 are not

Table 15.4 Theory complexities in the artificial logical domains

<i>Domain</i>	<i>C4.5</i>	<i>CONJ</i>	<i>DISJ</i>	<i>MofN</i>	<i>XofN</i>	<i>XofN(r)</i>
DNF1	263.0	128.8	65.9	62.3	300.6	116.7
DNF2	202.0	63.6	69.6	50.1	234.0	49.2
DNF3	101.8	49.5	42.5	40.0	88.0	48.4
DNF4	525.0	79.4	59.4	52.2	593.2	56.8
CNF1	271.2	69.7	82.9	62.0	314.2	61.1
CNF2	192.8	56.7	51.0	48.8	227.0	42.1
CNF3	94.4	46.2	43.3	40.0	90.0	36.1
CNF4	532.8	64.1	70.3	52.2	607.8	52.7
MX6	48.6	19.4	19.2	17.5	60.6	25.2
MX11	168.6	51.8	46.0	51.5	243.6	54.4
Parity4	238.4	41.2	38.5	27.1	9.0	9.0
Parity5	1339.4	900.8	903.6	268.2	13.4	14.4
Maj11	461.6	447.1	454.0	13.0	23.0	13.0
Maj13	527.4	553.0	559.8	15.0	27.0	15.0

large, as *CONJ* also finds most correct terms for DNF1, while *DISJ* also finds most correct disjunctions for CNF1.

To confirm this point, *CONJ* and *DISJ* were run again in the DNF1 and CNF1 domains, but feature values were presented as “1, 0” for each feature. All other factors remained the same, including training and test sets. In DNF1, *CONJ* achieves a 99.9% accuracy and 64.5 complexity, while *DISJ* achieves a 98.3% accuracy and 106.7 complexity. In CNF1, the accuracy and complexity are 99.4% and 81.2 for *CONJ*, while they are 100.0% and 63.9 for *DISJ*. Reversing the order in which the feature-value pairs ($x_i = 0$) and ($x_i = 1$) are considered during search reverses the ordering of the accuracies and the ordering of the theory complexities obtained by *CONJ* and *DISJ* on each of DNF1 and CNF1. These results illustrate, from another aspect, that constructing new features in the form of conjunctions and constructing new features in the form of disjunctions are very similar for learning trees on logical concepts.

Now, we consider *MofN* and *XofN*. *MofN* performs as well as *CONJ* and *DISJ* in the DNF, CNF, and multiplexor domains, and significantly better in the majority and parity domains in terms of accuracy and theory complexity. *XofN* performs significantly better than *CONJ* and *DISJ* in the majority and parity domains, significantly better than *MofN* in the parity domains, and as well as *CONJ*, *DISJ*, and *MofN* in all other domains. Note that, as mentioned above, *XofN* needs subranging to solve the DNF, CNF, and multiplexor problems. Although *CONJ*, *DISJ*, and *MofN* learn correct trees for Parity4, the trees are more complex than those learned by *XofN*. Compared with *XofN*, *MofN* learns less accurate and more complex trees for Parity5. In summary, M-of-N representations show performance advantages over conjunctions and dis-

Table 15.5 Accuracies (%) in the ten UCI domains

<i>Domain</i>	<i>C4.5</i>	<i>CONJ</i>	<i>DISJ</i>	<i>MofN</i>	<i>XofN</i>	<i>XofN(r)</i>
Cleveland Heart Disease	73.3	75.9	75.9	76.3	79.8	78.2
Hepatitis	78.2	78.1	79.4	79.4	79.4	79.4
Liver Disorders	62.1	62.9	65.2	61.8	70.1	66.1
Diabetes	71.5	73.0	72.0	74.1	70.8	71.7
Wisconsin Breast Cancer	94.8	95.3	94.6	95.6	94.9	95.1
Promoters	76.3	83.8	82.8	82.7	88.5	83.5
Nettalk(Phoneme)	81.1	83.5	83.6	84.0	83.9	84.1
Nettalk(Stress)	82.7	86.3	86.7	86.9	87.6	87.6
Nettalk(Letter)	73.7	77.4	76.8	77.0	76.9	77.0
Tic-Tac-Toe	84.7	98.1	98.1	98.0	98.4	97.9

junctions for constructive tree learning in these artificial domains. Moreover, X-of-N representations show performance advantages over these three types of representation.

Tables 15.5 and 15.6 summarize the accuracies and theory complexities of the algorithms in the ten UCI domains. There is no significant accuracy difference between XofN and XofN(r) in all these domains. The differences among the theory complexities of CONJ, DISJ, MofN, XofN and XofN(r) are not large. One algorithm may build simpler trees than other algorithms in some domains, but it may learn more complex trees than other algorithms in some other domains. No algorithm dominates others across the ten domains in terms of lower theory complexity. Therefore, the analysis below focuses on accuracies of CONJ, DISJ, MofN, and XofN.

Compared with C4.5, each of CONJ, DISJ, and MofN achieves significant improvements in four domains, while XofN achieves significant improvements in seven domains. No method gets a significantly worse result than C4.5 in any of these domains. This suggests that XofN performs somewhat better than CONJ, DISJ, and MofN.

When comparing CONJ, DISJ, and MofN, no accuracy difference between any two of them is significant in any domain. CONJ is more accurate than DISJ in four domains, less in four domains, and equally in two domains. This result shows that CONJ and DISJ have similar performance in the real-world domains where nominal features and multiple classes are involved. MofN has higher accuracies than CONJ in six domains and lower in four domains. It has higher accuracies than DISJ in six domains, lower in three domains, and equal in one domain. This suggests that MofN has a slight advantage over CONJ and DISJ.

Compared with CONJ, DISJ, and MofN, XofN has higher accuracies than CONJ in seven domains and lower in three domains; XofN has higher accuracies than DISJ in eight domains, lower in one domain, and equal in one domain;

Table 15.6 Theory complexities in the ten UCI domains

Domain	C4.5	CONJ	DISJ	MofN	XofN	XofN(r)
Cleveland Heart Disease	49.8	45.1	44.9	38.2	41.1	42.0
Hepatitis	13.6	12.1	12.1	12.2	13.4	13.3
Liver Disorders	79.4	87.7	85.7	81.3	89.1	95.9
Diabetes	128.8	141.0	145.3	140.7	153.6	170.8
Wisconsin Breast Cancer	20.6	25.1	18.6	16.3	26.3	22.1
Promoters	22.6	16.2	16.2	12.6	13.9	14.3
Nettalk(Phoneme)	2339.2	1219.9	1173.7	1572.0	1506.0	1483.9
Nettalk(Stress)	2077.3	810.7	811.8	802.7	739.6	816.0
Nettalk(Letter)	3394.9	1883.9	1872.8	2300.9	2242.4	2229.0
Tic-Tac-Toe	128.5	44.2	54.5	50.4	42.8	79.6

XofN has higher accuracies than MofN in five domains, lower in four domains, and equal in one domain. XofN is significantly more accurate than CONJ in the Cleveland Heart Disease and Nettalk(Stress) domains, significantly more accurate than DISJ in the Cleveland Heart Disease domain, and significantly more accurate than MofN in the Liver Disorders domain. No other difference is significant. These results show that constructive decision tree learning can benefit from the stronger representation power of X-of-Ns in terms of higher prediction accuracy in some real-world domains, compared with conjunctive, disjunctive, and M-of-N representations.

Advantages of MofN over CONJ and DISJ, and advantages of XofN over CONJ, DISJ, and MofN in the UCI domains are not as obvious as those in the artificial logical domains. One reason might be that these UCI domains do not contain many X-of-N and M-of-N concepts, although we do find some such as the one shown in Figure 15.1. Another example in the Tic-Tac-Toe domain is X-of-{top-left-square=x, middle-middle-square=x, bottom-right-square=x}. If its value is 0, the class is “-”. If its value is 3, the class is “+”. If its value is 1 or 2, the class needs to be further decided. There are three other possible X-of-N concepts.¹ All other appropriate new features are conjunctions. Therefore, XofN only performs slightly better than CONJ, DISJ, and MofN. Moreover, there is no noise or missing values in these artificial logical domains, while most of these UCI domains have noise and some have missing values. How noise and missing values affect XofN and the others is an issue for future investigation.

15.4.3 Computational Requirements

The search space for creating an X-of-N representation is the same as that for generating a conjunction or a disjunction. The search space for constructing an M-of-N is larger, since to create an M-of-N representation, we must search

for a set of attribute-value pairs as well as a value M . When the number of attribute-value pairs in the set is large, searching for a good value M can be expensive. However, this does not mean that for a given learning task, CONJ, DISJ, and XofN need the same execution time, while MofN needs more time. The reason is that the sizes of learned trees can be very different. CONJ, DISJ, MofN, and XofN construct one new feature at each decision node, so the sizes of unpruned trees can seriously affect their execution times.

For example, the DNF2 concept is suited to CONJ and DISJ. In this domain, CONJ and DISJ use 221.0 and 205.2 seconds on a DEC AXP 3000/500 workstation respectively on an average of ten trials. MofN also solves this problem, but it spends much more time, 3971.1 seconds. XofN uses 1260.6 seconds, less than that for MofN. The Maj11 concept is most appropriate for MofN. It uses, on average, 34.5 seconds to solve this problem (with correct trees). Both CONJ and DISJ fail to learn correct trees in this domain, and use much more time than MofN – 225.1 and 260.0 seconds for CONJ and DISJ respectively. XofN generates correct trees for Maj11 as well, but it needs only 19.2 seconds, even less than the time required by MofN. The Parity4 concept is most suited to XofN, although all CONJ, DISJ, MofN, and XofN build correct trees in this domain. On average, they spend 42.2, 37.6, 53.6, and 8.6 seconds respectively. XofN requires the least time and MofN the most. To have a reference point, we give the execution times for the selective induction algorithm C4.5. They are 1.3, 1.7, and 0.6 seconds respectively in the DNF2, Maj11, and Parity4 domains. Compared with C4.5, all of CONJ, DISJ, MofN, and XofN are much slower, but they learn much better decision trees in terms of both higher prediction accuracy and lower theory complexity.

15.5 DISCUSSION

The experimental results in the previous section have shown that conjunctions and disjunctions, as new features for decision tree learning, have similar performance for logical concepts including DNF and CNF concepts. The only difference between the CONJ and DISJ algorithms is that CONJ creates conjunctions while DISJ creates disjunctions. Their performance is very similar in the artificial domains, even in the real-world domains involving nominal features and multiple classes.

However, the constructive decision tree learning algorithm FRINGE (Pagallo, 1990) performs quite well on DNF concepts, but it does not work well on CNF concepts. FRINGE creates a new feature by using the conjunction of two conditions near the leaf for each positive path in a tree. For solving CNF problems, Pagallo (1990) proposes the DUAL FRINGE algorithm. It constructs a new feature using the disjunction of negations of two conditions near the leaf for each negative path in a tree. DUAL FRINGE does work well on CNF concepts, but not on DNF concepts.

From this, it might be thought that constructing conjunctions has an advantage over constructing disjunctions for constructive tree learning on DNF concepts, and vice versa on CNF concepts. However, this is not true. The

reason why FRINGE is biased towards DNF concepts and DUAL FRINGE is biased towards CNF concepts is not that FRINGE constructs conjunctions and DUAL FRINGE constructs disjunctions as new features. The real reason is that FRINGE uses positive paths to create conjunctions and DUAL FRINGE uses negative paths to generate disjunctions. If FRINGE is extended to construct conjunctions from both positive and negative paths, like SYMFRINGE (Yang, Rendell, and Blix, 1991), it should be able to attack both DNF and CNF problems. There is no essential difference between conjunctions and disjunctions as constructive operators for decision tree learning at least in domains with binary features and two classes.

15.6 RELATED WORK

Almost all papers that describe constructive induction algorithms compare a few new feature construction methods. For example, Matheus (1989) compares four different constructive operand selection strategies with conjunction as the constructive operator. During the comparisons, the same constructive induction framework for tree learning is used. He concludes that in general cases, the fringe strategy appears to be the best of the four strategies owing to its production of relatively accurate and concise trees. Pagallo (1990) compares the FRINGE algorithm with the DUAL FRINGE and SYMMETRIC FRINGE algorithms on two CNF concepts. The three algorithms share the same overall control structure for learning trees and constructing new features. However, both the constructive operators and condition selection strategies are different for FRINGE and DUAL FRINGE. SYMMETRIC FRINGE is their combination. It is shown that FRINGE is good at learning DNF concepts; DUAL FRINGE is good at learning CNF concepts; and SYMMETRIC FRINGE can handle both DNF and CNF problems. Yang et al. (1991) compare three methods of constructing new features that are used by the FRINGE family of algorithms within the same tree learning and constructive induction framework. They find that the limited construction of conjunction and disjunction performs better than the construction of pure conjunction and the unrestricted construction of conjunction and disjunction on some small DNF and CNF concepts.

Many papers compare totally different constructive induction and selective induction algorithms. For example, Ragavan and Rendell (1993) compare their constructive tree learning algorithm LFC with ID3, FRINGE, GREEDY3, BP, and MARS. LFC achieves quite high accuracy in a few artificial and real-world domains. Bloedorn et al. (1993) compare their multistrategy constructive induction algorithm AQ17-MCI with other AQ algorithms including AQ14, AQ17-HCI, AQ17-NT, and AQ17-DCI in the “Noisy and Irrelevant Monks2” domain. AQ17-MCI reveals its advantage over the others in this domain. Wnek and Michalski (1994) compare AQ17-HCI with five selective and constructive induction algorithms, and show the advantage of AQ17-HCI in four artificial domains. Zheng (1995a) compares the XofN algorithm with different learning algorithms including ID2-of-3 (Murphy and Pazzani, 1991).

If different algorithms are used for comparing different new feature construction methods, it is not very clear where the performance differences come from. Therefore, in this chapter, we focused on comparing the effects, on constructive induction, of constructing different types of new feature using a single learning algorithm. The objective of doing so is to diminish effects that are created by differences of other factors which might affect the performance of constructive induction.

15.7 CONCLUSIONS AND FUTURE WORK

From analysis, we know that conjunctive and disjunctive representations have very similar performance on logical concepts for decision tree learning, even on CNF concepts that are intuitively thought to be suited only to disjunctive representations and DNF concepts that are intuitively thought to be suited only to conjunctive representations. The M-of-N representation is more representationally powerful than conjunction and disjunction, while the X-of-N representation is the most powerful representation among the four types.

To empirically compare the effects of constructing four different types of new feature (conjunctive, disjunctive, M-of-N, and X-of-N representations) on constructive decision tree learning, a single constructive tree learning algorithm has been developed. It constructs one of these four types of new feature: conjunctive, disjunctive, M-of-N, or X-of-N representations when the corresponding option setting is specified. All other factors that may affect the performance of an algorithm such as the tree learning method, new feature construction strategy, new feature search method, and evaluation function are exactly the same. This makes it possible to focus on comparisons of the performance of different types of new feature without the effects produced by differences of other factors.

The experimental evaluation results in the artificial domains clearly show that for decision tree learning, constructing conjunctions and constructing disjunctions have similar performance, constructing M-of-Ns has advantages over constructing conjunctions and disjunctions, while constructing X-of-Ns has advantages over constructing all these three types in terms of higher prediction accuracy and lower theory complexity. This suggests that the order of the representation power of conjunctive, disjunctive, M-of-N, and X-of-N representations is reflected in their performance in decision tree learning. The experimental results in the real-world domains also support these conclusions although this support is not as strong as that from the artificial domains.

The effects of noise and missing values on the four types of new feature remain to be investigated. Some other issues for future research are: to compare these four types of new features using more complex M-of-N concepts and more real-world domains as well as to try more sophisticated search methods and discretization methods for primitive continuous-valued features.

Acknowledgments

The research reported here was done when the author was at the University of Sydney. The research was partially supported by an ARC grant (to J.R. Quinlan) and by a research agreement with Digital Equipment Corporation at the University of Sydney. The chapter was composed at Deakin University. Some experimental results were published in a short paper entitled "Effects of Different Types of New Attribute on Constructive Induction" in *Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence*, Los Alamitos, CA: IEEE Computer Society Press, 1996. The author appreciates the advice and suggestions J.R. Quinlan has given. Discussions with P. Langley led to the idea of comparing the effects of different types of new feature on constructive induction by using a single learning algorithm. Many thanks to J.R. Quinlan for providing C4.5, also to D. Aha, J. Catlett, N. Indurkhy, P. Langley, K.M. Ting, T. Van de Merckt, A. Varšek, and G.I. Webb for their helpful comments that improved earlier drafts of this chapter.

Notes

1. They correspond to three positions on a diagonal of the Tic-Tac-Toe board.

References

- Bloedorn, E., Michalski, R.S., and Wnek, J. (1993). Multistrategy constructive induction: AQ17-MCI. *Proceedings of the Second International Workshop on Multistrategy Learning*, pages 188-203.
- Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, R.A. (1984). *Classification And Regression Trees*. Wadsworth, Belmont, CA.
- Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. *Proceedings of the Fifth European Working Session on Learning*, pages 164-178. Springer-Verlag, Berlin.
- Fayyad, U.M. and Irani, K.B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022-1027. Morgan Kaufmann, San Mateo, CA.
- Matheus, C.J. (1989). *Feature Construction: An Analytic Framework and an Application to Decision Trees*, Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.
- Merz, C.J. and Murphy, P.M. (1997). UCI Repository of Machine Learning Databases [<http://www.ics.uci.edu/~mlearn/MLRepository.html>]. Department of Information and Computer Science, University of California, Irvine, CA.
- Michalski, R.S. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:349-361.
- Murphy, P.M. and Pazzani, M.J. (1991). ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees. *Proceedings of the Eighth International Workshop on Machine Learning*, pages 183-187. Morgan Kaufmann, San Mateo, CA.

- Pagallo, G. (1990). *Adaptive Decision Tree Algorithms for Learning from Examples*, Ph.D. Thesis, University of California at Santa Cruz, Santa Cruz, CA.
- Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.
- Ragavan, H. and Rendell, L. (1993). Lookahead feature construction for learning hard concepts. *Proceedings of the Tenth International Conference on Machine Learning*, pages 252-259. Morgan Kaufmann, San Mateo, CA.
- Spackman, K.A. (1988). Learning categorical decision criteria in biomedical domains. *Proceedings of the Fifth International Conference on Machine Learning*, pages 36-46. Morgan Kaufmann, San Mateo, CA.
- Van de Merckt, T. (1993). Decision trees in numerical attribute spaces. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1016-1021. Morgan Kaufmann, San Mateo, CA.
- Vapnik, V.N. and Chervonenkis, A.Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theor. Probab. Appl.*, 16:264-280.
- Wnek, J. and Michalski, R.S. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14:139-168.
- Yang, D., Rendell, L., and Blix, G. (1991). A scheme for feature construction and a comparison of empirical methods. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 699-704. Morgan Kaufmann, San Mateo, CA.
- Zheng, Z. (1995a). Constructing nominal X-of-N attributes. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1064-1070. Morgan Kaufmann, San Mateo, CA.
- Zheng, Z. (1995b). Continuous-valued X-of-N attributes versus nominal X-of-N attributes for constructive induction: A case study. *Proceedings of the Fourth International Conference for Young Computer Scientists*, pages 566-573. Peking University Press, Beijing.

16 CONSTRUCTIVE INDUCTION: COVERING ATTRIBUTE SPECTRUM

Yuh-Jyh Hu

Information and Computer Science Department
University of California, Irvine
Irvine, California 92697
yhu@ics.uci.edu

Abstract: Inductive algorithms rely strongly on their representational biases. Representational inadequacy can be mitigated by constructive induction. This chapter introduces several important issues for constructive induction and describes a new multi-strategy constructive induction algorithm (GALA2.0) which is independent of the learning algorithm. Unlike most previous research on constructive induction, our methods are designed as a preprocessing step before standard machine learning algorithms are applied. We present the results which demonstrate the effectiveness of GALA2.0 on real domains for two different learners: C4.5 and backpropagation.

16.1 INTRODUCTION

The ability of an inductive learning algorithm to find an accurate concept description depends heavily upon the representation (Flann & Dietterich, 1986). Concept learners typically make strong assumptions about the vocabulary used to represent these examples. The vocabulary of features determines not only the form and size of the final concept learned, but also the speed of the convergence (Fawcett & Utgoff, 1992). Learning algorithms that consider a single attribute at a time may overlook the significance of combining features. For example, C4.5 (Quinlan, 1993) splits on the test of a single attribute while constructing a decision tree, and CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991) specializes the complexes in Star by conjoining a single literal or dropping a disjunctive element in its selector. Such algorithms suffer from the standard problem of any hill-climbing search: the best local decision may not lead to the best global result.

One approach to mitigate these problems is to construct new attributes. The need for useful new attributes has been suggested by many researchers (Matheus, 1991; Aha, 1991; Ragavan *et. al.*, 1993). Constructing new attributes by hand is often difficult (Quinlan, 1983). The goal of constructive induction is to automatically transform the original representation space into a new one where the regularity is more apparent (Dietterich & Michalski, 1981), thus yielding improved classification accuracy. If we organize all types of new attributes in some order, we could build an attribute spectrum. Thus, constructive induction could be considered learning the *right* portion of the attribute spectrum. Several machine learning algorithms perform feature construction by extending greedy, hill-climbing strategies, including FRINGE (Pagallo, 1989), GREEDY3 (Pagallo & Haussler, 1990), DCFringe (Yang *et. al.*, 1991), CITRE (Matheus & Rendell, 1989). These algorithms construct new features by finding local patterns in the previously constructed decision tree. These new attributes are added to the original attributes, the learning algorithm is called again, and the process continues until no new useful attributes can be found. However, such greedy feature construction algorithms often show their improvement only on particular artificial concepts (Yang *et. al.*, 1991). FRINGE-like algorithms are limited by the quality of the original decision tree from which they construct new features. Some reports show that if the basis for the construction of the original tree is a greedy, hill-climbing strategy, accuracies remain low (Rendell & Ragavan, 1993).

An alternative approach is to use some form of lookahead search. Exhaustive lookahead algorithms like IDX (Norton, 1989) can improve decision tree inductive learners without constructing new attributes, but the lookahead is computationally prohibited, except in simple domains. LFC (Ragavan & Rendell, 1993; Ragavan *et. al.*, 1993) mitigate this problem by using directed lookahead and by caching features. However, LFC's quality measure(i.e., blurring measure) limits this approach. MRP (Perez & Rendell, 1995) is demonstrated to learn several complex relations, the meanings of its extensional representation is implicit in the data, and usually difficult to interpret.

In this chapter, we first highlight some important issues for constructive induction, then introduce a new feature construction algorithm which addresses both the bias of the initial representation and the search complexity. And finally, we suggest future research directions in constructive induction.

16.2 IMPORTANT ISSUES FOR CONSTRUCTIVE INDUCTION

There are several important issues about constructive induction. These issues determine its applicability and behavior. We detail these issues in the following subsections.

16.2.1 Interleaving vs. Preprocessing

By interleaving we mean that the selective induction process and constructive induction process are intertwined into a single algorithm. Most current constructive induction algorithms fall into this category.

Interleaving strategy may better tie together selective induction and constructive induction, but on the other hand, constructive induction is also strongly affected by selective induction process bias, and thus limits its applicability. By keeping these processes separate, the constructive induction algorithm can be used as a preprocessor to any standard selective induction algorithm. With the preprocessing model one can also test the appropriateness of the learned features over a wider class of learning algorithms.

16.2.2 Hypothesis-driven vs. Data-driven

Hypothesis-driven approaches construct new attributes based on the hypotheses generated previously. They begin by constructing a hypothesis such as a decision tree, and then examine the hypothesis to generate new attributes. After new attributes are added to the primitive ones, instances are redescribed by the derived attributes and the primitive ones. The same process is repeated until no good new attributes could be found. Hypothesis-driven approaches are a two-edged sword. They have the advantage of previous knowledge/hypotheses and the disadvantage of being strongly dependent on the quality of previous knowledge.

On the other hand, data-driven approaches cannot benefit from previous hypotheses, but can avoid the strong dependency. Members of this family construct and evaluate new features through analyzing the training data directly.

16.2.3 Absolute Measures vs. Relative Measures

Absolute measures evaluate the quality of an attribute over the training examples without regard to how the attribute was constructed. Examples of such measures include entropy variants (e.g., blurring), information gain, gain ratio, error-rate, etc. While it is important that a new attribute performs well, it is also important that it has significant improvement over its parents. We refer to this as a relative measure since it depends on the quality improvement relative to a new attribute over its parents.

The combination of absolute and relative measures not only helps constrain the search space, but also more importantly provides the guidance for constructive induction algorithms to search through the right portions of the space.

16.2.4 Operators

The simplest operators for constructing new attributes are boolean operators (i.e. “and”, “or” and “not”), which are what most constructive induction algorithms use. Though Boolean operators are simple, by composite Boolean operations, ideally we are able to learn any subconcept in a target concept that could be represented in disjunctive (or conjunctive) normal form. How-

ever, due to the intractable search space and the limitations of greedy feature construction and selection, as the target concepts become more complicated, Boolean operators may not be sufficient. To alleviate the limitations of Boolean operators, other operators that consider more complex attribute interactions are also incorporated. For example, m-of-n operator, projection operator and functional operators are applied in some systems.

16.2.5 Attribute Spectrum

If we represent the new attributes in disjunctive normal form, we could build a spectrum of the attributes based on the number of terms involved. At one end are the ones with relatively few terms; at the other, those with many terms. There is no uniform correlation between the number of terms and the capability of the algorithms, and there is no universal algorithm to cover the whole spectrum. The process bias of a learning algorithm determines the type of new attributes it could learn. In other words, the portion of attribute spectrum covered by a learning algorithm is biased by its learning process.

16.2.6 Background Knowledge

By applying construction operators, we are able to generate new attributes from the primitive ones. However, in unfamiliar problems, the search space of new attributes is sometimes huge and beyond the reach of the operators. In this case, we need the background knowledge as an external bias to direct the constructive induction.

Most work in knowledge-guided constructive induction has assumed approximately complete background knowledge; however, in practice domain knowledge is usually partial. For domains that are barely understood, even fragmentary knowledge could be used to narrow the search space of constructed features.

16.2.7 Single Strategy vs. Multiple Strategies

The types of derived new attributes depend on the learning strategy applied. The process bias of a learning strategy affects the formation of new attributes. For example, new attributes learned by a hypothesis-driven learning strategy are biased by the iterative learning process.

In terms of the attribute spectrum introduced earlier, the larger portion of the spectrum a bias could cover, the better the bias is. However, there is no universal learning process bias that could cover the whole spectrum. Thus a single learning strategy only covers part of the attribute spectrum. As the target concept becomes more complicated, it is no longer adequate. In order to cover a larger portion of the attribute spectrum, multiple learning strategies are needed. With multiple process biases working together, multiple strategies cover a wider portion than a single strategy.

16.3 GALA2.0: COMBINING NEURAL NETWORK WITH ITERATIVE ATTRIBUTE CONSTRUCTION

16.3.1 Motivation

One important issue in constructive induction is the types of new attributes constructed that form an attribute spectrum as mentioned earlier. At one end are the attributes with relatively few terms, denoted *oligoterm attributes*; at the other end, those with many terms, denoted *polyterm attributes*. And there are many more in between. There is no uniform correlation between the number of terms and the capability of the algorithms, and there is no universal algorithm to cover the whole spectrum. One possible approach to covering more of the spectrum is to combine different construction strategies (Hu, 1997).

GALA1.0, a preprocessor approach to constructive induction, which applies relative measures and iterative attribute combination techniques is capable of generating complex but relatively smaller Boolean combinations as new attributes (Hu & Kibler, 1996), thus it could be used to construct those new attributes that belong to one end of the spectrum, i.e., *oligoterm attributes*.

However, as any iterative attribute construction algorithm, it has an inherent drawback, i.e., as the concept becomes larger in the number of terms in disjunctive normal form, they fail to find useful new attributes. Larger numbers of attribute combinations incur more attribute interaction that hinders the performance of the algorithms (Hu & Kibler, 1996; Perez & Rendell, 1996). Therefore, we need another construction strategy to cover the other end of the spectrum. Because it is impossible to build a learner to learn any arbitrarily long and complex concept, we currently concentrate on the prototypical structures, such as the m-of-n rules, majority, etc, which often exist in the real domains like medical diagnoses (Spackman, 1988).

Given a target concept containing prototypical structures and other complex but relatively shorter Boolean combinations, our strategy is to first extract the prototypical structure and represent it as a single new attribute. Second, with the new prototypical attributes added to the primitives, we then apply GALA1.0 to extract the remaining Boolean combinations in the target concept. Since the new attributes generated encapsulates the complexity of the prototypical structures, GALA1.0 is able to learn the remaining Boolean combinations. The question left is how we extract the prototypical structures.

The multilayer perceptron is probably the most studied neural network technique (Hertz *et. al.*, 1991; Kung, 1993). Each hidden unit draws a simple decision surface in the input space, and then the output units combine these individual regions to form a final region corresponding to the target concept. As the decision region of each hidden unit is formulated in a linear function, some of the hidden units are likely to converge to meaningful linear threshold functions, and could be used as the basis of prototypical structures such as the m-of-n rules, majority, etc. The linear threshold functions could then be transformed into new attributes to represent prototypical structures. Therefore, the

neural network is our answer to the question left above. The general framework of our approach is described in Figure 16.1.

```

Given : a set of training examples E
        a set of primitive attributes P
Return: a set of new attributes NEW

Procedure GALA2.0(E,P)
    H = hidden units from ANN(E)
    Transform H to a set of prototypical linear features N
    Represent E in P+N as E'
    NEW={}
    NEW = N + GALA1.0(P+N,E',threshold,NEW)
    Return (NEW)

```

Figure 16.1 General Framework of GALA2.0

16.3.2 Generating Prototypical Linear Features by Neural Network

In this subsection, we discuss how we construct prototypical linear features from a neural network.

The new attributes directly derived from the hidden units of the neural network are difficult to interpret. From the point of view of a prototypical structure, there are two causes of the incomprehensibility. The first is the irrelevant primitive attributes. In the feed-forward neural network architecture, each hidden unit is fully connected to all the input units (i.e., primitive attributes). After the network converges, the links between the hidden units and the irrelevant input units may still carry non-zero weights. These irrelevant non-zero weights, though usually small, make the representation difficult to interpret. It would be more human-interpretable if all the irrelevant weights are zero. The other cause is the weights themselves. The weights are unlikely to be integers after the network converges. It is difficult to infer the prototypical structures from a set of non-integer weights. For example, a 3-of-5 rule is better represented by $X_1 + X_2 + X_3 + X_4 + X_5 \geq 3$ than $1.02X_1 + 0.98X_2 + 1.05X_3 + 0.97X_4 + 0.98X_5 \geq 2.93$.

To overcome the above problems, we propose a two-stage weight normalization method. The irrelevant weights are usually significantly smaller than others in magnitude and often with an opposite sign. In the first stage, we use the mean of the weight magnitude as the criterion. The weights below the mean and of the opposite sign are set to zero. The reason why we consider the

sign is that in case all input units are relevant and of the same sign, we will not discard any relevant weight simply because it is below the mean. Along with the change of weights, we use the difference between the hidden unit value before and after the weight change to adjust the threshold of the hidden unit. In the second stage, we normalize the weights with the mean of the remaining non-zero weights and apply the round-off function to suppress other insignificant weights to zero. Thus we generate comprehensible prototypical linear features. More details could be found in Figure 16.2.

Given : a set of weights and thresholds W,
 a set of training examples E
 Return: a set of normalized weights and thresholds

```

Procedure normalize(W,E)
  Let W' be W
  Let m be the mean of the absolute values
    of the weights in W'
  Let s be the sign (i.e., + or -) of the weight with
    max absolute value
  For each weight w' in W'
    if (abs(w') < m) and (w' with opposite sign to s)
      w' = 0
    sumdiff = 0
    For each training example e in E
      sumdiff = sumdiff+(eW-eW')
    diff = sumdiff/|E|
    Let m be the mean of the absolute values
      of the non-zero weights in W'
    For each weight w' and threshold t' in W'
      w' = round(w/m);
      t' = round((t'+diff)/m);
  Return (W')
```

Figure 16.2 Weights and Thresholds Normalization

The set of weights and thresholds are difficult to interpret, but after we *normalize* them the concept represented by these weights and thresholds is easier to understand. We use $(\sum_{i=1}^8 x_i \geq 5) + x_1 \bar{x}_3 \bar{x}_5 x_7 + \bar{x}_2 \bar{x}_4 \bar{x}_6 x_8$, a Boolean function with total 12 attributes, as an example to illustrate the idea. Using 5% of the total 4096 examples as the training set, we trained a neural net with the backpropagation algorithm. The neural net has 12 input units, 6 hidden units and one output unit.

The weights and thresholds before and after normalization are shown in Figure 16.3. To save the space, we only show hidden unit 1,3, and 5. In Figure 16.3, each hidden unit is described by 12 weights followed by a threshold. Before normalization, the weights and thresholds we learned are not understandable even though the fifth hidden unit approximately converged to $\sum_{i=1}^8 x_i \geq 5$. However, after the normalization, the fifth hidden unit is easily interpreted as a 5-of-8 rule.

The weights of the hidden units before normalization :

Hidden Unit 1: 4.43948, -0.90867, -1.85273, -0.09461, -2.56182,
0.44131, 3.80673, 0.88315, 1.21998, -0.02205,
0.03032, -1.60084, **threshold = -7.18849**

Hidden Unit 3: 1.49163, -0.95159, -0.32819, -0.27491, 1.43506,
-0.56259, 1.37591, 0.87493, 0.52599, -0.69051,
1.22448, 2.60315, **threshold = -2.63325**

Hidden Unit 5: 3.13984, 3.82628, 3.94562, 3.75279, 3.37203,
3.75229, 3.07394, 3.34460, -0.24504, 0.06971,
-0.26181, -0.70992, **threshold = -15.6727**

The weights of the hidden units after normalization :

Hidden Unit 1: 2, 0, -1, 0, -1, 0, 1, 0, 0, 0, 0, -1, **threshold = -2**
Hidden Unit 3: 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 2, **threshold = -3**
Hidden Unit 5: 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, **threshold = -5**

Figure 16.3 Results of Normalization

16.3.3 Generating Oligoterm Attributes by GALA1.0

The idea of GALA1.0 (Generation of Attributes for Learning Algorithms) is to consider those constructed features which have high relative and absolute gain ratio. This will be defined more precisely. The algorithm has three basic steps. The general flow of the algorithm is generate-and-test, but it is complicated since testing is interlaced with generation. The overall control flow is given in Figure 16.4. For each partition of the data, only one new attribute is added to the original set of primitives. Partitioning is stopped when the set is homogeneous or below a fixed size, currently set at 10.

Booleanize. Suppose f is a feature and v is any value in its range. We define two boolean attributes, $Pos(f, v)$ and $Neg(f, v)$ as follows:

$$Pos(f, v) \equiv \begin{cases} f = v & \text{if } f \text{ is a nominal or boolean attribute} \\ f > v & \text{if } f \text{ is a continuous attribute} \end{cases}$$

$$Neg(f, v) \equiv \begin{cases} f \neq v & \text{if } f \text{ is a nominal or boolean attribute} \\ f \leq v & \text{if } f \text{ is a continuous attribute} \end{cases}$$

The idea is to transform each real or nominal attribute to a single boolean attribute by choosing a single value from the range. The algorithm is more precisely defined in Figure 16.5. This process takes $O(AVE)$ time where A is the number of attributes, V is the maximum number of attribute-values, and E is the number of examples. The net effect is that there will be two boolean attributes associated with each primitive attribute.

Given: Primitive attributes P, training examples E, threshold,
cycle limit c and new attributes NEW
(NEW is empty when GALA invoked the first time)

Return: a set of new attributes NEW

```
Procedure GALA1.0(P,E,threshold,c,NEW)
  If (size(E) > threshold) and (E is not all of same class)
    Then Set Bool to Boolean attributes from Booleanize(P,E)
        Set Pool to attributes from Generate(Bool,E,c)
        Set Best to attribute in Pool with highest gain ratio
          (if more than one, pick one of smallest size)
        Add Best to NEW
        Split on Best
        N = empty set
        For each outcome, Si, of Split on Best
          Ei = examples with outcome Si on split
          NEWi = GALA1.0(P,Ei,threshold,c,NEW)
          N = union of N and NEWi
        NEW = union of NEW and N
        Return NEW
  Else Return empty set
```

Figure 16.4 GALA1.0

Generation. Conceptually, GALA1.0 uses only two operators, conjunction and negation, to construct new boolean attributes from old ones. Repetition of these operators yields the possibility of generating any boolean feature. However only attributes with high heuristic value will be kept. Figure 16.6 describes the iterative and interlaced generate and test procedure. If in each cycle we only keep B best new attributes (i.e., beam size is B), the procedure takes

$O(cB^2E)$ time where c is the pre-determined cycle limit, B is the beam size, and E is the number of examples.

Given: Attributes P and examples E .

Return: set of candidate boolean attributes.

Procedure Booleanize (P, E)

 Set Bool to empty.

 For each attribute f in P , find the v

 such that $\text{Pos}(f, v)$ has highest gain ratio on E .

 Add $\text{Pos}(f, v)$ and $\text{Neg}(f, v)$ to Bool.

 Return Bool

Figure 16.5 Transforming real and nominal attributes to boolean attributes

Heuristic Gain Filtering. In general, if A is an attribute we define $GR(A)$ as the gain ratio of A (Quinlan, 1993). If a new attribute A is the conjunction of attributes A_1 and A_2 , then we define two relative gain ratios associated with A as:

$$UPPER - RGR(A) = \max\left\{\frac{GR(A) - GR(A_1)}{GR(A)}, \frac{GR(A) - GR(A_2)}{GR(A)}\right\}.$$

$$LOWER - RGR(A) = \min\left\{\frac{GR(A) - GR(A_1)}{GR(A)}, \frac{GR(A) - GR(A_2)}{GR(A)}\right\}.$$

We consider the relative gain ratio only when the conjunction has a better gain ratio than each of its parents. Consequently this measure ranges from 0 to 1 and is a measure of the synergy of the conjunction over the value of the individual attributes. To consider every new attribute during feature construction is computational impractical. Coupling relative gain ratio with gain ratio constrains the search space without overlooking many useful attributes. We define $\text{mean}(GR(S))$ as the average absolute gain ratio of each attribute in S . We also define the mean relative gain ratios (i.e., $\text{mean}(UPPER-RGR(S))$ and $\text{mean}(LOWER-RGR(S))$) over a set S of attributes similarly. We use these measures to define the GainFilter in Figure 16.7.

16.3.4 Experimental Results

There are two objectives of the experiment. Firstly, we want to test whether GALA2.0 can improve the performance of other standard inductive learners by adding new attributes. Secondly, we want to justify that our new approach is competitive compared with current constructive induction algorithms.

Given: a set of boolean attributes P
 a set of training examples E
 cycle limit C

Return: new boolean attributes

Procedure Generate(P,E,C)

 Let Pool be P

 Repeat

 For each conjunction of Pos(f,v) or Neg(f,v)
 with Pos(g,w) or Neg(g,w).

 If conjunction passes GainFilter, add it to Pool

 Until no new attributes are found or reach cycle limit C

 Return Pool

Figure 16.6 Attribute Generation

Given: a set of new Attributes N

Return: new attributes with high GR and RGR

Procedure GainFilter(N)

 Set M to those attributes in N whose gain ratio is
 better than mean(GR(N)).

 Set M' to those attributes in M whose UPPER-RGR is
 better than mean(UPPER-RGR(N)) or LOWER-RGR is
 better than mean(LOWER-RGR(N))

 Return M'.

Figure 16.7 Filtering by mean absolute and relative gain ratio

We used C4.5 (Quinlan, 1993), which is a representative decision tree algorithm, and the neural net with backprop as the standard inductive learners in our experiment. For comparison with other feature learning programs, we

used LFC when it was applicable. LFC, based on the interleaving model, combines its generation of new attributes with its concept learning program. We chose LFC because it has been reported that it outperforms many current constructive induction algorithms (Ragavan & Rendell, 1993). To keep a fair comparison, we did not consider any constructive induction algorithm that incorporates background knowledge such as MIRO (Drastal *et. al.*, 1989), or anyone that uses extensional representation like MRP (Perez & Rendell, 1995). We tested our new approach on twelve real domains in UCI machine learning databases. The test domains are summarized in Table 16.1.

In each case, two thirds of the examples form the training set and the remaining examples form the test set. The results are shown in Table 16.2. Each result is averaged over 20 runs. The second through fourth column is the accuracy of the neural net, C4.5 and LFC respectively. Because LFC is a two-class learning algorithm, we did not apply it to multi-class domains. The fifth and six columns present the new accuracies of the neural net and C4.5 after adding the new attributes generated by GALA2.0. Significant improvement (0.05 level in one tailed paired t-test) by GALA2.0 over the inductive learner is marked with “1”, and significant difference of accuracy between GALA2.0 and LFC is marked with “2”.

The experimental results showed that GALA2.0 significantly improves the inductive learners on most of the domains, and is very competitive at least compared with one current constructive induction algorithm. In no case was the introduction of the generated attributes significantly harmful to the learning algorithm.

Table 16.1 Domain Summary

Domain	No. of Ex	No. of Real/Nominal Attr	No. of Classes
Breast Cancer	699	9/0	2
Heart Disease	303	5/8	2
Liver Disorder	345	7/0	2
Ionosphere	351	34/0	2
Lymphography	148	3/15	4
Promoter	106	0/57	2
Echocardiogram	131	6/1	2
Hayes Roth	160	0/4	3
Glass	214	9/0	7
Iris	150	4/0	3
Solar Flare	323	9/3	6
Pima Diabetes	768	8/0	2

In all experiments, the parameters of C4.5 were set to default values to keep the consistency. For the neural net, the learning rate and the momentum were both set to 0.1, and the attribute values were normalized between 0 and 1. We

also adopted the suggested heuristics for a fully connected network structure: initial weights selected at random and a single hidden layer whose number of nodes was half the total number of input and output nodes (Ragavan & Rendell, 1993; Rumelhart *et. al.*, 1986).

Table 16.2 Results of Real Domains

Domain	NN	C4.5	LFC	NN+GALA2.0	C4.5+GALA2.0
Breast Cancer	92.7	94.5	94.2	94.8 ¹	95.5 ¹²
Heart Disease	74.3	72.3	75.2	76.9 ¹²	76.7 ¹²
Liver Disorder	66.1	62.1	62.4	68.1 ¹²	65.6 ¹²
Ionosphere	87.3	90.1	90.3	90.6 ¹	92.3 ¹²
Lymphography	87.1	76.0	N/A	87.3	78.1 ¹
Promoter	71.9	73.9	75.1	78.2 ¹²	82.0 ¹²
Echocardiogram	64.6	60.0	63.1	68.4 ¹²	62.9 ¹
Hayes Roth	70.2	74.9	N/A	80.9 ¹	81.8 ¹
Glass	60.9	63.8	N/A	62.1	62.6
Iris	95.6	94.1	N/A	95.6	94.4
Solar Flare	64.9	69.5	N/A	65.9	70.2
Pima Diabetes	67.1	71.6	71.1	67.9	72.0

16.4 DISCUSSION

Research on constructive induction has been carried on for almost two decades since it was first introduced. In spite of the different design philosophies and system implementation details, they all have the same goals, to improve learning by providing better representations. After two decades of studies, how can constructive induction go any further on the same track, and what else can be done ? In this section, we discuss possible research directions of constructive induction; however, we do *not* imply or suggest which directions are more important or urgent than others. The objective of this discussion is mainly to provide some insights into constructive induction, and to keep the research in this field moving forward.

16.4.1 The Need for Comprehensible Derived Attributes

Most of the efforts of constructive induction have been directed to improved classification accuracy. However, the comprehensibility of new attributes could also be important for at least two reasons: (1) comprehensible new attributes are good because they further understanding of the domain; (2) comprehensible intermediate concepts may help the learner learn better. Recently, machine learning techniques have been successfully applied to knowledge discovery and data mining. Besides the concern of efficiency in the data mining community,

comprehensibility is also an important issue. People not only want to know whether the data contain valuable information, but also what the information is. Constructive induction techniques that are capable of generating useful comprehensible new features would contribute to answering the question.

16.4.2 The Need for New Strategies and Background Knowledge

Based on the attribute spectrum mentioned earlier, most of the existing systems could only generate oligoterm attributes. As concepts become more complicated where many terms involved, they all fail to construct useful new attributes. Though some algorithms are designed to tackle this problem, yet they are still restricted by their limited search methods and evaluation measures. We need new learning strategies and attribute quality measures to cover a larger portion of the attribute spectrum. One plausible approach is to combine multiple learning strategies and apply relative measures. In addition, to further constrain the search space, background knowledge has to be better employed. Besides complete background knowledge, systems could also make the most use of fragmentary knowledge to guide the selection of construction operators and derived attributes.

16.4.3 The Need for Better Understanding of Human Behavior

The original motivation of constructive induction is to ease humans from the burden of designing representations. We rely on machine learning techniques to automatically construct useful new representations. However, little work has been devoted to studying how human experts come up with good representations. More specifically, we know little about how human experts relate primitive features to higher-level abstract intermediate concepts in real domains. It would be useful if we could better learn about how human efforts are involved in the process of developing good representations. Incorporating their philosophies and methodologies into constructive induction will be one more step ahead in this research area. Not only the search strategy and construction operators in constructive induction could be much improved, but background knowledge could also be better exploited.

16.4.4 The Need for Across-Domain Learning

The standard machine learning methodology is to learn a single concept at a time. Within this paradigm, constructive induction uses training examples to learn a better representation and thus could improve the performance. By then the learning is over. When faced with a different problem/concept, constructive induction repeat the same learning process without taking into account those representations learned before. Unlike machine learning, people learn fast because they bring good representations to the problem, representations they learned on previous problems even if they are different problems. The main difference is that people learn from a series of problems instead of a sin-

gle problem. Different problems may have different solutions, but they often share the same useful representations. Based on this new paradigm, constructive induction is expected to generate a good representation across multiple domains it learned before. Within this new learning paradigm, more complicated transfer and transformation of representations occur across a series of problems/concepts; consequently, a single learning event has been upgraded to continual learning.

References

- Aha, D. (1991). Incremental Constructive Induction: An Instanced-Based Approach. in *Proceedings of the 8th Machine Learning Workshop*, pages 117-121.
- Clark, P. and Boswell, R. (1991). Rule Induction with CN2: some recent improvements. European Working Session on Learning, pages 151-161.
- Clark, P. and Niblett, T. (1989) The CN2 Induction Algorithm. *Machine Learning* 3, pages 261-283.
- Dietterich, T.G. and Michalski, R.S. (1981). Inductive Learning of Structural Description : Evaluation Criteria and Comparative Review of Selected Methods. *Artificial Intelligence* 16 (3), pages 257-294.
- Drastal, G., Czako, G. and Raatz, S. (1989). Induction on an Abstraction Space : A Form of Construction Induction. in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 708-712.
- Fawcett, T.E. and Utgoff, P.E. (1992). Automatic Feature Generation for Problem Solving Systems. in *Proceedings of the 9th International Workshop on Machine Learning*, pages 144-153.
- Flann, N.S. and Dietterich, T.G. (1986). Selecting Appropriate Representations for Learning from Examples. in *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 460-466.
- Hertz, J., Krogh, A. and Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley.
- Hu, Y. and Kibler, D. (1996). Generation of Attributes for Learning Algorithms. in *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 806-811.
- Hu, Y. (1997). Learning Different Types of New Attributes by Combining the Neural Network and Iterative Attribute Construction. in *Proceedings of the 9th European Conference on Machine Learning*, pages 124-137.
- Kung, S.Y. (1993). *Digital Neural Networks*. Prentice Hall.
- Matheus, C.J. and Rendell, L.A. (1989). Constructive Induction on Decision Trees. in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 645-650.
- Matheus, C.J. (1991). The Need for Constructive Induction. in *Proceedings of the 8th Machine Learning Workshop*, pages 173-177.
- Norton, S.W. (1989). Generating better Decision Trees. in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 800-805.

- Pagallo, G. (1989). Learning DNF by Decision Trees. in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*.
- Pagallo, G. and Haussler, D. (1990). Boolean Feature Discovery in Empirical Learning. *Machine Learning* 5, pages 71-99.
- Perez, E. and Rendell, L. (1995). Using Multidimensional Projection to Find Relations. in *Proceeding of the 12th Machine Learning Conference*, pages 447-455.
- Perez, E. and Rendell, L. (1996). Learning Despite Concept Variation by Finding Structure in Attribute-based Date. in *Proceeding of the 13th Machine Learning Conference*, pages 391-399.
- Quinlan, J.R. (1983). Learning efficient classification procedures and their application to chess end games. in Michalski et. al.'s *Machine Learning : An artificial intelligence approach*. (Eds.)
- Quinlan, J.R. (1993). C4.5 : Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA.
- Ragavan, H., Rendell, L., Shaw, M., and Tessmer, A. (1993). Complex Concept Acquisition through Directed Search and Feature Caching. in *Proceeding of the 13th International Joint Conference on Artificial Intelligence*, pages 946-958.
- Ragavan, H. and Rendell, L. (1993). Lookahead Feature Construction for Learning Hard Concepts. in *Proceeding of the 10th Machine Learning Conference*, pages 252-259.
- Rendell L.A. and Ragavan, H. (1993). Improving the Design of Induction Methods by Analyzing Algorithm Functionality and Data-Based Concept Complexity. in *Proceeding of the 13th International Joint Conference on Artificial Intelligence*, pages 952-958.
- Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning Internal Representations by Error Propagation. in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol 1, pages 318-362.
- Spackman, K. (1988). Learning Categorical Decision Criteria in Biomedical Domains. in *Proceeding of the 5th International Workshop on Machine Learning*, pages 36-46.
- Yang, D-S., Rendell, L.A., and Blix, G. (1991). A Scheme for Feature Construction and a Comparison of Empirical Methods. in *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 699-704.

17 FEATURE CONSTRUCTION USING FRAGMENTARY KNOWLEDGE

Steve Donoho¹ and Larry Rendell²

¹SRA International, Inc.
4300 Fair Lakes Court
Fairfax, VA 22033
donohos@iisd..sra..com

²Computer Science Dept.
University of Illinois
405 North Mathews Avenue
Urbana, IL 61801
rendell@cs..uiuc..edu

Abstract: Domain knowledge can range from a nearly complete theory to a sparse set of ill-defined hunches. Most work in knowledge-guided constructive induction has used nearly complete theories to guide feature construction. This chapter focuses on fragmentary knowledge: knowledge that is too incomplete to yield specific new features directly. One important way knowledge, fragmentary knowledge can guide feature construction is by providing prior evidence for or against groups of constructed features, thus limiting the space of features that must be searched. Several common types of fragmentary knowledge can be used in this manner, as particularly shown with two case studies in bankruptcy prediction and turfgrass management.

17.1 INTRODUCTION

In unfamiliar problems, the performance of a standard learning program is often poor because the training examples' natural representation is a bad match for the algorithm's inductive bias. A common approach is therefore to retain the original learning program but redescribe the examples using a new set of features derived from the original features. When automated, this process is called *feature construction* (the entire transformation and learning procedure is

called *constructive induction* (Michalski, 1983)). Since the combinatorics are often prohibitive, domain knowledge is used to constrain the search whenever feasible.

In practice, domain knowledge is usually partial. It has varying degrees of incompleteness and segmentation, ranging from a nearly complete theory to a sparse set of ill-defined hunches. Most systems that incorporate knowledge into constructive induction use almost complete theories, and immediately *deduce* new features from the original features (Drastal and Raatz, 1989). Once features are deduced, they can be used either directly or as starting points for searching for new features. Some interesting variations on this theme include sophisticated selection of features from different levels of a theory (Ortega and Fisher, 1995), applying a theory revision system instead of a standard induction program (Mooney and Ourston, 1991), measuring the extent to which a theory is matched (Donoho, 1996), and feature generation for state-space search (Fawcett and Utgoff, 1992).

Less work has been done in situations where only fragmentary knowledge is available. Fragmentary knowledge is too incomplete to yield specific new features. Fragmentary knowledge falls too far short of being a complete theory to deduce anything from it. But it can be used to narrow the search for constructed features. For example, in tic-tac-toe Matheus (1989) considered only constructed features whose component features were all X's or all O's. Aronis and Provost (1994) used inheritance hierarchies to narrow the search for feature relationships. Hirsh and Japkowicz (1994) used a deductive theory to specify new features but then incorporated fragments of knowledge to guide a search branching out from them.

This chapter explores several types of fragmentary knowledge and how it can be incorporated into feature construction. This study is part of our larger effort to understand the relationship between constructive induction and the entire spectrum of knowledge ranging from almost complete theories to highly fragmentary knowledge (Donoho, 1996). Section 17.2 discusses various examples of fragmentary knowledge, as we attempt to find underlying similarities and group them into *types* of knowledge useful for feature construction. Section 17.3 explores a number of issues involved in incorporating fragmentary knowledge into feature construction. Sections 17.4 and 17.5 use our recent experiments in bankruptcy prediction and turfgrass management to demonstrate some of these issues. Section 17.6 argues for the utility of our approach and discusses some broader issues.

17.2 TYPES OF FRAGMENTARY KNOWLEDGE

Our ultimate goal is to improve feature construction by understanding whether there are *types* of knowledge that are particularly relevant to specific construction techniques, and if so, how best to exploit such an understanding. We begin with an attempt to categorize fragmentary knowledge that arises in various domains found in the literature and elsewhere.

Relevance knowledge. Relevance knowledge indicates that a group of features is relevant to a particular goal class or intermediate concept. Features that are all related to the same intermediate concept are likely to be related to each other. For example, in the Horse Colic domain there are 27 elementary features. Of these, the three *nasogastric reflux*, *feces examination*, and *abdomen examination* relate to the intermediate concept *obstruction*. These features are perhaps more likely to combine in some useful way than an arbitrary group, and thus their combinations should be explored first. Similarly, in determining the insurance risk of various automobiles, the features *body style*, *make*, *number of doors* are relevant to the intermediate concept *family car*. The features *horsepower*, *body style*, *make*, and *aspiration* relate to the intermediate concept *sporty car*.

Relevance knowledge does not tell exactly how features should be combined, but it does show that some combinations are more favorable than others. Constructed features composed primarily of features all relevant to a single concept are more likely to be useful than features composed of an arbitrary group.

Support knowledge. Support knowledge falls between deductive theories and relevance knowledge. It tells more than which features should be considered together but does not completely specify how to combine them. Support knowledge takes the following form: a feature is known to support a particular goal class or intermediate concept when it has a particular value. For example, in auto insurance risk the feature *auto style* is not only relevant to the concept *family car*, it also supports *family car* when it has the value *station wagon*.

In the turfgrass domain, the goal is to predict whether a particular type of Kentucky Bluegrass will grow well under the given conditions. Features include the soil's *pH*, *phosphorus*, *potassium*, *nitrogen fertilization*, etc. as well as climate conditions such as *precipitation* and *temperature*. A *pH* between 6.0 and 7.0 is known to foster growth. Soil texture between *loam* and *sandy clay loam*, phosphorus above 75 lbs/acre, and a mowing height between 0.75 and 2.5 inches are also known in general to lead to good Bluegrass growth. As with relevance knowledge, features that support the same concept are likely to interact. In addition, they are more likely to interact in ways that combine their support. The conjunction ($pH = 6.5$) & ($phos = 80$) is more likely to be useful than ($pH = 6.5$) & not ($phos = 80$) because both ($pH = 6.5$) and ($phos = 80$) support good growth.

Correlation knowledge. This type of knowledge is a continuous version of support knowledge. Correlation knowledge relates numeric features to goal classes or intermediate concepts. It indicates that as a particular feature value rises (or falls), the associated concept is more likely to be true. For example, in predicting corporate bankruptcy, a company's *cash* level is positively correlated with the company's success. All else equal, the more cash a company has, the more likely it is to succeed. A company's *long term debt* is negatively correlated with the company's success. All else equal, the higher a company's *long term*

debt, the less likely it is to succeed. This knowledge limits the way these features can be meaningfully combined. If *cash* is subtracted from *long term debt* the result is meaningful; it is negatively correlated to success. If *cash* is added to *long term debt* the result is neither positively nor negatively correlated; it is unrelated to success and is therefore less likely to be useful. Constructed features that are correlated to some important concept should be searched before those correlated to nothing important.

Contingency knowledge. Contingency knowledge gives information that the effect of one feature is contingent upon the value of another feature. It augments other types of knowledge such as relevance knowledge, support knowledge, or correlation knowledge. For example, in weather prediction *cloudiness* is negatively correlated with temperature contingent upon it being daytime. In the nighttime, cloudiness is actually positively correlated with temperature because the clouds hold in the heat. In determining insurance risk, *horsepower* is relevant to the concept *sporty car* contingent upon the feature *body style*. If the body style is *station wagon* or *minivan*, *horsepower* becomes irrelevant. Contingency knowledge tightens the constraints of the knowledge it augments. Constructed features with the contingent condition true are more likely to be useful than those without it.

This knowledge may also manifest itself as knowledge of the absence of contingencies. If there are not contingencies between two features, then the two will behave independently. Changing the value of one will not affect the meaning of the other. For example, in credit screening, many features including income, employment stability, savings, debt, credit history, etc. all contribute to the final decision, but the value of any one feature does not affect the contribution of the others. Situations such as these are conducive to M-of-N and weighted sum features because a change in the value of one feature does not influence the support of the others.

Normalization knowledge. Normalization knowledge involves a single feature correlated to multiple intermediate concepts — some relevant, others irrelevant. The goal of normalization knowledge is to normalize a feature to eliminate correlations to irrelevant intermediate concepts. For example, when predicting corporate bankruptcy, a company's *net income* is correlated to both its success and size. If IBM has a net income of \$1,000,000, this is very bad. If Bob's Laundromat has a net income of \$1,000,000, this is very good. It is desirable to normalize *net income* to eliminate correlation to size. It is also known that both *assets* and *sales* are measures of size; therefore, dividing by either of these or some additive combination of them would normalize *net income*. In the Boston housing price domain, the number of criminal acts committed in a town is correlated to both the town's safety and size. A number of measures of size such as population, physical size, and operating budget could be used to normalize this feature to measure safety.

Proximity knowledge. Objects that are physically close are more likely to interact than objects that aren't. Events that occur within a brief time period are more likely to be related than distant events. Proximity knowledge indicates that a group of features represent objects or events that are in close physical, temporal, or functional proximity.

For example, in domains involving DNA sequences, nucleotides that are close together are more likely to interact than distant nucleotides. In protein structure prediction, amino acids that are close in the peptide chain are more likely to interact. In the musk domain, the goal is to predict whether a molecule of a given shape has a musk smell or not. The features are distances measured from the center of the molecule to its surface along a number of rays extending from the center. Thus each feature represents the distance to a small patch of the molecule's surface. Physically close patches are more likely to interact to describe some important larger aspect of the molecule's surface. A similar phenomenon arises in medical diagnosis. A physician asks a series of questions and runs a series of tests on a patient, considering the results after each step. Questions and tests requested close in the series are more likely to be related. Once again, a structure/function model of a mechanical system shows how components are linked together. Features derived from closely linked components are more likely to interact than others. Proximity knowledge allows the feature construction system to project onto a subset of features and consider them in isolation. Constructed features whose components are all close in some way are more likely to be useful than a combination with arbitrary components.

Inheritance hierarchies. These show how features can be grouped according to their similarities. For example, in predicting a star's luminosity from its stellar spectrum the features are the presence or absence of spectral lines corresponding to substances in the star's atmosphere. Aronis and Provost (1994) use an inheritance hierarchy to group features with common properties: singly ionized, multiply ionized, neutral, molecules. The features *Fe*, *Ti*, and *Mg* are perhaps more likely to behave interchangeably because they all indicate the presence of a neutral element. Inheritance hierarchies have the same effect as proximity knowledge; they group features that are more likely to interact in some way.

Constructed features composed primarily of features that are closely related in the inheritance hierarchy should be favored over arbitrary combinations. Less complete inheritance hierarchies can also be helpful. In a vision problem such as image segmentation, the three features red intensity, green intensity, and blue intensity can be grouped as features measuring color intensity. In an auto insurance risk problem, the features city-mpg and highway-mpg can be grouped as fuel usage measurements, and auto length, width, and height can be grouped as size measurements.

Dimensional analysis. Knowledge of features' dimensions can make certain feature combinations illegal (Kokar, 1986). For example, in a mechanical

physics domain a feature with the units *meters* should not be added to a feature with the units *kilograms*. In an electrical domain, *volts* should not be added to *amps*. In a financial domain, *dollars* should not be added to *dollars/year*. Furthermore, some legal combinations should also be excluded because they have little qualitative meaning. For example, multiplying *dollars* by *dollars* gives *dollars*², but *dollars*² has little practical meaning.

17.3 INCORPORATING FRAGMENTARY KNOWLEDGE INTO SEARCH

Figure 17.1 shows where the types of knowledge discussed above fit into a larger spectrum. On the far left are complete, correct theories for which no induction is needed. Slightly to the right of this are theories that are incomplete or incorrect in some minor way. They may be overly general, have gaps at the top or middle, or may contain small errors, but they are complete enough that new features can be deduced from them. Further to the right lies the region of fragmentary knowledge. Fragmentary knowledge falls too short of a complete theory to deduce anything, but can be used to narrow the search for constructed features. My methods are based on viewing feature construction as search through the space of constructed features. Three ways knowledge can improve this search space are by changing its starting point, boundaries, and connectivity.

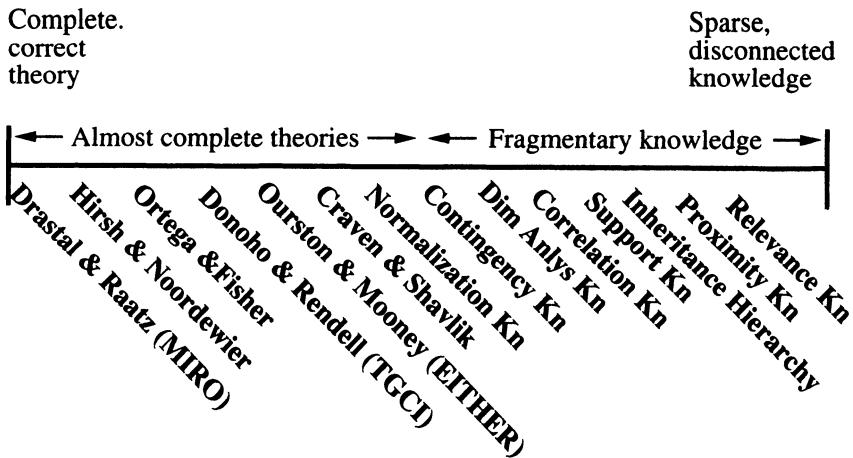


Figure 17.1 Knowledge can range from complete theories down to sparse sets of ill-defined hunches.

Starting Point. Where in the feature space should search begin? In the absence of knowledge, search usually begins with syntactically simple features in accordance with Occam's Razor. Some more complete types of knowledge,

though, point to specific features, and these can be starting points for search. These starting points are valuable because they may be located in regions of the feature space that would never be reached without knowledge. Points can be unreachable for a variety of reasons. In some cases they are syntactically complex and building them up from simple features would take many steps. In other cases they contain feature relationships such as XOR's that most heuristic searches overlook.

Boundaries. A second issue is what firm boundaries to set on search. In the absence of knowledge there may be no firm boundaries, or Occam's Razor may be applied to favor features with fewer components. By excluding regions of the feature space, knowledge allows limited computing resources to search areas of the feature space that would otherwise never be reached. One way knowledge does this is by giving prior evidence for or against groups of features. For example, dimensional analysis knowledge divides the feature space into two regions, one with legal unit combinations and one with illegal unit combinations, and the boundary between the two is clear. Other times there is a gradual change from high quality features to low. In these cases a quality threshold must be decided upon to set a boundary on search.

Connectivity. Thirdly, knowledge may change the connectivity of the space of constructed features. Connectivity determines the order in which the space will be traversed: search proceeds from one feature to those considered to be similar to it. The most common non-knowledge bias for determining connectivity is syntactic similarity. The boolean feature $A \wedge B \wedge C \wedge D$ is similar to $A \wedge B \wedge D$ and $A \wedge B \wedge C \wedge D \wedge G$ because the last two are formed by adding or dropping a single condition. This is based on the underlying semantics that adding a conjunct reduces the instances covered to a subset of those previously covered. Contingency knowledge may show that a partial match of a pattern is almost as good as a full match, and the feature $A \wedge B \wedge C \wedge D$ is similar to $(A \wedge B \wedge C) \vee (A \wedge B \wedge D) \vee (A \wedge C \wedge D) \vee (B \wedge C \wedge D)$. This effectively changes the connectivity of the space of constructed features.

The reliability of a piece of knowledge is frequently unknown until it is actually used. For example, a piece of knowledge may divide the feature space into multiple regions ranging from high probability of usefulness to low. Should the best area be searched extensively, or should mediocre areas be included using a less thorough search? This issue is really one of balancing the reliability of the knowledge with the reliability of the data. If the experimenter relies too much on knowledge, he may miss out on opportunities for the data to expose flaws in the knowledge. Conversely, though, relying too heavily on data may reduce accuracy if the knowledge is highly reliable. A systematic means of balancing knowledge and data is needed. Consider the following scenario as an illustration of this process. The knowledge in a particular domain includes a deductive theory and dimensional analysis knowledge. The dimensional analysis knowledge can be used to create a boundary between high quality and low

quality features, and the deductive theory provides a handful of starting points in the highly rated region. A simple first experiment is to use the starting points verbatim as new features. A standard induction algorithm is applied using this new representation, and the test accuracy is higher than the original representation but lower than desired. A second experiment conducts a breadth-first search from the starting points and returns a set of variations of the starting point features that are highly rated according to a training set of data. The accuracy of this representation is higher than the first experiment. A third experiment conducts a beam search from the starting points staying within the bounds set by the dimensional analysis knowledge. This representation produces accuracy slightly lower than the second experiment. We can conclude for this series of experiments that a good set of features lie close to the starting points but require a thorough search to be found. This process of iteratively loosening the assumptions made about knowledge is an effort to find a correct balance between knowledge and data while expending the least design and implementation effort.

17.4 BANKRUPTCY EXPERIMENTS

This section presents a case study in which we incorporate dimensional analysis knowledge, correlation knowledge, and normalization knowledge into constructive induction for bankruptcy prediction. In the bankruptcy domain each datum is a vector of features giving the financial position of a particular company for a given year. A vector is labeled “bankrupt” if the company filed for bankruptcy within 6 months after the financial information was reported; otherwise it is labeled “non-bankrupt.” The goal is to classify an unlabeled company given its financial position. The original features are taken from a company’s published balance sheet and statement of assets and liabilities; they include total sales, cost of operation, interest payments, cash, accounts receivable, other current assets, non-current assets, current liabilities, and long term debt.

Table 17.1 summarizes the knowledge available for this problem. One type of knowledge available is from dimensional analysis — the units of a new feature must make sense. It makes no sense to add a feature with the units *dollars* to a feature with the units *dollars/year*. In addition to ruling out illegal combinations, many legal but intuitively meaningless units can also be barred such a *dollars²/year*. Only features with the final units *dollars*, *dollars/year*, *years*, and *unitless* have nothing against them. For example the feature *CO – AR* has evidence against it because it is illegal to subtract *dollars* from *dollars/year*. The feature *CO × AR* also has evidence against it because *dollars²/year* has no intuitive meaning. The feature $\frac{AR}{CO}$ has the units *years* which is legal and meaningful.

A second type of available knowledge is correlation — features are positively or negatively correlated with the good of the corporation. *Total sales* is positively correlated: if everything else is held constant, the higher the total sales, the better off a corporation is. Likewise, *long term debt* is negatively correlated

Table 17.1 Bankruptcy domain knowledge: each feature's units and how each feature is correlated to the good of the company.

<i>Feature</i>	<i>Units</i>	<i>Qualitative Correlation</i>
total sales (TS)	\$/yr	positive
cost of operation (CO)	\$/yr	negative
interest payments (INTR)	\$/yr	negative
cash (CASH)	\$	positive
accounts receivable (AR)	\$	positive
other current assets (OCA)	\$	positive
non-current assets (NCA)	\$	positive
current liabilities (CL)	\$	negative
long term debt (LTD)	\$	negative

because, all else held constant, a higher long term debt is bad for the corporation. These correlations affect how the features can be combined. When *cash* is added to *accounts receivable*, the result is positively correlated because both its components are. When *total sales* is added to *long term debt*, the result is neither positively nor negatively correlated because its components are correlated differently. This knowledge provides evidence for feature combinations that are known to be somehow qualitatively correlated with the good of the company.

A third piece of knowledge is also available. It is a weak form of normalization knowledge — the values of the original features are not only correlated to the good of the company but are also correlated to other factors such as the company's size, industry, etc., although an exact model of these correlations is not available. Features that measure one aspect of a company against another aspect of the same company normalize the features by eliminating irrelevant correlations. This knowledge provides evidence for features that are ratios of one value to another. As a bias toward simplicity, we viewed this knowledge as supporting constructed features that are ratios of sums.

The above three pieces of knowledge divide the space of constructed features into many sections: features that have supporting evidence from the second and third pieces of knowledge and no evidence against them from the first, features with some evidence for and some against, features with no evidence for but none against, etc. A design decision must be made as to how much to rely on the knowledge as opposed to relying on the data and common heuristics for measuring feature worth with respect to the data. This will determine the technique selected for searching the space of constructed features.

As indicated in Section 17.2, experiments are often simpler when the knowledge is considered reliable. Therefore, our approach is to start with simple ex-

periments that rely heavily on the integrity of the knowledge, then, if necessary, conduct experiments that rely more on data and bias. For this reason we chose to examine only features that have support from the correlation knowledge and normalization knowledge and no evidence against them from the dimensional analysis knowledge. A preliminary scan of the space of remaining features revealed that exhaustive search was possible. This was implemented by creating all possible attribute sums (and differences) that did not violate the correlation and dimensional analysis knowledge and then creating ratios of all the binary combinations of these.

Table 17.2 Bankruptcy results with 99% confidence ranges.

<i>Feature Set</i>	<i>Accuracy</i>
Guessing majority class	71.0%
Original features	
Voting	68.6% \pm 1.3
C4.5	77.2% \pm 1.7
Fringe	73.2% \pm 1.6
LFC	74.4% \pm 2.9
Popular financial ratios	85.8% \pm 1.2
Automatically constructed features	
Automatically selected	86.2% \pm 0.7
Hand-selected	87.2% \pm 0.3

To test the algorithm outlined above, a dataset of 200 corporations was available; 58 bankrupt, 142 non-bankrupt. The constructed features were evaluated and ordered by their individual information gain with respect to a set of training examples. These training data were then redescribed using the constructed features and input to the induction program. A voting induction program was used in which each feature is individually used to predict an example's class, and the class receiving the most votes is selected. The set of voting features are selected greedily from the set of available features: the feature that gives the largest performance improvement is iteratively selected until no more features improve performance. The same voting induction program was also used with the original features and with 16 standard ratios from the financial literature. The results of these experiments are summarized in Table 17.2. Each result is the average of 10 different 10-fold cross-validation trials.

The accuracy using the automatically constructed features was much higher than using the original features using any induction technique. The accuracy of the automatically constructed features was comparable to that of the set of popular financial ratios. The results prove that automatic feature construction using only fragments of domain knowledge can match the results of years of manual representation change. An additional promising result is that a group

of 7 automatically constructed features gave 87.25% accuracy. This group of 7 features was hand-selected by the authors in an effort to produce a diverse set of features. Two additional experiments using this same knowledge with less restricting assumptions are described in (Donoho, 1996).¹

In addition to the constructed features slightly outperforming the experts' ratios as a group, individual features were found that got higher accuracy than many of the individual expert ratios. Some of these include:

$$\frac{TS - CO - INTR}{CASH - CL - LTD}$$

$$\frac{CASH - CL - LTD}{CASH + AR + OCA + NCA - CL}$$

$$\frac{CASH - CL}{CASH + AR + OCA + NCA - CL - LTD}$$

These could possibly be used in place of a similar popular ratio or could lead to the development of a new, useful ratio.

17.5 TURFGRASS MANAGEMENT EXPERIMENTS

The goal of the turfgrass domain is to give advice on what kind of grass to plant under various soil and climate conditions. For example, if you build a new house and are planning to seed the yard, you might consult a turfgrass expert who would suggest a grass that would grow well in your soil and climate. But there are many species of grass and often 20 to 30 variations or "cultivars" of each species, and an expert usually has in-depth experience with only a handful of cultivars. Furthermore, new cultivars are being developed each year in the search for healthier, more robust grass. Thus an expert system containing knowledge of all species and cultivars is needed. Inducing this knowledge from data is one approach to building such a system.

This study focuses on the species Kentucky Bluegrass. From 1981 to 1985 the National Turfgrass Evaluation Program (NTEP) grew 20 cultivars of Kentucky Bluegrass under a variety of climate and soil conditions and evaluated their performance. Eleven features measured were: average monthly temperature, monthly degree days, monthly precipitation, soil texture, soil pH, phosphorus level, potassium level, nitrogen fertilization level, shade, mowing height, and irrigation practices. These features and their possible values are summarized in Table 17.3. The performance of each plot of grass was rated as high, middle, or low. Each plot was measured multiple times over the five year test resulting in 1035 examples in each of the 20 datasets.

The bankruptcy experiments in Section 17.4 showed how feature construction could be used to improve predictive accuracy. Another goal of feature construction is the discovery of interesting or unexpected feature relationships. In the turfgrass domain this means discovering combinations of features that are expected to produce poor grass growth but instead result in good growth. This case study demonstrates how support knowledge (Section 17.2) can be

used to guide feature construction to discover feature relationships that produce unexpected results.

Table 17.3 Turfgrass features and values and optimal growth ranges for Kentucky Bluegrass.

Feature	Values	Optimal Range
Temperature (T)	0 to 85+ degrees F	60 to 85 degrees F
Degree days (DD)	0 to 600+	No information
Precipitation (Pr)	0 to 8.5+ inches	2.5 to 5.5 inches
Soil texture (Tx)	Sandy, loamy sand, sandy loam, loam, silty loam, silt, sandy clay loam, silty clay loam, sandy clay, silty clay, clay	Loam, silty loam, silt, sandy clay loam
pH	3.5- to 8.6+	6.0 to 7.0
Phosphorus (P)	0 to 451+ lbs/acre	75+ lbs/acre
Potassium (K)	0 to 501+ lbs/acre	200+ lbs/acre
Nitrogen (N)	0 to 8.1+ lbs/1000 ft/yr	3 to 6 lbs/1000 ft/yr
Shade (Sh)	Dense shade, uniform shade, partial shade, light shade, full sun	Full sun
Mowing height (MH)	0 to 4.1+ inches	0.75 to 2.5 inches
Irrigation (Irrg)	None, during severe stress and dormancy, to prevent dormancy, to prevent stress	To prevent dormancy, to prevent stress

The optimal ranges in Table 17.3 can be viewed as support knowledge: certain feature values are good for the grass, and other values are bad for the grass. For example, Kentucky Bluegrass generally grows best when the soil pH is between 6.0 and 7.0. Soil pH's outside this range are generally not good for Kentucky Bluegrass. Feature combinations that additively combine things correlated to the good of the grass are likely to be more correlated to the good of the grass than mixtures of features good and bad for the grass. For example, the feature pH=6.5 & N=4.0 & Texture=loam is more likely to be correlated to good growth than pH=5.0 & N=4.0 & Texture=loam because the former combines three conditions expected to cause good growth while the latter combines

two good and one bad. Since the task is to find unexpected good growth, this knowledge limits our search space for feature relationships by excluding relationships known to result in good growth. So the feature space can be ordered according to how likely a feature is to indicate good grass growth.

In the search for features that indicate unexpected good growth, we chose to reject only features that had *no* conditions outside the optimal range. This choice was based on discussions with our domain expert that grass growth is usually limited by its weakest condition. Even with this knowledge and its restrictions on the feature space, the feature space is still too large to search exhaustively. We chose to conduct a hill climbing search. This search started with conjuncts of two degenerate ranges and examined features that incrementally extended the ranges. The range extension that gave the greatest improvement was chosen, and the search continued from that point. If no extension gave improvement, and the feature covers a sufficient number of examples, the search stops. A more detailed description of the algorithm is given in (Donoho, 1996).

Table 17.4 Some constructed features that produce results that are much better than expected for Kentucky Bluegrass.

Cultivar	Constructed feature	% Covered	Avg Class Value
Aspen	(K \leq 150) & (Irrg to prevent dormancy)	4.3	2.96
Benson	(4.6 \leq pH \leq 6.0) & (Tx = sandy loam to loam)	9.2	2.66
Glade	(Tx = slty cly lm to sny cly) & (.6 \leq MH \leq 2)	6.3	2.78
Monopoly	(P \leq 60) & (No irrigation)	6.5	2.70
Rugby	(No irrigation) & (3.1 \leq N \leq 5.0)	4.6	2.89
Trenton	(Tx = sandy loam) & (Temp \leq 49.9)	6.1	2.85
Victa	(P \leq 60) & (Tx = sandy loam to loam)	6.5	2.79
Wabash	(No irrigation) & (2.1 \leq MH \leq 2.5)	5.0	2.86

The algorithm discovered a number of unexpected relationships some of which are summarized in Table 17.4. The 1035 examples were partitioned into a training set containing 690 examples and a testing set consisting of 345 examples. The algorithm produced a number of unexpected feature relationships for each cultivar which were ranked according to their average class value calculated using the training data. The predictive worth of a relationship could be determined by calculating its average class value using the unseen testing data. The columns in Table 17.4 give a cultivar, a constructed feature for that

cultivar, the percentage of test set examples that feature is true for, and the average class value of those examples. Examples in the highest class have a value 3, middle class 2, and lowest class 1. An average class value of 3.0 is a pure set with good growth. Our domain expert found a number of these constructed features interesting. One constructed feature the expert pointed out as interesting was the Aspen feature. Kentucky Bluegrass is not expected to grow optimally when potassium (K) is below 200 lbs/acre, but Aspen grows very well when the potassium is below 150 lbs/acre if the irrigation is standard (irrigate to prevent dormancy). Features such as this one are useful for knowing which cultivar to plant under certain poor conditions. They can also reveal which cultivars maintain good growth under certain low maintenance conditions.

17.6 DISCUSSION

Whereas nearly complete theories provide straightforward means to improve constructive induction, weaker knowledge may also be used. Section 17.2 discussed and exemplified eight common types of fragmentary knowledge. Instead of pinpointing specific features, fragmentary knowledge narrows the space of constructed features that must be searched, allowing limited resources to find a representation that improves inductive accuracy. Benefits of using knowledge in this manner include:

- Fragmentary knowledge of any degree of reliability can be applied; as long as it has some value, it will likely aid search for new features.
- The search procedure can suggest reliability of knowledge. After iteratively loosening assumptions (expanding the boundaries of search) the proper balance emerges, between relying on knowledge and relying on data.
- Multiple fragments of knowledge can be combined using standard evidence combination techniques.
- Once knowledge has identified the boundaries of search, any number of common search techniques may be used.

Our larger goal is to understand what knowledge from the entire spectrum (Figure 17.1) can contribute to the task of improving representation. This includes understanding the effects of knowledge on the constructed feature space and its potential to correct common flaws in the instance space (Donoho, 1996). In general, our work indicates that constructive induction is a good mechanism for incorporating knowledge because of the flexibility of the combination: Knowledge speeds the production of new features that increase inductive accuracy, and this process of constructive induction provides a means to determine knowledge reliability. A frequent side effect of the constructive induction approach is to strengthen, augment, and (re)form domain knowledge.

Aside from the results of this chapter, Donoho (1996) addressed knowledge toward the theory end of the spectrum. It showed that in the promoters domain, a combination of constructive induction and knowledge guidance produced significantly better accuracy and speed than approaches that begin with a particular learning system and apply domain knowledge without regard for the relative (but unknown) reliability of that knowledge.

Part of our research effort includes cataloging examples of knowledge helpful for feature construction and categorizing them according to their similarities. Such a taxonomy of feature construction knowledge would benefit researchers interested in improving their problem representation by helping them identify and characterize the knowledge in their domain and understand its potential for changing the feature space and instance space.

Another goal is to link knowledge types to applicable feature construction techniques. For example, the eight types of knowledge discussed in this chapter each relate to low-level construction operators and operands in somewhat different ways. With respect to higher-level issues, assorted open questions include combining knowledge offering different contributions, handling conflicting knowledge, and better understanding how to balance domain-specific knowledge with data and domain-independent bias.

In work combining domain knowledge and induction, the focus has often been on some extension of existing techniques rather than on a broad picture of the interplay among all data and domain information. Our approach shifts the research emphasis away from refinements of particular inductive techniques and toward the study of standardized principles for incorporating knowledge to aid induction.

Acknowledgments

The authors gratefully acknowledge the support of NSF grant IRI-9204473. We also thank the members of the Inductive Learning Group (Albert Tsai, Gunnar Blix, Tom Ioerger, Eduardo Perez, Ricardo Vilalta, and Larry Watanabe) and anonymous reviewers for helpful feedback at various stages of this work.

Notes

1. We also ran experiments with C4.5 as the underlying learner instead of voting. These gave results that were approximately 3% lower across the board.

References

- Aronis, J. M. and Provost, F. J. (1994). Efficiently constructing relational features from background knowledge for inductive machine learning. In *Proceedings of AAAI-94 Workshop on Knowledge Discovery in Databases*, pages 347–358.
- Craven, M. and Shavlik, J. (1995). Investigating the value of a good input representation. *Computational Learning Theory and Natural Learning Systems*, 3.

- Donoho, S. (1996). *Knowledge-Guided Constructive Induction*. PhD thesis, University of Illinois.
- Drastal, G. and Raatz, S. (1989). Empirical results on learning in an abstraction space. In *Proceedings of the 1989 IJCAI*, pages 708–712.
- Fawcett, T. and Utgoff, P. (1992). Automatic feature generation for problem solving systems. In *Proceedings of the 1992 International Conference on Machine Learning*, pages 144 – 153.
- Hirsh, H. and Japkowicz, N. (1994). Bootstrapping training-data representations for inductive learning: A case study in molecular biology. In *Proceedings of the 1994 National Conference on Artificial Intelligence*, pages 639–644, Seattle, WA.
- Kokar, M. (1986). Discovering functional formulas through changing representation base. In *Proceedings of the 1986 National Conference on Artificial Intelligence*, pages 455–459.
- Matheus, C. J. (1989). *Feature Construction: An Analytical Framework and an Application to Decision Trees*. PhD thesis, University of Illinois, Urbana, IL.
- Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161.
- Mooney, R. and Ourston, D. (1991). Constructive induction in theory refinement. In *Proceedings of the 1991 International Workshop on Machine Learning*, pages 178–182.
- Ortega, J. and Fisher, D. (1995). Flexibly exploiting prior knowledge in empirical learning. In *Proceedings of the 1995 IJCAI*, pages 1041–1047.

18 CONSTRUCTIVE INDUCTION ON CONTINUOUS SPACES

João Gama and Pavel Brazdil

LIACC, FEP - University of Porto

Rua Campo Alegre, 823

4150 Porto, Portugal

{jgama|pbrazdil}@ncc.up.pt

Abstract: In this chapter we discuss the problem of selecting appropriate operators for constructive induction. We argue that in problems that are described, at least partially, by continuous features, discriminant analysis is a useful tool for constructive induction. This new method for constructive induction was implemented in system *Ltree*. *Ltree* is a data driven constructive induction system, able to define decision surfaces both orthogonal and oblique to the axes defined by the attributes of the input space. This is done by combining a decision tree with a linear discriminant by means of constructive induction. At each decision node, *Ltree* defines a new instance space by the insertion of new attributes that are projections of the examples that fall at this node over the hyper-planes, given by a linear discriminant function. This new instance space is propagated down through the tree. Tests based on those new attributes are oblique with respect to the original input space. *Ltree* is a probabilistic tree in the sense that it outputs a class probability distribution for each query example. The class probability distribution is computed at learning time, taking into account the different class distributions on the path, from the root to the actual node. We have carried out experiments on sixteen benchmark datasets and compared our system with other well known decision tree systems, that don't use constructive induction, like C4.5, OC1, and LMDT. On these datasets we have observed that our system has advantages in terms of accuracy and tree size at statistically significant confidence levels.

18.1 CONSTRUCTIVE INDUCTION

Learning can be a hard task if the attribute space is inappropriate to describe the target concept, given the bias of an algorithm. To overcome this prob-

lem some researchers propose the use of constructive learning. Matheus and Rendell (Matheus and Rendell, 1989) define Constructive Induction as: *the application of a set of constructive operators to a set of existing features resulting in the construction of one or more new features intended for use in describing the target concept.* Performance of selective induction algorithms, in terms of predictive accuracy, is poor if the task-supplied attributes are not appropriate for describing target theories. From the machine learning literature, we selected two illustrative examples of this situation: the first one is the *KRK* problem, and the second one is the *Iris* dataset. In the *KRK* problem, the goal is to classify as *legal* or *illegal* the chess board position formed by randomly placing the three pieces *White king*, *White rook*, and *Black king*. Task attributes consists on the column and the raw of each piece. Michie and Bain applied a state-of-the-art decision-tree learner with 69% of accuracy, not differing significantly from that achieved by the default rule. By applying a constructive operator *difference* to all of the pairwise task supplied attributes, the accuracy of the decision-learner grows to 97%.

In the well known Iris dataset, the product of features *Petal Length* and *Petal Width* produces a single feature which has the dimension area. This new single feature is the basis for a decision rule that produces only four errors in the whole dataset. On both examples, the target concept is hard to learn, for selective induction algorithms that use only the task-supplied attributes. The verified increase in performance occurs because the insertion of new attributes allows the description, in a decision-tree language, of useful subconcepts for describing the target concept. On the KRK problem the useful concepts are *same column* and *same raw*, whilst on the Iris problem the subconcept was *Area*. These subconcepts could not be built by any DNF system using only original attributes. The insertion of the new attributes produces an extension of the representational power and relaxes the language bias of Decision Tree learning algorithms.

In (Wneq and Michalski, 1994), Wneq and Michalski classify existing Constructive Induction systems into four categories: *Data-Driven constructive induction systems*, that analyze and explore the input data, particularly the interrelationships among descriptors used in the examples, and on that basis suggest changes in the representation space. *Hypothesis-Driven Constructive Induction Systems*, that incrementally transform the representation space by analyzing inductive hypothesis generated in one iteration and then using detected patterns as attributes for the next iteration. *Knowledge-Driven Constructive Induction Systems*, that apply expert-provided domain knowledge to construct and/or verify new representation spaces. *Multistrategy Constructive Induction Systems* that combine different approaches and methods for constructing new representation space.

Constructive induction systems usually consist of two components: one for constructing new attributes, and the other for generating theories. After the construction, new attributes are treated exactly in the same way as the primitive ones. For a non-trivial learning problem the number of possible construc-

tive operators, such as logical operators and mathematical operators, and the number of possible operands, are usually very large, so it is not feasible to search through all the possible combinations. Usual systems select as a bias, a set of possible operators and reduce the search space of new attributes. Conjunction, disjunction, and negation are commonly used constructive operators, whereby they produce binary attributes. Another type of constructive operator is *M-of-N* including the variants *at-least M-of-N*, *at-most M-of-N*, and *exactly M-of-N*. A *M-of-N* operator generates boolean attributes. It consists on a value *M* and a set of *N* conditions based on existing attributes. An *at-least M-of-N* attribute is true if at least *M* of the *N* conditions are true. Zheng (Zheng, 1996) proposes the *X-of-N* constructor that returns the number of true conditions. It generates ordered discrete values. One of the more divulged constructive induction system is the *FRINGE* family of algorithms (Pagallo and Haussler, 1990). All algorithms from this family, iteratively build a decision tree based on the existing attributes (initially only primitive attributes), and then construct new attributes by using conjunctions and/or disjunctions of two conditions from the tree paths. The new attributes are added to the set of existing attributes, and the process is repeated. The conditions used for generating new attributes are chosen from fixed positions in the paths, either near the root and/or near the fringe of a tree.

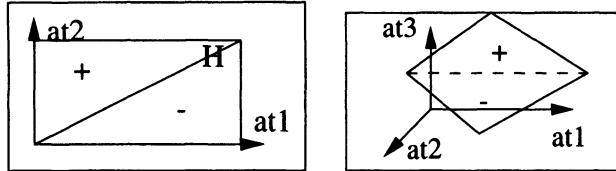
The difficult problem in the constructive induction area, is how to choose the appropriate operators for the problem in question. Few systems explore the possibility of new continuous attributes. Among them, OC1 and LMDT were used in our empirical comparisons and are described later. In this chapter, we argue that in domains described at least partially by numerical features, Discriminant Analysis is a useful tool for constructive induction (Yip and Webb, 1994).

In the next section, we describe the motivation behind *Ltree* and an illustrative example using Iris dataset. Section 3 presents, in detail, the process of tree building and pruning. Section 4 presents related work, namely, from the area of oblique decision trees. In section 5 we perform a comparative study between our system and other oblique trees on sixteen benchmark datasets from the StatLog and UCI repositories. The last section presents conclusions of the chapter.

18.2 AN ILLUSTRATIVE EXAMPLE

Consider an artificial two class problem defined by two numerical attributes, which is shown in figure 18.1.

By running C4.5 (Quinlan, 1993), we obtain a Decision Tree with 77 nodes. Obviously this tree is much more complex than expected. By analyzing paths in the tree we systematically find tests on the same attribute. One of such paths, from the root to a leaf, is given:

**Figure 18.1** The original and the new instance space

IF $at_1 \leq 0.493$ AND $at_2 \leq 0.32$ AND $at_1 \leq 0.247$ AND $at_2 \leq 0.196$ AND $at_1 \leq 0.141$ AND $at_2 > 0.116$ THEN Class -

This means that the tree is making an approximation to an oblique region by means of a staircase-like structure. This is an evidence that the description language for this problem is not appropriate. In this case, a simple test $at_1 > at_2$ is enough to build a decision surface. The description language postulates relations between attributes and its values: it does not capture relations between attributes.

Table 18.1 The new trees

$at_3 = at_1 - at_2$	$at_3 = at_1 / at_2$
IF $at_3 \leq 0$ Then Class -	IF $at_3 \leq 1$ Then Class -
IF $at_3 > 0$ Then Class +	IF $at_3 > 1$ Then Class +

If we introduce a new attribute at_3 , based on the difference between at_1 and at_2 , or based on the quotient, C4.5 is able to build a tree with 3 nodes, shown on table 18.1. Both trees perfectly describe the target concept, but the operators used for the construction of the new attribute have some limitations: the first one has a constant *slope* and the second one has a constant *intercept*.

A more general approach could be done using linear discriminant functions as a constructive operator. By running a linear discriminant procedure, in the problem in Figure 18.1, we obtain one discriminant: $H = 0 + 12 * at_1 - 12 * at_2$. Line H (Figure 18.1 a) is the hyper-plane generated by the linear discriminant. A new attribute (at_3) is generated projecting the examples over this hyper-plane. The new instance space is given in Figure 18.2.

In the new instance space a hyper-rectangle orthogonal to the new axis (Figure 18.2) is able to split classes using only one test. Running C4.5 on this new dataset, we get the tree: IF $at_3 \leq 0$ THEN Class - ELSE Class +.

This is one of the smallest possible trees that discriminates the classes. The test defines a hyper-plane that is orthogonal to the axis defined by attribute at_3 and parallel to the axes defined by at_2 and at_1 . Rewriting the rule IF $at_3 > 0$ THEN + in terms of the original space, that is in terms of at_1 and at_2 , we get: IF $at_1 > at_2$ THEN CLASS +. This rule defines a relation between attributes, thus is oblique to the at_1 and at_2 axes. This very simple

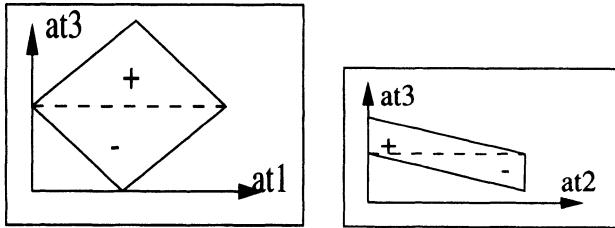


Figure 18.2 Two different views of the new instance space

example illustrates one fundamental point: Constructive Induction based on combinations of attributes extends the representational power and overcomes limitations of the language bias of Decision Tree algorithms.

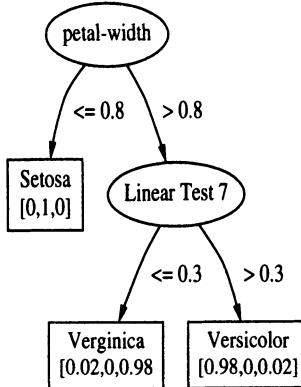
We have implemented a two step algorithm, that explores this idea as a pre-processing step. Given a dataset, a linear discriminant builds the hyper-planes. New attributes are created projecting all the examples over the hyper-planes. The transformed dataset is then passed to C4.5. We refer to this algorithm as *C45Oblique*. It was also used in our experimental study with very promising results.

Ltree dynamically explores the constructive step of *C45Oblique*. At each decision node *Ltree* builds new attributes based on linear combinations from the previous ones. If the "best" orthogonal axis involves one of the new attributes, the decision surface is oblique with respect to the original axes. There are two new aspects in our contribution to the state of the art on oblique decision trees. The first one is that the new attributes are propagated down through the tree. This allows non-linear fittings because the new attributes built at lower nodes contain terms based on attributes built at previous nodes¹. The second aspect is that the number of attributes is variable along the tree, including for two nodes at the same level. Another aspect is that this is a probabilistic decision tree in the sense that when classifying an example the tree outputs a class probability distribution that takes into account, not only the distribution at the leaf where the example falls, but also a combination of the distributions of each node from the path that the example follows.

18.2.1 The Iris data

To illustrate the basic ideas behind *Ltree*, we will use the well-known Iris dataset. The tree generated by *Ltree* is shown on Fig 18.3.

This is a tree with 5 nodes. It misclassifies 2.6% of the examples on the test data². For the same data C4.5 generates a tree with 11 nodes with 4.7% of error rate. In this example, *Ltree* proceeds as follows: at the root of the tree, there are 4 attributes and 3 classes. For each class, there are 50 examples. Two hyper-planes are constructed based on linear combinations of the four original attributes. All examples are projected over the two hyper-planes and the instance space is reconstructed by inserting two new attributes. *Ltree* call

**Figure 18.3** The Tree for Iris dataset

them *Linear Test 5* and *Linear Test 6*. Using the Gain Ratio as criteria, the attribute that *best* splits the data is chosen: in this case the attribute *Petal width*. Since the tested attribute is continuous, the original dataset is divided into two distinct datasets. In the first one, all examples belong to the same class: *Ltree* returns a leaf and the splitting process terminates. The second dataset, has 6 attributes and two classes. One hyper-plane is generated, and the examples are projected over the hyper-plane. A new attribute is built and the new instance space has 7 attributes and two classes.

Now, the attribute that best splits the data is the *Linear Test 7*, which is a linear combination of the original attributes plus the ones built at the root of the tree. The equation of the new attributes are:

$$\begin{aligned}
 \text{Linear 5} &= -32 -3*[1] - 4*[2] +7*[3] +16*[4] \\
 \text{Linear 6} &= -10 +8*[1] +15*[2]-21*[3] -25*[4] \\
 \text{Linear 7} &= +283 -4*[1] - 6*[2] +7*[3] +13*[4]-301*[5]+301*[6]
 \end{aligned}$$

This attribute splits the data into two datasets with only one class. The process of generating the tree terminates.

18.3 GROWING THE TREE

Ltree is a top down induction Decision Treedecision tree system. The main algorithm is similar to many other programs from the TDIDT family, except for the construction of new attributes at each decision node.

At each decision point, *Ltree* dynamically computes new attributes. New attributes are linear combinations of all ordered attributes of the examples that fall at this decision node. To build new attributes the system considers at each decision point, different number of classes. Taking into account the class distribution at this node, *Ltree* only considers those classes whose number of examples are greater than the number of attributes³. Suppose that in a decision node *Ltree* considers q_{node} classes ($q_{node} \leq q$). A linear discriminant function⁴ is a linear composition of the attributes where the sum of squared differences

between class means is maximal relative to the internal class variance. It is assumed that the attribute vectors for examples of class A_i are independent and follow a certain probability distribution with probability density function f_i . A new point with attribute vector \vec{x} is then assigned to that class for which the probability density function $f_i(\vec{x})$ is maximal. This means that the points for each class are distributed in a cluster centered at μ_i . The boundary separating two classes is a hyper-plane and it passes through the mid point of the two centers. If there are only two classes one hyper-plane is needed to separate the classes. In the general case of q classes, $q-1$ hyper-planes are needed to separate the classes. By applying the linear discriminant procedure described below, we get $q_{node} - 1$ hyper-planes. The equation of each hyper-plane is given by (Henery, 1993):

$$H_i = \alpha_i + \sum_j \beta_{ij} * x_j \\ \alpha_i = -\frac{1}{2}\mu_i^T S^{-1}\mu_i \text{ and } \beta_i = S^{-1}\mu_i$$

Ltree builds $q_{node} - 1$ new attributes. All the examples of this node are extended with the new attributes. Each new attribute $NewAtt_i$, is the projection of the example over the $hyper-plane_i$. The projection is computed as the *dot product* of the example vector \vec{x} by the coefficients β_i of the $hyper-plane_i$. *Ltree* uses a *Singular Value Decomposition* (SVD) in order to compute S^{-1} . SVD is numerically stable and is a tool for detecting sources of collinearity. This last aspect is used as a method for reducing the features used at each linear combination.

Because class distribution varies along the tree, the number of new attributes is also variable. Two different nodes (also at the same level) may have different number of attributes. As the tree grows and the classes are discriminated, the number of new attributes decreases. New attributes are propagated down the tree. This is the most original aspect of *Ltree*.

18.3.1 Splitting Rule

It is known that building the optimal tree (in terms of accuracy and size) for a given dataset is a NP complete problem. In this situation, we must use heuristics to guide the search. A splitting rule typically works as a one-step lookahead heuristic. For each possible test, the system hypothetically considers the subsets of data obtained. The system chooses the test that maximizes (or minimizes) some heuristic function over the subsets. By default *Ltree* uses Gain Ratio (Quinlan, 1993) as the splitting criteria. A test on a nominal attribute will divide the data into as many subsets as the number of values of the attribute. A test on a continuous attribute will divide the data into two subsets: *attribute value > cut point* and *attribute value ≤ cut point*. To determine the *cut point*, we follow a process similar to C4.5.

18.3.2 Stop Criterion

The usual stopping criteria for Decision Trees is: stop building the tree when all the examples that fall at a decision node are from the same class. In noisy domains, more relaxed rules are needed. *Ltree* uses the following rule: stop the tree growing if the percentage of the examples from the majority class is greater than a user defined parameter (by default 95%). If there are no examples at a decision node, *Ltree* returns a leaf with the class distribution of the predecessor of this node.

18.3.3 Smoothing Classes Distribution

In spite of the positive aspects of divide and conquer algorithms, one should be concerned about the statistical consequences of dividing the input space. Dividing the data can improve the bias of an estimator, because it allows data fine fitting, but, in general, it increases the variance. The use of *soft thresholds* is an example of a methodology that tries to minimize the effects of splitting the data⁵. *Ltree* uses a kind of smoothing process (Buntine, 1990) that usually improves performance of tree based classifiers. When classifying a new example, the example traverses the tree from the root to a leaf. The class attached to the example takes into account not only the class distribution at the leaf but all class distributions of the nodes in the path. That is, all nodes in the path contribute to the final classification. Instead of computing class distribution for all paths in the tree at classification time, as it is done in (Quinlan, 1992), *Ltree* computes a class distribution for all nodes when growing the tree. This is done recursively, taking into account class distributions at the current node and at the predecessor of the current node.

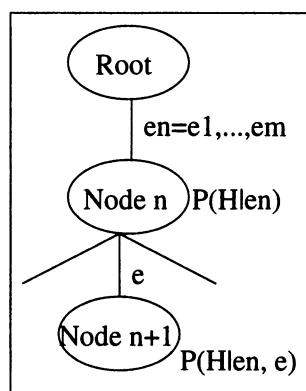


Figure 18.4 Smoothed Classes Distribution

The justification for this procedure can be found at the Bayesian formalism. At each node we have a class distribution that depends on the sequence of observed evidences given by attribute values found on the path from the root to this node. Let H denote the hypothesis that an example is from $Class_i$, and

$e_n = e_1, e_2, \dots, e_m$ the set of observed attribute values (Figure 18.4). A new fact e is observed (this corresponds to the value of the attribute tested at this node) and a set of examples is moved from the current node, $Node_n$, to a descendant one $Node_{n+1}$. We can compute the impact of the new data e by the formula:

$$P(H|e_n, e) = P(H|e_n) \frac{P(e|e_n, H)}{P(e|e_n)}$$

The belief $P(H|e_n)$, assumes the role of the prior probability in the computation of the new impact of data, and then summarizes the past experience. To update the belief we only need to multiply it by the likelihood function, which measures the probability of the new data e , given the hypothesis and the past observations. In the context of tree learning, $P(e|e_n)$ is the probability that one example that falls at $Node_n$ goes to $Node_{n+1}$, and $P(e|e_n, H)$ is the probability that one example from class H goes from $Node_n$ to $Node_{n+1}$ (Pearl, 1988).

This recursive formulation of updating beliefs, allows *Ltree* to efficiently compute class distributions for all nodes when growing the tree.

We have performed an intensive experimental comparison using a regular Decision Tree and turning the smoothing process *on* and *off*. We have observed that the smoothed scheme improves both accuracy and tree size, but while the improvement on the tree size was consistent at statistical significant confidence levels, in terms of accuracy, it was not statistically significant.

Pruning the tree and processing missing values explore the smoothed class distributions. Another side effect is when considering problems with cost matrices: there are very simple algorithms for minimizing costs that use class probabilities (Michie et. al., 1994).

18.3.4 Pruning

Pruning is considered to be the most important part of the tree building process, at least in noisy domains. Statistics computed at deeper nodes of a tree have low level of significance due to the small number of examples that fall at these nodes. Deeper nodes reflect the training set too much (overfitting) and increase the error due to the variance of the classifier. Several methods for pruning decision trees are presented in literature (Breiman et. al., 1984; Quinlan, 1993). The process that we use exploits the way how *Ltree* computes class probability distributions at each node. Usually, for example in C4.5, pruning is a process that increases the error rate on the training data. In our case, this is not necessarily true. The class that is assigned to an example takes into account the path from the root to the leaf, and is often different from the majority class of the examples that fall at one leaf. At each node, *Ltree* considers the *static error* and the *backed-up error*. *Static error* is the number of misclassifications considering that all the examples that fall at this node, are classified from the majority class taken from the smoothed class distribution at this node. *Backed-up error* is the sum of misclassifications of all subtrees of the current node. If *Backed-up error* is greater or equal than *Static error*, then the node is replaced by a leaf with the class distribution of the node.

18.3.5 Missing Values

When using a tree as a classifier, the example to be classified passes down through the tree. At each node a test based on the values of the attributes is performed. If the value of the tested attribute is not known (often in real data some attribute values are unknown or undetermined) the procedure can't choose the path to follow. Since a decision tree constitutes a hierarchy of tests, the *unknown problem* has special relevance on this type of classifiers. *Ltree* passes the example through all node branches where the unknown attribute value was detected (Buntine, 1990). Each branch outputs a class distribution. The output is a combination of the different class distributions that sum to 1.

18.4 RELATED WORK

In CART book (Breiman et. al., 1984), Breiman et al. suggest the use of a linear combination of attributes. The algorithm, presented in CART, cycles through the attributes at each step, performing a search for an improved linear combination split. Reconstruction of the input space by means of new attributes based on combinations of original ones, appears as a pre-processing step in Yip and Webb (Yip and Webb, 1994) and in Ittner and Schlosser (Ittner and Schlosser, 1996). Yip's system CAF, incorporates new attributes based on canonical discriminant analysis⁶. They show that such techniques can improve the performance of machine learning algorithms. *Ltree* uses a similar strategy at each decision node. One advantage is that for each sub-space, local inter-dependencies between attributes are captured. Another advantage of *Ltree* is the constructive step: new attributes are propagated downwards through the tree. The basic idea of Ittner's system is the construction of all possible pairwise products and squares of the original numerical attributes. The dataset is augmented with those new attributes. Induction is done by OC1 which is described later. In spite of some positive aspects (namely non linearity) the choice of constructors is perfectly arbitrary.

Similar to CART, OC1 (Murthy et. al., 1994), uses a hill-climbing search algorithm. Beginning with the best axes-parallel split, it randomly perturbs each of the coefficients until there is no improvement in the impurity of the hyper-plane. As in CART, OC1 adjusts the coefficients of the hyper-plane individually finding a local optimal value for each coefficient at a time. The innovation of OC1 is a randomization step used to get out of local minima. As such, OC1 explicitly search for a set of coefficients that minimizes the impurity function. OC1 generates binary trees. The implementation that we have used in comparative studies only accepts numerical descriptors. Linear Machine Decision Tree (LMDT) by Brodley and Utgoff (Brodley and Utgoff, 1995), uses a different approach. Each internal node in an LMDT tree is a set of linear discriminant functions that are used to classify an example. The training algorithm repeatedly presents examples at each node until the linear machine converges. Because convergence cannot be guaranteed, LMDT uses heuristics to determine when the node has stabilized. Trees generated by LMDT are

not binary. The number of descendants of each decision node is equal to the number of classes that fall at this node. Trees build by LMDT always use, at each node, multivariate splits. The method for construction of *discriminants* is a error-minimizing process.

Ltree can use both univariate and multivariate splits, and only choose a multivariate split, if it minimizes also the *entropy* criteria. This point and the downward propagation of the new attributes are the main advantage of *Ltree* over OC1 and LMDT.

18.5 EXPERIMENTS

In order to evaluate our algorithm we performed a 10 fold Cross Validation (CV) on sixteen datasets from the StatLog⁷ repository and from the UCI repository⁸. On the first set of experiences we compared C4.5 with *C45Oblique*. On the 16 datasets, an increase of performance on 11 datasets, was verified. At a confidence level of 90% from *t paired tests*, the difference was significant in 4 datasets. At this confidence level, *C45Oblique* was significantly worse on 2 datasets (Letter and Votes).

On the second set of experiences, we compared *Ltree* with two oblique decision trees: LMDT and OC1, that are available through the Internet, and C4.5 (release 8) as the state of the art in the area of decision trees. Datasets were permuted once before the CV procedure. All algorithms were used with default settings. In each CV iteration, all algorithms learned on the same train partition of the data⁹. Classifiers were tested also on the same test partition of the data. In order to have statistical confidence on the differences we have compared the results using *t paired tests*. The null hypothesis is that *Ltree* has the same performance in comparison to each of the other algorithms. Column *ttest* refers to the significance of the comparison between *Ltree* and LMDT, OC1, C4.5, and *C45Obl* in that order. A + or - sign means that there was a significant difference at 90% of confidence level. A minus sign means that *Ltree* was worse than the given algorithm at this confidence level, and a plus means that *Ltree* performed significantly better. Table 18.2, presents the mean and error rate standard deviations on the first line of each dataset. The second line refers to tree sizes in terms of number of nodes. For each dataset the best result is presented in *boldface*.

Our system has an overall good performance. There are two main reasons. The first one is the use of linear combinations of attributes that capture inter-relations between variables and the propagation of these new attributes down through the tree. The second reason is the use of smoothing. However, *Ltree* is significantly worse than *C45Obl* in one dataset. We think this is due to the fact that C4.5 is more sophisticated than *Ltree* in details (like default parameters and internal thresholds). Using *Ltree* in a set of experiments, but performing the constructive step only at the root of the tree, the results obtained were, most of the time, worse than the standard process.

Table 18.2 Error rates and Tree sizes

	<i>ttest</i>	<i>Ltree</i>	<i>LMDT</i>	<i>OC1</i>	<i>C4.5</i>	<i>C45Obl</i>
Australian	+	13.9±4 14.2	35.9±12 72.6	14.8±6 13.2	15.3±6 35.5	14.9±5 21.1
Balance		10.4±4 6.9	12.0±4 59.6	10.4±3 20.8	34.6±4 43.5	9.0±4 7.6
Breast(W)	+	+	3.1±2 3.0	5.1±4 19.2	6.4±5 19.4	6.1±6 29.0
Diabetes	+	+	24.4±5 43.2	31.9±7 139.8	28.0±8 33.0	24.7±7 42.6
German	+		26.4±5 18.5	39.5±10 85.0	25.7±5 24.4	29.1±4 151.1
Glass			34.7±8 34.8	37.0±10 52.0	32.8±11 16.4	34.6±10 44.4
Heart	+	+	+	15.5±4 4.2	25.9±5 29.0	29.3±7 15.6
Hepatitis		+	18.9±9 10.6	20.3±13 6.6	25.5±13 7.8	24.9±13 17.0
Iono		+	9.4±4 15.0	18.8±11 11.0	11.9±3 9.8	9.1±5 27.4
Iris	+	+	2.7±3 5.0	6.0±7 7.0	7.3±6 6.2	4.7±5 8.4
Letter		-	18.2±1 1522.0	- ^a 2349.0	16.7±1 1962.6	13.7±1 1823.6
Segment		+	2.9±1 54.6	3.6±1 50.9	4.5±1 37.2	3.6±2 81.2
Vehicle	+	+	22.5±5 97.4	21.8±5 74.1	34.1±5 120.0	28.8±4 139.0
Votes	+		3.5±4 4.4	22.3±12 11.6	6.7±6 3.0	2.8±3 10.8
Waveform	+	+	17.8±2 167.2	20.2±3 115.8	22.1±1 61.8	23.9±2 309.2
Wine		+	2.8±3 5.0	3.4±4 4.0	9.0±6 9.2	6.7±8 9.6
<i>Mean</i>			14.2 125	20.2 ^a 49 ^a	17.8 171	17.6 184
						14.7 147

^aOn *Letter* data LMDT returns *Bus Error* on 9 of the ten iterations of CV.

18.5.1 How far from the best ?

For each dataset we consider the algorithm with the lowest error rate. Call *error margin* the standard deviation from a Bernoulli distribution for the error rate of this algorithm. This is computed as $E_m = \sqrt{\frac{\text{Error}_{\text{Best}} * (1 - \text{Error}_{\text{Best}})}{|\text{TestSet}|}}$.

The distance of an algorithm error rate from the best algorithm on each dataset, in terms of the *error margin*, $\frac{\text{Error}_i - \text{Error}_{\text{Best}}}{E_m}$, gives us an indication

about algorithm's performance in reference to the best algorithm. Table 18.3 presents the average of the distances over all datasets where all algorithms run.

Table 18.3 Averages of Bernoulli Distances

	<i>Ltree</i>	<i>Lmdt</i>	<i>OC1</i>	<i>C4.5</i>	<i>C45Obl</i>
<i>Average</i>	0.39	5.57	3.46	3.68	0.75

There is strong evidence that if we need to use a learning classifier on new data, without anymore information, we will first try *Ltree*.

Ltree, LMDT, and OC1 generate trees with multivariate tests on each of their decision nodes. Tree size between these systems could be compared directly. Comparisons with univariate tests are more problematic because we must take into account the increase in complexity of the combinations of attributes. Oblique trees (namely *Ltree*) are more compact than univariate ones. Although some nodes contain multivariate tests which are more difficult to understand, the overall tree is, on the whole, substantially smaller.

The training time for divide and conquer is often in orders of magnitude faster than gradient based algorithms (Jordan and Jacob, 1994). This is one of the aspects why *Ltree* is significantly faster than other oblique decision trees like OC1 and LMDT that use gradient descendent approaches to determine hyper-planes. We verify, for example on *Letter* data, that OC1 takes about 10 hours in a Sun Sparc10 to run 1 fold of cross validation. Similar time was needed for LMDT. Each run of *Ltree* takes around 50 minutes.

18.6 CONCLUSIONS

We described a method for the construction of oblique decision trees, combining a Decision Tree with a linear discriminant by means of Constructive Induction. There are two main features in our system. The first one is the use of Constructive Induction. When building the tree, new attributes are computed as linear combinations from the previous ones. As the tree grows and the classes are discriminated, the number of new attributes decreases. The new attributes, created at one node, are propagated downwards through the tree. This allows non-linear fittings because the new attributes built at lower nodes contain terms based on attributes built at previous nodes. The second aspect is that this is a probabilistic decision tree in the sense that, when classifying an example, the tree outputs a class probability distribution that takes into account not only the distribution at the leaf where the example falls, but a combination of the distributions of each node on the path that the example follows.

In problems with numerical attributes, attribute combination extends the representational power and relaxes the language bias of univariate decision tree algorithms. In an analysis based on Bias-Variance error decomposition (Kohavi and Wolpert, 1996), *Ltree* combines a linear discriminant which is known to

have a high bias, but a low variance with a decision tree known to have low bias, but high variance. This is the desirable composition of algorithms. We use constructive induction as a way of extending the description language bias. Using Wolpert's terminology (Wolpert, 1992), the constructive step performed at each decision node is a bi-Stacked Generalization.

From this point of view, the proposed methodology can be seen as a general architecture for combining algorithms by means of Constructive Induction, a kind of *local bi-stacked generalization*. The constructive operator used can be easily replaced by a quadratic discriminant, a logistic discriminant, or a neural net. This architecture could be used on regression problems by replacing our attribute constructor operator, the linear discriminant function, by an operator based on principal components analysis.

We have shown that this methodology can improve accuracy, tree size, and learning times, comparatively to other oblique decision trees' systems that don't use constructive induction.

Acknowledgments

Gratitude is expressed to the support given by the PRAXIS XXI project, FEDER project, and the Plurianual support attributed to LIACC. I thank my colleagues from LIACC for reading previous versions of this chapter.

Notes

1. We call them non-linear, in the sense that they can not be discovered using only the primitive attributes.
2. In a 10 Fold Cross Validation
3. Because data underfits the concept, see Breiman et al. (Breiman et. al., 1984)
4. Throughout this chapter we adopt the following notation. The training set consists of N examples drawn from q known classes. For each class there are n_i examples, each example is defined by p attributes and is represented by the vector $\vec{x} = (x_1, \dots, x_p)$. The sample mean vector for each class is represented by μ_i . The sample covariance matrix for the i class is S_i , the pooled covariance matrix S is given by $\sum_i \frac{n_i \cdot S_i}{n - q}$.
5. The propagation down of the new attributes is also a more sophisticated form of soft thresholds.
6. This is similar to *C4.5Oblique*
7. <http://www.ncc.up.pt/liacc/ML/statlog/index.html>
8. <http://www.ics.uci.edu/AI/ML/Machine-Learning.html>
9. Slightly syntactic modifications of the data were necessary to run OC1.

References

- Breiman, L., Friedman, J., Olshen, R., and Stone, C., (1984). Classification and Regression Trees. Wadsworth International Group.
- Brodley, C. and Utgoff, P., (1995). Multivariate Decision Trees. in *Machine Learning*, 19, Kluwer Academic Publisher.

- Buntine, W, (1990). A theory of learning Classification rules. PhD thesis, University of Sydney.
- Henery, B. (1993). FORTRAN programs for Discriminant Analysis. Internal report, Dep. Statistics and Modelling Science, University of Strathclyde.
- Ittner, A. and Schlosser, M. (1996) Non linear Decision Trees-Ndt. in *Proceedings of 13 International Conference on Machine Learning* - IML96, Ed. Lorenza Saitta.
- Jordan, M. and Jacob, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computing*, n.6.
- Kohavi, R. and Wolpert, D. (1994). Bias plus variance decomposition for zero-one loss functions. in *Proceedings of 13 International Conference on Machine Learning* - IML96, Ed. Lorenza Saitta.
- Matheus,C. and Rendell, L. (1989). Constructive Induction on Decision Trees. in *Proceedings of IJCAI 89*.
- Michie, D., Spiegelhalter, J. and Taylor, C. (1994) Machine Learning, Neural and Statistical Classification. Ellis Horwood.
- Murthy, S., Kasif, S., and Salzberg, S. (1994). A system for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*.
- Pagallo, G. and Haussler, D. (1990) 'Boolean feature discovery in empirical learning. *Machine Learning* 5, Kluwer Academic Publisher.
- Pearl, J. (1988). Probabilistic Reasoning in intelligent systems: Networks of Plausible Inference. Morgan Kaufmann.
- Quinlan, R. (1992). Learning with continuous classes. in *Proceedings of AI'92*.
- Quinlan, R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann.
- Wneq, J. and Michalski, R. (1994). Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. in *Machine Learning*, 14.
- Wolpert, D. (1992). Stacked Generalisation. *Neural Networks Vol.5*.
- Yip, S., and Webb, G. (1994). Incorporating canonical discriminant attributes in classification learning. in *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, Morgan Kaufmann.
- Zheng, Z. (1996). "Constructing New Attributes for Decision Tree Learning. Ph.D. Thesis, University of Sydney.

V Combined Approaches

19

EVOLUTIONARY FEATURE SPACE TRANSFORMATION

Haleh Vafaie¹ and Kenneth De Jong²

¹Datamat Systems Research, Inc.
8260 Greensboro Dr., Suite 255
McLean, VA 22102
USA
hvafaie@dsri.com

²Computer Science Department
George Mason University
Fairfax, VA 22030
USA
kdejong@gmu.edu

Abstract: The ability to accurately identify and classify objects described in terms of feature vectors is highly dependent on the particular features chosen. In complex domains such as image understanding and machine learning it is frequently the case that there are many candidate features, but little insight as to which, if any, make up an effective set for identification and classification. In this chapter we describe a system that highly automates the process of constructing and selecting useful feature sets. A key element of the design of this system is the use of evolutionary computation techniques to manage the underlying search processes. Empirical results are presented which suggests that this approach is highly effective in complex, real world problem domains.

19.1 INTRODUCTION

Accurate classification is an important component for developing effective systems for many problem domains such as image processing and machine learning. Classification systems assign a set of input objects to a set of decision classes. The accuracy of the produced results mainly depends on how well a problem is represented using a set of features. In most applications of classification

systems considerable effort is placed on finding a feature set that produces satisfactory results. For example, in machine learning and image understanding, finding an acceptable set of features for accurate recognition and classification is a key element in designing efficient and implementable systems.

The amount of effort required to properly represent a problem can vary widely. If an appropriate set of features is known and available, the representation for a given problem is simple. For cases where there are a few candidate features, an acceptable feature set can frequently be found through trial and error. However, for many real world problems there are many candidate features, there is little insight or knowledge as to which features are more suitable, and it is not even clear that an adequate set of features is available.

In such cases what is required are tools to assist the system designer in both constructing and selecting useful features. In the remaining sections we review some of the approaches that have been taken in the past, we describe a new approach based on evolutionary computation techniques, and we present experimental evidence of the robustness of this approach.

19.2 APPROACHES TO FEATURE SELECTION AND CONSTRUCTION

Feature selection and feature construction methods apply different strategies to change a problem's representation space. This section presents a short characterization of the various approaches.

19.2.1 Feature Selection

There is strong motivation to design and implement systems using small feature sets, since every feature used can increase the cost and running time of a classification system as well as reduce the accuracy of the produced result. At the same time there is a potentially conflicting need to include a sufficient set of features to achieve high accuracy under difficult conditions. This has led to the development of a variety of feature selection techniques within the image processing community for finding an "optimal" subset of features from a larger set of possible features. These strategies fall into two main categories (Vafaie, 1997).

The first approach selects features independent of their effect on classification performance. This generally involves transforming the original features using domain-specific procedures to form a new reduced set of features that preserves most of the information provided by the original features and is more reliable because of the removal of redundant and noisy features. The major difficulty here is in identifying an appropriate set of transformations.

The second approach selects a subset N of the available M features in such a way that classification performance is not significantly degraded. In most cases this involves some kind of a search procedure used for finding effective subsets of features. Exhaustively trying all the subsets is computationally prohibitive when there are a large number of features. Non-exhaustive search techniques

such as sequential backward elimination and sequential forward selection avoid the combinatorial explosion of exhaustive search at the expense of ignoring non-linear interactions among the given features.

There has been an increasing interest in research on optimal feature selection in the machine learning community. Two basic approaches have been taken: the filter approach and the wrapper approach (Kohavi and John, 1995). The filter approach selects features independent of their effect on the performance of the inductive learning algorithm. This approach is typically computationally less expensive, but is also frequently less effective than methods which correlate feature set changes directly with classification performance.

In the wrapper approach, the feature subset selection is performed using an induction algorithm as part of the evaluation function. Here the traditional biases for simplicity lead to efficient induction procedures that produce individual rules (trees) containing only a few features to be evaluated. However, each rule (tree) can and commonly does use a different set of features, resulting in much larger cumulative feature sets than is generally acceptable for many classification problems. This problem is elevated by the tendency of traditional machine learning algorithms to overfit the training data, especially in the context of noisy data, resulting in the need for a variety of ad hoc truncating (pruning) procedures for simplifying the induced rules (trees).

19.2.2 Feature Construction

In most real world problems, the relationship between the initial feature set and the desired classification behavior is complex. Considerable research has focused on the restructuring of the initial representation space to reduce this complexity. This has resulted in the development of several approaches for modifying the representation through the introduction of new features. The strategies applied fall into two main categories (Vafaie, 1997). The first approach involves substantial effort from domain experts to develop a set of domain-specific procedures for transforming the original features into a new set of more useful features.

For many domains, however, such expertise is unavailable, resulting in the system designer exploring combinations of features in a systematic or trial-and-error fashion in order to find a suitable feature set. Such approaches, however, scale up poorly.

As an alternative, researchers have been experimenting with various machine learning techniques, applying them in combination with domain dependent techniques to accomplish automated feature construction. The work in this area can be categorized based on the search methods employed for finding features to be added to the initial feature set such that the representation of a given problem is improved. Many systems use the structure of the evolving classifier to guide their search. In these types of algorithms the search for new features typically involves a divide-and-conquer approach. These algorithms work well when there are a very small number of interacting features. However,

these methods are limited when dealing with complex problems involving many interacting features.

Another group of systems uses the behavior of the learning algorithm to guide the search. New features are constructed when a failure or misclassification occurs. A form of greedy search is used to find new features, which reduces computation time but fails to handle interacting sets of features.

Some systems use domain knowledge or theory to guide the search for new features. However, these systems are appropriate only if the domain theory or knowledge is available and can be stated in the required format.

Other systems create new features solely based on searching the space of all possible combinations of the initial features. However, to minimize computation time only pairwise combinations of features are considered.

19.2.3 Summary

It is evident that both feature selection and feature construction methods are faced with difficult search problems, since they have to explore very large search spaces. The search methods used that are based on heuristics or some knowledge about the problem apply greedy or directed search algorithms to search the space more efficiently. However, these methods are very brittle and may get trapped on local peaks when dealing with complex and noisy data, producing inadequate results (Vafaie and De, Jong, 1993).

Since genetic algorithms are best known for their ability to efficiently search large spaces about which little is known and have been shown to be quite robust across a wide range of problem domains, they seem to be an excellent choice as a search procedure to be used for feature selection and feature construction strategies (Bala et al., 1995; Bala et al., 1996). In the remaining sections of this chapter we describe a system which is based in these ideas, and we present results which show its effectiveness in both constructing and selecting useful features in complex, real world domains.

19.3 GENERAL SYSTEM ARCHITECTURE

Our methodology for transforming initial feature sets into more effective ones is shown in Figure 19.1.

In this approach it is assumed that an initial set of features will be provided as input in addition to a training set in the form of feature vectors representing positive and negative examples of the various classes for which classification is to be performed. Depending on the state of the problem either the feature selection module or feature construction module is applied to search the problem space to improve the accuracy of a given classification system.

Attempting to perform both feature construction and feature selection in parallel raises complex issues of synchronization and representation. By allowing both processes to operate independently, one can avoid such issues and achieve considerable flexibility in controlling the way that feature selection and feature construction processes interact. In particular, there is sufficient gener-

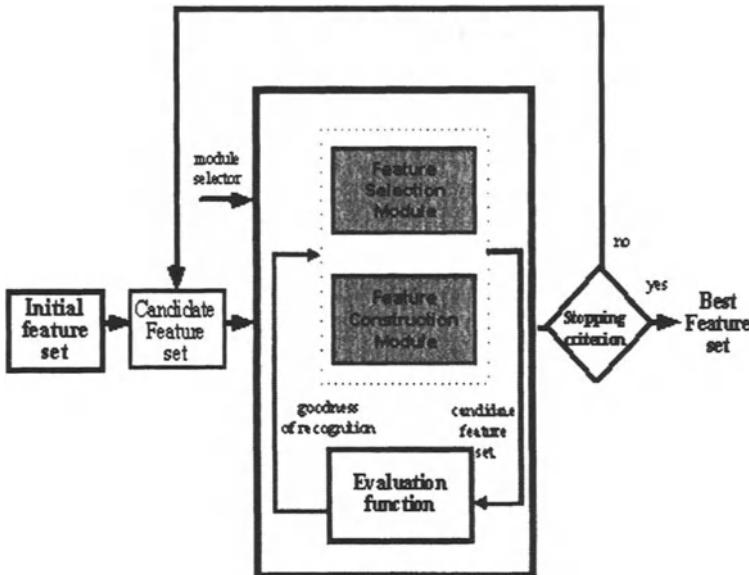


Figure 19.1 Block diagram of the system architecture.

ability here to allow the modules to be selected dynamically, depending on the state of the problem.

Both the feature selection and feature construction processes are based on genetic algorithms that use an evaluation function as a feedback to guide their search. In this system that feedback consists of the performance of each of the selected feature subsets evaluated by invoking an evaluation function with the accordingly modified feature space and training set, and measuring classification accuracy. The best feature subset found is then output as the recommended set of features to be used in the final design of the classification system.

19.3.1 Genetic Algorithms

Genetic algorithms (GAs) are adaptive search techniques that were initially introduced by Holland (Holland, 1975). Genetic algorithms commonly maintain a constant-sized population of individuals that represent samples of the space to be searched. Each individual is evaluated based on its overall fitness with respect to the given application domain. New individuals are constructed by selecting high performing individuals to produce "offspring" that preserve many of the features of their "parents". The result is an evolving population that has improved fitness with respect to the given goal.

Using two main genetic operators, crossover and mutation forms new individuals (offspring) for the next generation. Traditional crossover operates

by randomly selecting a point in the two selected parents gene structures and exchanging the remaining portion of the parents to create new offspring. Therefore, crossover combines the features of two individuals to create two similar offspring. Mutation operates by randomly changing one or more components of a selected individual. It acts as a population perturbation operator and is a means for inserting new information into the population. This operator prohibits any stagnation that might occur during the search process.

The main issues in applying GAs to any problem are selecting an appropriate representation and an adequate evaluation function and will be described in the following subsections.

Representation for Feature Selection. For feature selection, the principal interest is in representing the space of all possible subsets of the given feature set. Consequently, the simplest form of representation is to have one binary gene for each feature, in which the value of a gene represents the presence (or absence) of that feature in the feature set. Therefore, each individual consists of fixed-length binary string representing some subset of the given feature set. For example, an individual of length d corresponds to a d dimensional binary feature vector X , where $x_i = 0$ represents elimination and $x_i = 1$ indicates inclusion of the i th feature.

The benefit of this representation is that the classical GA operators (binary crossover and mutation) can easily be applied to this representation without any change. This eliminates the need for designing new genetic operators, or making any other modifications to the standard form of genetic algorithms.

Representation for Feature Construction. The most natural way to construct useful new features are by forming combinations of existing features through a well-chosen set of operators. For example, in image processing domains initial features are commonly real-valued and effective new features are derived by combining existing features using simple arithmetic operations (such as $+$, $-$, $*$, $/$). Therefore, new features can be described as expressions combining features and operators, such as $(F1 - (F2 * F3))$ or $(F4 - F8)$, and represented naturally as tree structures.

However, we are interested in evolving sets of features that work well together, consequently an individual in this case is a variable-length structure representing a set of features, some of which may be from the original feature set and some of which may be expressions. For example,

$$\{((F1 - F2) * (F2 + F4)), (F4 - F8), F7\}$$

would represent a set of 3 features, the first two of which are expressions. Such feature sets (individuals) are naturally represented as a list of tree structures as illustrated in Figure 19.2.

Closely related to the choice of representation is the selection of useful forms of the genetic operators of crossover and mutation. To efficiently apply the recombination operator to this representation it must be taken into consideration

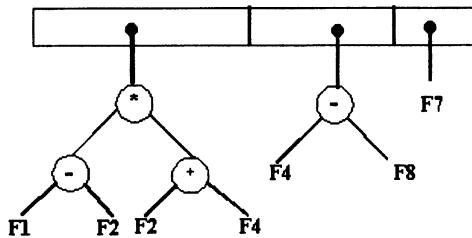


Figure 19.2 An example of an individual.

that there are two levels of representations: feature set used to solve a problem and individual features themselves. Consequently, two forms of recombination operators are needed to best accommodate this representation.

At the feature set level, the main interest is in evaluating the effectiveness of different combinations of features and hence, a form of the traditional operators can accommodate both the required exploration and the exploitation. In this case, selecting segments of parents delimited by the gene (feature) boundaries and exchanging them to create offspring as illustrated in Figure 19.3 performs crossover.

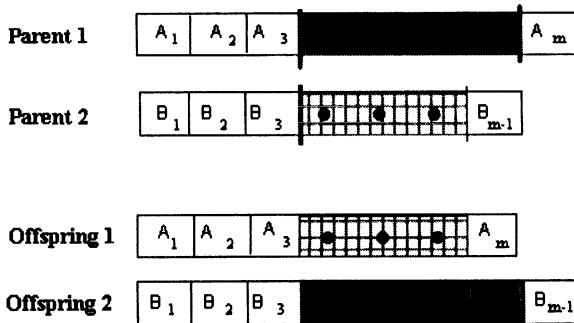


Figure 19.3 An example of crossover at gene boundaries.

The mutation operator maintains variation in the population by simply replacing the selected gene (feature) with a member of the initial feature set as shown in Figure 19.4.

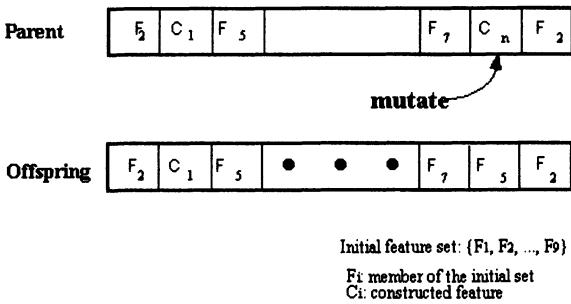


Figure 19.4 An example of mutation on simple genes.

At the feature level, a tree represents each feature or gene. Here, a new form of genetic operators is necessary to accommodate the tree representation. A crossover must permit for expansion as well as truncation of features in terms of the number of operators and the initial features used. Hence, it must be able to perform recombination on partial subtrees developed as part of any feature. The crossover operator we use follows the one used by Koza for his genetic programming paradigm, which is well suited for variable length hierarchical structures (Koza, 1994). In this operation portions of parents that are selected by the crossover point are swapped. Figure 19.5 illustrates how such an operator might exchange selected features from two feature sets.

Applying mutation to tree structures is straightforward, and involved randomly selecting an operator or feature node to be modified as illustrated in Figure 19.6.

19.3.2 Evaluation Function

Choosing an appropriate evaluation function is an important step for successful application of GAs to any problem domain. Evaluation functions provide GAs with feedback about the fitness of each individual in the population. GAs then use this feedback to bias the search process to provide an improvement in the population's average fitness.

Even though the feature selection module and the feature construction module use different representations, they are both designed to evolve better feature subsets. Hence, a single evaluation function can be used for both modules that estimates the "fitness" of a feature subset. Since the primary interest is in improving classification performance, fitness evaluation involves a multi-step procedure illustrated in Figure 19.7.

The evaluation function is divided into three main steps. After a feature subset is selected, the initial training data that consists of the entire set of feature vectors and class assignments corresponding to examples from each of the given classes is modified. This is done by adding the values for new features

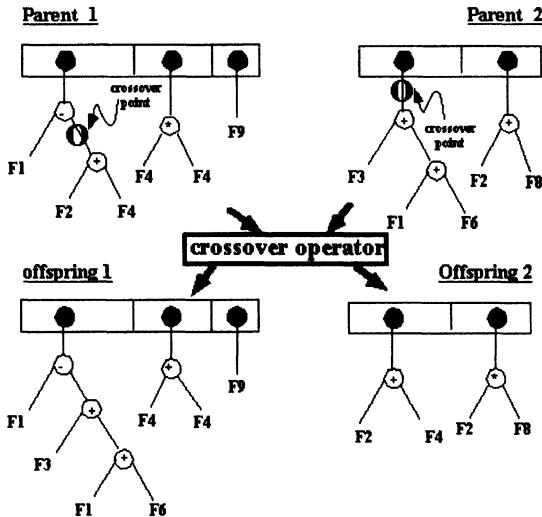


Figure 19.5 An example of crossover on tree structures.

that are not present in the initial feature vectors, and removing the values for features that are not present in the selected set of feature vectors. The second step is to construct a classifier using the given features in conjunction with the newly modified training data. In the experiments presented in this chapter C4.5 (Quinlan, 1986) is used to develop decision tree classifiers. Other classifier construction techniques could be used as well. Our choice of C4.5 was based on the fact that it is fast, readily available and widely used for generating classifiers. The last step is to evaluate the classification performance of the C4.5-produced decision tree on the test data.

19.4 EXPERIMENTAL SETUP

For the experiments reported here, C4.5 (without any modification) was used to build up the decision trees for the evaluation procedure, and standard GA modules were used for feature selection and construction. For all components, standard default parameter settings from the literature were used. For the GA modules this meant a crossover rate of 0.6, a mutation rate of 0.001, and a constant population size of 50. For C4.5, the pruning confidence level was set to default value of 25%.

The data collected involved measuring both the changes in number of features used and the classification accuracy associated with the “optimal” feature sets produced. Since classification accuracy can vary considerably as a function of how data sets are split into training and testing data, results are reported as

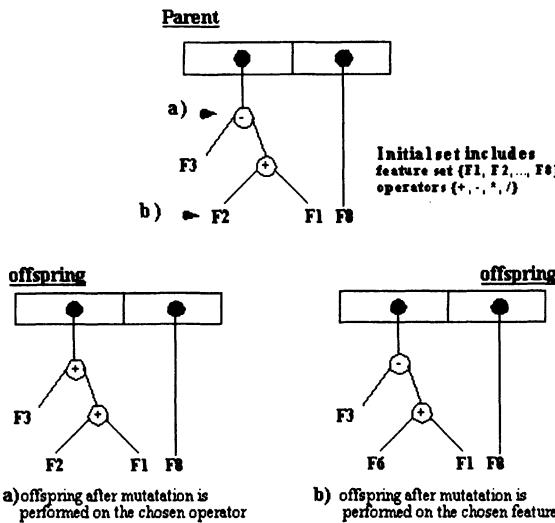


Figure 19.6 Examples of mutation on tree structures.

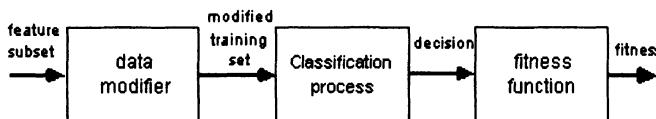


Figure 19.7 Feature set evaluation function.

average error rates along with 95% confidence intervals. The results reported here are based on a two-level split of available data. First, one third of the data is set aside for final testing purposes. The remaining two thirds is used by the feature selection and construction modules. Since GA procedures involved are stochastic algorithms, the results are averaged over 10 independent runs. Internally, C4.5 constructs its decision trees by splitting its data into two subsets: a training set (two thirds) and a testing set (one third). This two level splitting procedure was repeated three times to minimize any artifacts due to the choice of split points.

19.4.1 Initial Experimental Results on Facial Data

The ability to detect important facial features is an essential component of any face recognition system (Bala et al., 1995). Among the many facial features available the eyes play the most important role in face recognition, because accurate eye detection is a significant part of face normalization which facilitates

further localization of facial landmarks. Eye detection allows one to focus attention on important facial configurations, to filter out structural noise, and to achieve eventual face recognition.

The facial data used in these experiments comes from the FERET data base (Gutta et al., 1995). The images used to create the data are cut from human face images and 105 features are produced for each example using standard feature extraction techniques as illustrated in Figure 19.8.

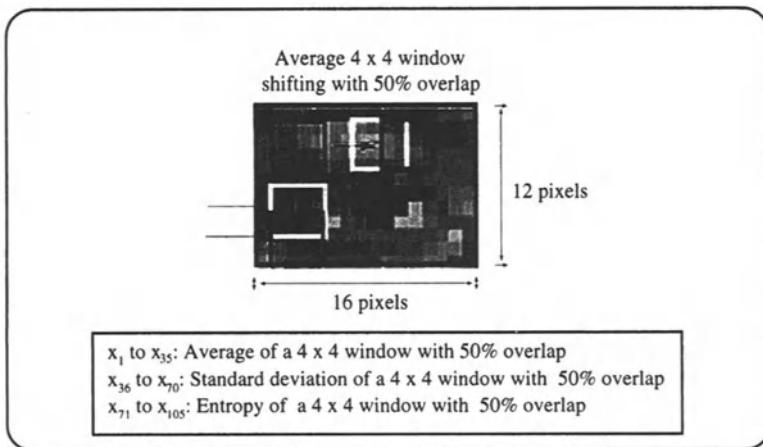


Figure 19.8 Extraction of Face Features.

This represents a fairly typical situation in which it is relatively easy to define and extract a large set of candidate features, but it is not clear at all as to whether this feature set contains a small subset of useful features. In the experiments described here 154 images were used (102 for training and 52 for testing) to see which, if any, of the 105 extracted features were useful for eye detection.

Figure 19.9 shows the reduction in average error rate obtained by using only the feature construction module. Steady improvement is observed, but levels off at fairly high error rates. In these experiments the feature construction module uses simple arithmetic operators (+, -, *, /) to construct new combinations of the initial features. Of the three two-level splits, the second data set produced the highest accuracy (lowest error rate) and the feature subset it produced during the best run of this set was used to generate the decision tree to be used on the unseen test data.

Figure 19.10 shows the corresponding performance improvements obtained using feature selection alone on the same three data sets. Note the error rates produced are significantly lower than before, but still quite high.

Figure 19.11 shows average performance of the combined effects of both selection and construction. In this case the selection module was run first to find

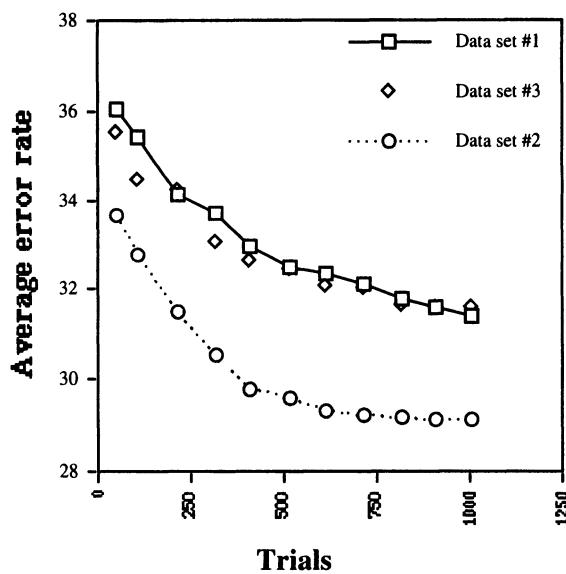


Figure 19.9 Average error rate using only feature construction.

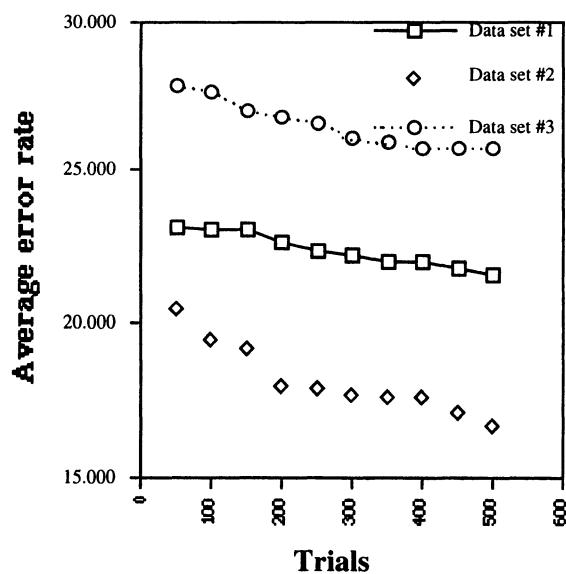


Figure 19.10 Average performance using only feature selection.

the best subset from the initial feature set, and then the construction module was invoked in an attempt to provide further improvements. As illustrated, the construction module was able to do so.

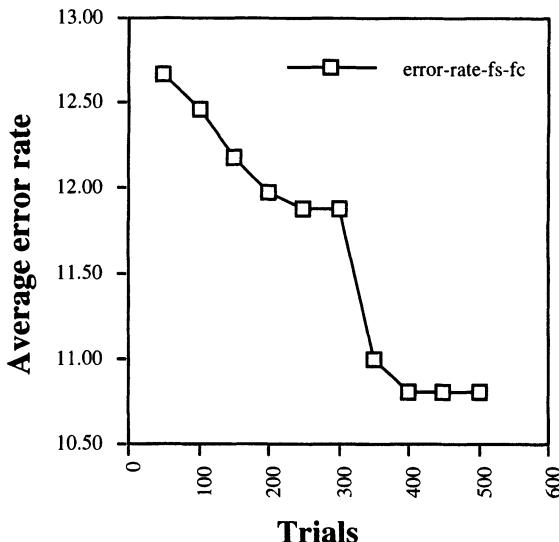


Figure 19.11 Results of running feature construction after feature selection.

Table 19.1 summarizes the results of these experiments by indicating both the size of the selected feature subset and its classification accuracy on previously unseen data. 95% confidence intervals on error rates are given in parentheses.

Table 19.1 The results of experiments using facial landmark data.

	Full Feature Set	Reduction via Selection	Reduction via Construction	Reduction via Both
No. of Feat.	105	48	73	45
Error Rate	21.9% (6.5)	16.8% (5.6)	15.3% (4.5)	12.3% (4.3)

As illustrated, the combined effects of selection and construction produced both the smallest feature subsets and the best accuracy. Interestingly, although not illustrated here, the results of first applying construction and then selection were consistently less effective. However, application patterns which started with selection and then alternated between construction and selection continued to produce transformations of the feature space in which both the number of features and the error rates were reduced as illustrated in Table 19.2.

Table 19.2 The results of representation space changes for face data.

<i>Transformation Sequence</i>	<i>Number of Features</i>	<i>Error Rate</i>	<i>Repres. Change?</i>
Original feature set	105	21.9%	N/A
FS	48	16.8%	Yes
FS-FC	45	12.3%	Yes
FS-FC-FS	24	19.4%	Yes
FS-FC-FS-FC	17	14.7%	Yes
FS-FC-FS-FC-FS	9	9.6%	Yes
FS-FC-FS-FC-FS-FC	10	7.7%	Yes
FS-FC-FS-FC-FS-FC-FS	10	7.7%	No
FS-FC-FS-FC-FS-FC-FS-FC	10	7.7%	No
FS-FC-FS-FC-FS-FC-FS-FC-FS	10	7.7%	No
FS-FC-FS-FC-FS-FC-FS-FC-FS-FC	10	7.7%	No

FS = feature selection module, and FC = feature construction.

The end result is a rather dramatic reduction in both the number of features required and the classification error rate. This data also illustrates a simple way to more fully automate this process: namely, alternating between selection and construction until a cycle produces no change to the feature set.

19.5 COMPARATIVE STUDIES

To measure how well our system compares to other methods, a series of experiments were performed. In these experiments the classification error rate was used as the main measure of the performance. The comparison was performed on two artificial problems and one real-world problem. A variety of learning algorithms were selected to represent a range of approaches and behavior. These included REDWOOD and FRINGE, two decision tree programs; GREEDY3 and GROVE, two decision list programs; AQ17-HCI, a system that generates new features from rule patterns; and LFC, a look ahead feature construction program. In these experiments we applied our system to the selected problems and compared its performance with results reported in the literature (Perez and Rendell, 1995; Ravagan and Rendell, 1993; Wnek, 1993).

Table 19.3 characterizes the problem domains. The artificial problems used were two well-studied Boolean functions: the 11-multiplexor function and the 5-parity function. The real world problem was the Pima Diabetes data set obtained from the machine learning database located at University of California at Irvine.

Table 19.4 shows the results of our experiments together with the corresponding performance for the chosen systems using ten fold cross validation runs (Weiss and Kulikowski, 1991).

Table 19.3 Characteristics of the experimental data.

<i>Concept</i>	<i>No. of attributes</i>	<i>No. of redundant attributes</i>	<i>No. of training examples</i>	<i>No. of testing examples</i>
MX-11	32	21	1600	2000
PAR-5	32	27	4000	2000
Pima Diabetes	8	0	699	77

Table 19.4 Comparative Average Error Rates.

<i>Concept</i>	<i>RED WOOD</i>	<i>FRINGE</i>	<i>GREEDY</i>	<i>GROVE</i>	<i>AQ17 HCI</i>	<i>LFC</i>	<i>GA</i>
MX-11	13.1	0.0	0.5	3.9	0.0	0.0	0.2
PAR-5	36.5	22.1	45.8	41.3	0.0	28.1	0.0
Diabetes	46.3	46.3	42.1	*	*	33.2	27.7

* indicates results not available.

The data shows our system to be very competitive on all the problems, and particularly strong on the Diabetes problem which is more representative of real-world problems.

19.6 SUMMARY AND CONCLUSIONS

Finding an appropriate representation for different problem domains is an active research area. The amount of effort necessary for proper representation of a problem can vary widely. The focus of the reported research has been on developing an autonomous system that is capable of finding an appropriate representation for a given classification problem by transforming the feature space using both feature selection and feature construction techniques.

The experimental results obtained indicate the potential advantages of using GA-based feature selection and construction techniques to improve classification performance. The reported experiments and results show a substantial improvement in the classification and recognition of real world images along with a significant reduction in the number of features used. In comparison with other approaches, this evolutionary approach seems particularly effective when applied to large, complex real-world problems.

Although the facial data sets used for our experiments are quite complex in comparison with symbolic machine learning data sets, they are still modest

from an image processing point of view. Our system needs to be tested on larger and more complex problems.

There are a number of interesting directions one might pursue in extending the GA-based methodology. One extension would be to use GAs with some background knowledge. For example, including user or expert provided estimates of the relative importance of various features for classification could further speedup the process and improve the performance.

Currently, classification performance is the only feedback used by the GA. However, data such as the actual features used by C4.5 is available and could potentially be used in a Lamarckian way to speed up the evolutionary process.

The current procedure of alternating between feature selection and feature construction is effective, but simplistic. A possible extension is to consider dynamic interaction between feature selection and feature construction based on co-evolutionary GAs currently being developed in other domains.

We are currently exploring these and other extensions in order to be able to apply our evolutionary approach to significantly larger feature space transformation problems.

Acknowledgments

The authors wish to thank Dr. Harry Wechsler and Jeffrey Huang for providing the facial data sets used in our experiments, as well as the use of facilities in the Center for Parallel and Distributed Computation.

References

- Bala, J., DeJong, K., Huang, J., Vafaie, H., and Wechsler, H. (1995). Hybrid learning using genetic algorithms and decision trees for pattern classification. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.
- Bala, J., DeJong, K., Huang, J., Vafaie, H., and Wechsler, H. (1996). Using learning to facilitate the evolution of features for recognizing visual concepts. *Evolutionary Computation*, 4(3):297–312.
- Gutta, S., Huang, J., Shah, I., Singh, D., Takacs, B., and Wechsler, H. (1995). Benchmark studies on face recognition. In *Proceedings of the International Workshop on Automatic Face and Gesture Recognition*.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.
- Kohavi, R. and John, G. (1995). Wrappers for feature subset selection. Technical Report Computer Science Department Technical Report, Stanford University.
- Koza, J. R. (1994). *Genetic Programming II*. MIT Press, Cambridge, MA.
- Perez, E. and Rendell, L. (1995). Using multidimensional projection to find relations. In Prieditis, A. and Russell, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 447–455. Morgan Kaufmann.

- Quinlan, J. (1986). The effect of noise on concept learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: an Artificial Intelligence Approach*, pages 149–166. Morgan Kaufmann.
- Ravagan, H. and Rendell, L. (1993). Lookahead feature construction for learning hard concepts. In Utgoff, P., editor, *Proceedings of the Tenth International Conference on Machine Learning*, pages 252–259. Morgan Kaufmann.
- Vafaie, H. (1997). *Using Genetic Algorithms for Restructuring Feature-Based Representation Spaces*. PhD thesis, George Mason University.
- Vafaie, H. and DeJong, K. (1993). Robust feature selection algorithms. In Koutsougeras, C., editor, *Proceedings of the Fifth International Conference on Tools with Artificial Intelligence*, pages 356–364. IEEE Computer Society Press.
- Weiss, S. and Kulikowski, C. (1991). *Computer Systems That Learn*. Morgan Kaufmann Publishers.
- Wnek, J. (1993). *Hypothesis-driven Constructive Induction*. PhD thesis, George Mason University.

20 FEATURE TRANSFORMATION BY FUNCTION DECOMPOSITION

Blaž Zupan ¹, Marko Bohanec ¹,
Janez Demšar ², and Ivan Bratko ^{2,1}

¹ J. Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
{blaz.zupan|marko.bohanec}@ijs.si

² Faculty of Computer Science and Informatics
University of Ljubljana
Tržaška 25, 1000 Ljubljana, Slovenia
{janez.demesar|bratko}@fri.uni-lj.si

Abstract: Function decomposition is a method that decomposes a dataset to an equivalent hierarchy of less complex datasets. In this chapter we show that function decomposition can contribute to feature transformation as a method for (1) discovery of a hierarchy of new features and (2) construction of features to be added to the original dataset to improve machine learning from that dataset. A particular advantage of function decomposition is its capability to identify appropriate subsets of existing features and discover functions that map each subset to a new feature. These functions are induced from examples and are not predefined in any way.

20.1 INTRODUCTION

While not explicitly intended for feature transformation, some methods for switching circuit design implicitly deal with this problem. In 1950s and 1960s, (Ashenhurst, 1952) and (Curtis, 1962) proposed a *function decomposition* method that, given a truth table of a Boolean function, develops a hierarchy of less complex tabulated Boolean functions that are preferably realizable with simple logic gates. For example, such decomposition was used to find the circuit of the Boolean function in Figure 20.1. The decomposition discovered a feature

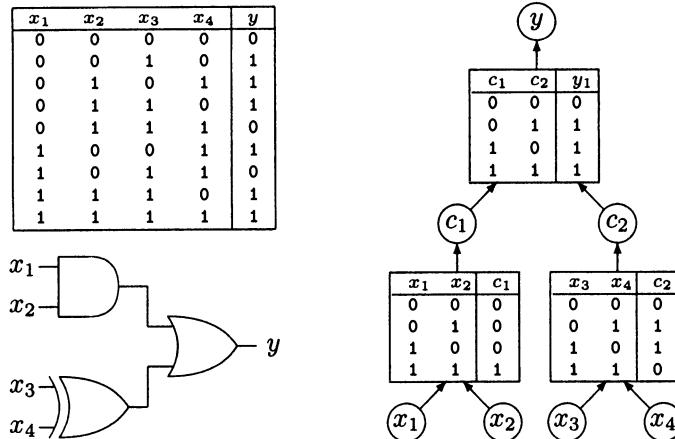


Figure 20.1 A truth table of a Boolean function (upper left), its switching circuit implementation (lower left), and corresponding feature hierarchy (right).

hierarchy with two intermediate features $c_1 = x_1 \text{ AND } x_2$ and $c_2 = x_3 \text{ XOR } x_4$, and the target concept $y = c_1 \text{ OR } c_2$. Both the hierarchy and the functions themselves are discovered by the decomposition method and are not given in advance. This is especially important from the viewpoint of feature construction, since the outputs of such functions (e.g., c_1 and c_2 from Figure 20.1) can be regarded as new features not present in the original problem description.

The basic principle of function decomposition is the following. Let a tabulated function $y = F(X)$ use a set of input features $X = x_1, \dots, x_n$. The goal is to decompose this function into $y = G(A, H(B))$, where A and B are subsets of features in X such that $A \cup B = X$. G and H are tabulated functions that are determined by the decomposition and are not predefined. Their joint complexity, determined by some complexity measure, should be lower than the complexity of F . Such a decomposition also discovers a new feature $c = H(B)$. Since the decomposition can be applied recursively on H and G , the result in general is a hierarchy of features. For each feature in the hierarchy, there is a corresponding tabulated function, such as $H(B)$, that determines the dependency of that feature on its immediate descendants in the hierarchy.

Ashenhurst-Curtis decomposition was intended for switching circuit design of completely specified Boolean functions. Recently, the decomposition was extended to handle incompletely specified Boolean functions (Perkowski and Uong, 1987; Wan and Perkowski, 1992). In the framework of feature extraction, (Ross et al., 1994b) used a set of simple Boolean functions to show the decomposition's capability to discover and extract useful features. This chapter presents an approach to feature transformation that is based on the extension of the function decomposition method by (Zupan et al., 1997). This method allows the decomposition to deal with functions that involve nominal (i.e., not necessarily binary) features, and is implemented in a system called **HINT** (Hi-

erarchy INduction Tool). Although HINT also supports noise handling (Zupan, 1997), this chapter focuses only on function decomposition mechanisms that do not explicitly deal with noise.

This chapter first introduces two basic components of function decomposition: single-step decomposition and search for the best attribute partition. Next, we show how these components can contribute to three feature transformation and selection tasks: (1) identification and removal of redundant features, (2) discovery of a hierarchy of new features and (3) construction of features to be added to the original dataset. The chapter concludes with an overview of related work and summary.

20.2 SINGLE-STEP FUNCTION DECOMPOSITION

The core of function decomposition method is a *single-step decomposition* which decomposes a tabulated function $y = F(X)$ into two possibly less complex tabulated functions G and H , so that $y = G(A, c)$ and $c = H(B)$. The resulting functions G and H have to be consistent with F . For the purpose of decomposition, a set of features X is partitioned to two disjoint subsets A and B , referred to as a *free* and *bound set*, respectively. For a given feature partition, the single-step decomposition discovers a new feature c and a tabular representation of H and G .

For example, consider a function $y = F(x_1, x_2, x_3)$ as given in Table 20.1. The input feature x_1 and the output feature y can take the values **lo**, **hi**; input features x_2 and x_3 can take the values **lo**, **med** and **hi**.

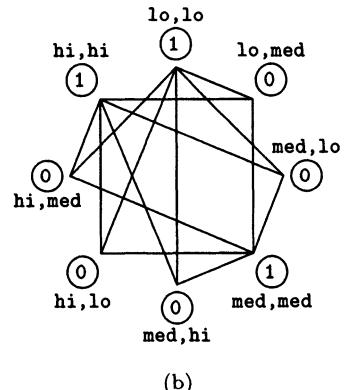
Table 20.1 Tabulated function $y = F(x_1, x_2, x_3)$.

x_1	x_2	x_3	y
lo	lo	lo	hi
lo	lo	med	lo
lo	med	lo	lo
lo	med	med	hi
lo	med	hi	lo
lo	hi	lo	lo
lo	hi	med	lo
lo	hi	hi	hi
hi	lo	lo	hi
hi	lo	med	hi
hi	med	med	hi
hi	hi	med	hi

Suppose that we want to discover a description of a new feature $c = H(x_2, x_3)$. For this purpose, we represent F by a *partition matrix* that uses the values of x_1 for row labels, and the combinations of values of x_2 and x_3 for column labels (Figure 20.2.a). Partition matrix entries with no corresponding instance in the tabulated representation of F are denoted with “-” and treated

x_2	lo	lo	med	med	med	hi	hi	hi
$x_1 \ x_3$	lo	med	lo	med	hi	lo	med	hi
lo	hi	lo	lo	hi	lo	lo	lo	hi
hi	hi	hi	-	hi	-	-	-	hi
c	1	0	0	1	0	0	0	1

(a)



(b)

Figure 20.2 Partition matrix (a) and the corresponding incompatibility graph (b) for the function from Table 20.1.

as *don't-care*. The column **lo,hi** of partition matrix has no instances and is not shown.

Each column in the partition matrix denotes the behavior of F for a specific combination of x_2 and x_3 . Columns that have pairwise equal row entries or at least one row entry is a *don't-care* are called *compatible*. The decomposition has to assign each column a label that corresponds to a value of c . If two columns are compatible, they can be assigned the same label. To preserve the consistency of original function and its decomposition, different labels have to be used for incompatible columns.

The partition matrix column labels are found by coloring an *incompatibility graph*. An example incompatibility graph is shown in Figure 20.2.b. It has a distinct node for each column of the partition matrix. Two nodes are connected if the corresponding columns of partition matrix are incompatible. For instance, the nodes **hi,hi** and **lo,med** are connected because their corresponding columns are incompatible due to the entries in the first row ($hi \neq lo$). The node labels of the incompatibility graph found by graph coloring are circled.

With optimal coloring of the incompatibility graph, the new feature c obtains a minimal set of values needed for consistent derivation of H and G from F . The optimal coloring of the graph from Figure 20.2.b requires two different colors, i.e., two abstract values for c . The tabulated functions G and H can then be derived straightforwardly from the labeled partition matrix.

The resulting decomposition is given in Figure 20.3. The following can be observed:

- The tabulated functions G and H are overall smaller than the original function F .
- By combining the features x_2 and x_3 we obtained a new feature c and the corresponding tabulated function H , which can be interpreted as

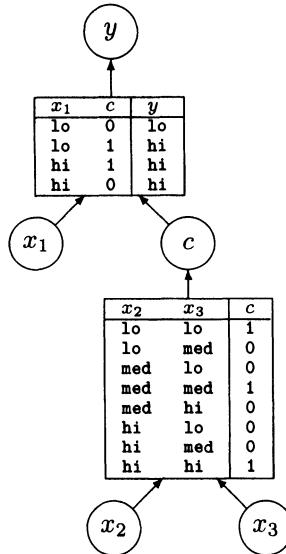


Figure 20.3 Decomposition of tabulated function from Table 20.1 using a new feature $c = H(x_2, x_3)$.

$c = \text{EQUAL}(x_2, x_3)$. Similarly, the function G that relates x_1 and c to y can be interpreted as $y = \text{MAX}(x_1, c)$.

- The decomposition generalizes some undefined entries of the partition matrix. For example, $F(\text{hi}, \text{med}, \text{hi})$ is generalized to hi because the column med, hi has the same label as columns lo, med and hi, med .

There are two problems when single-step decomposition is dealing with real-world functions: the partition matrix can become very large, and because of its incompleteness the optimal coloring of the corresponding incompatibility graph can be prohibitively complex. HINT solves the first problem by avoiding the explicit construction of partition matrix and deriving the incompatibility graph directly from the tabulated function (Zupan, 1997). Partition matrix is thus only a formal construct used for the explanation and analysis of decomposition algorithm. For the problem of coloring, HINT uses a suboptimal but efficient heuristic coloring algorithm (Zupan et al., 1997; Zupan, 1997) based on the Color Influence Method by (Perkowski and Uong, 1987) and (Wan and Perkowski, 1992).

20.3 FINDING THE BEST FEATURE PARTITION

So far, we have assumed that a partition of the features to free and bound sets is given. However, for each function F there are many possible partitions, each one yielding a different feature c and a different pair of functions G and H . In feature transformation, it is important to identify partitions that lead to simple

but useful new features. Typically, we prefer features with a small number of values, and those that yield functions G and H of low complexity.

The simplest measure for finding good feature partitions is the number of values required for the new feature c . That is, when decomposing F , a set of candidate partitions is examined and the one that yields c with a smallest set of possible values is selected for decomposition.

An alternative measure for the selection of partitions can be based on the complexity of functions. Let $I(F)$ denote a number of bits to encode some function $y = F(X)$:

$$I(F) = \log_2 |D_y| \prod_{x \in X} |D_x|$$

where D_x and D_y are value domains of features x and y , while $|D_x|$ and $|D_y|$ denote their cardinality. There are some other similar measures (Ross et al., 1994b; Biermann et al., 1982) that assume a different encoding for F and thus yield different $I(F)$, but they only slightly affect the performance of decomposition (Zupan, 1997).

The best partition is then the one that minimizes the overall complexity of newly discovered functions H and G , i.e., the partition with minimal $I(H) + I(G)$. The complexity of G and H should be lower than that of F , and decomposition takes place only if $I(H) + I(G) < I(F)$ for the best partition. If the latter criterion is satisfied, the function F is called *decomposable* with respect to the given attribute partition.

Table 20.2 provides an example and lists both above mentioned partition selection measures for the function from Table 20.1. This function can be decomposed in three different ways, where by both measures the partition $A = \{x_1\}$ and $B = \{x_2, x_3\}$ is favorable. Regarding the complexity measure I , this partition is also the only one for which F is decomposable (note that $I(F) = 18.0$).

Table 20.2 Partition selection measures for the function from Table 20.1.

Partition	$ c $	$I(H) + I(G)$	decomposable
$A = \{x_1\}, B = \{x_2, x_3\}$	2	13.0	yes
$A = \{x_2\}, B = \{x_1, x_3\}$	4	24.0	no
$A = \{x_3\}, B = \{x_1, x_2\}$	4	24.0	no

The number of possible partitions increases exponentially with the number of input features. To limit the complexity of search, the decomposition should examine only a subset of partitions. In HINT, the partitions examined are only those with less than b features in the bound set, where b is a user-defined parameter usually set to 2 or 3.

20.4 REDUNDANCY DISCOVERY AND REMOVAL

A special characteristic of function decomposition is its capability to identify and remove redundant features and/or their values. Namely, whenever a feature $c = H(x_i)$ is discovered such that c uses only a single value (i.e., H is constant function), the feature x_i is redundant and can be removed from the dataset. Similarly, whenever a feature $c = H(x_i)$ uses less values than x_i , the original feature x_i can be replaced by c .

Both these dataset transformations are particularly useful for data pre-processing, since they can reduce the size of the problem and increase the coverage of the problem space by the learning examples. HINT removes redundant features in the increased order of their relevance: the less relevant features are checked first, and if any type of redundancy mentioned above is discovered, the dataset is accordingly transformed. The features are estimated by a ReliefF measure of relevance (Kononenko, 1994). ReliefF estimates the features according to how well they distinguish among the instances that are close to each other. The relevance of feature x is then

$$W(x) = P(\text{different value of } x \mid k\text{-nearest instances with different } y) - P(\text{different value of } x \mid k\text{-nearest instances with same } y)$$

We use the version of ReliefF which determines the feature's relevancy based on at most 200 arbitrary selected examples, and which for every example examines $k = 5$ nearest instances of the same and of the different value for y .

20.5 DISCOVERING FEATURE HIERARCHIES

With all the above components, decomposition enables the discovery of feature hierarchies. Given a dataset, it is first pre-processed to remove redundant features and their values. Next, a single-step decomposition is used to decompose the pre-processed tabulated function F to G and H . This process is recursively repeated on G and H until they can not be decomposed further, i.e., their further decomposition would increase the overall complexity of resulting functions.

Let us illustrate this process by several examples. The first example is a well-known machine learning problem MONK1 (Thrun et al., 1991). The dataset uses six 2 to 4-valued input features (x_1 to x_6) and contains 124 (of 435 possible) distinct learning examples that partially define the target concept $x_1 = x_2 \text{ OR } x_5 = 1$. HINT develops a feature hierarchy (Figure 20.4) that

1. correctly excludes irrelevant input features x_3 , x_4 , and x_6 ,
2. transforms x_5 to x'_5 by mapping four values of x_5 to only two values of x'_5 ,
3. includes a new feature c and its tabular representation for $x_1 = x_2$, and
4. relates c and x'_5 with a tabular representation of the OR function.

In other words, the resulting hierarchy correctly represents the target concept.

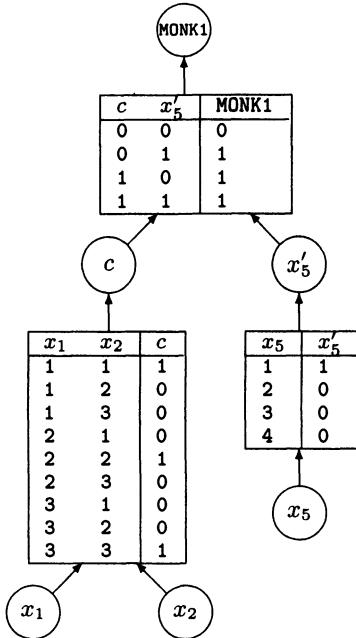


Figure 20.4 MONK1 feature hierarchy as discovered by HINT.

Similarly as MONK1, the MONK2 learning problem (Thrun et al., 1991) uses the same set of input features but defines the concept: $x_i = 1$ for exactly two choices of $i \in \{1, 2, \dots, 6\}$. The dataset contains 172 (of 435 possible) distinct learning examples. Although the discovered structure (Figure 20.5) does not directly correspond to the original concept definition, it correctly reformulates the target concept by introducing features that count the number of ones in their arguments. Also note that all input features that use more than two values are replaced by new binary features.

For a real-world example, consider a HOUSING dataset from a management decision support system for allocating housing loans (Bohanec et al., 1996). This system was developed for the Housing Fund of Slovenia and used since 1991 in 13 floats of loans with a total value of approximately 90 million ECU. Basically, its task is to rank applicants into one of nine priority classes based on 12 input features. The system uses a hierarchical decision model.

For this experiment, we wanted to assess the ability of decomposition to reconstruct the system's model given a random selection of classified cases. First, it was observed that in most cases when HINT was given 8000 or more learning examples (4% coverage of problem space), it discovered the same feature hierarchy (Figure 20.6). When presented to a domain expert, the discovered features were found meaningful. For instance, the feature c_4 represents the present housing status of the applicant. Features c_2 , c_6 , and c_8 respectively specify the way of solving the housing problem, applicant's current status, and his/her social and health conditions. Thus, the structure of the original model

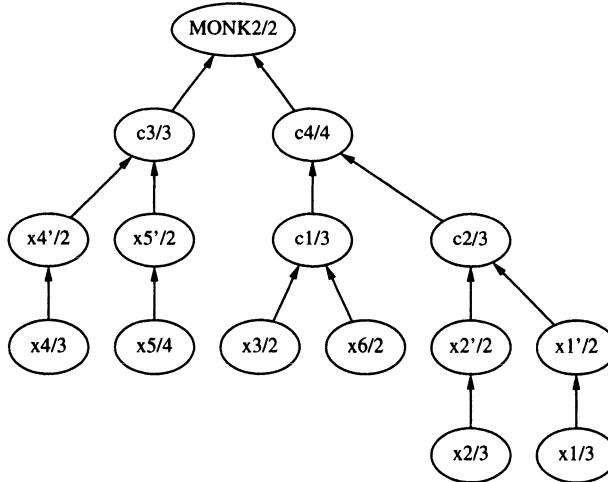


Figure 20.5 The feature hierarchy discovered for MONK2. Each node gives a name of the feature and cardinality of its set of values.

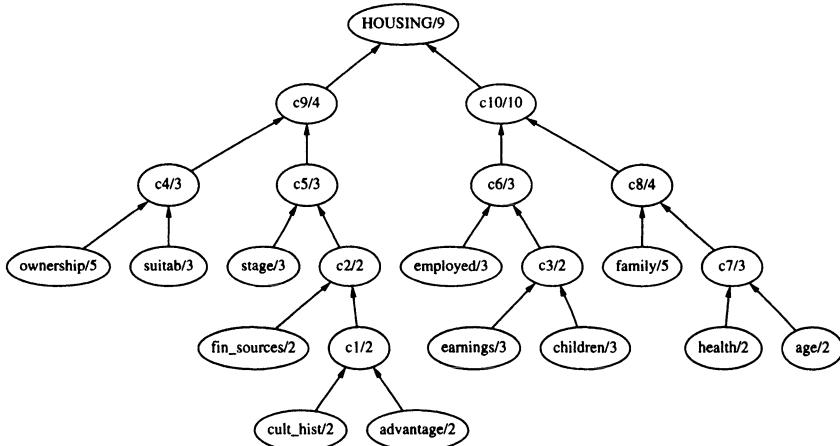


Figure 20.6 The feature structure discovered for housing loan allocation problem. Each node gives a name of the feature and cardinality of its set of values.

was successfully reconstructed. This reconstruction takes about 2.5 minutes of processor time on a HP J210 Unix workstation.

The quality of the functions discovered for housing problem was assessed by classification of the remaining examples that completely cover this domain but were not included in the learning set. The generalization of HINT was assessed as a learning curve (Figure 20.7) and compared to that of the state-of-the-art learning tool C4.5 (Quinlan, 1993) that induces decision trees. C4.5 was run with default options except for `-m1`, by which C4.5 trees were made

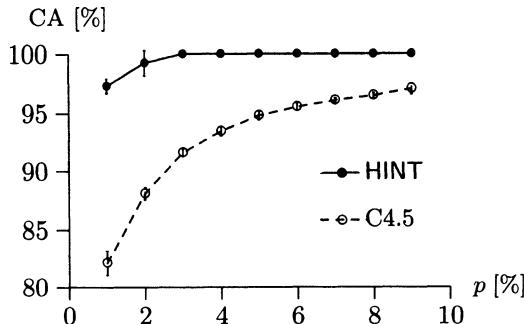


Figure 20.7 Averaged classification accuracy of HINT and C4.5 for 10 experiments with learning sets consisting of a random sample of p percents of the HOUSING problem space.

consistent with the original dataset. For this domain, HINT generalizes well as it achieves 100% classification accuracy by using learning sets that cover 3 or more percents of the problem space.

Housing problem domain can be viewed as a typical representative of practical domains in the field of decision making that HINT was tested on. Similar tests and experimental conclusion were for instance obtained by the rediscovery of hierarchical decision models for nursery school applications, job application evaluation, and performance evaluation of public enterprises (Zupan, 1997).

20.6 FEATURE CONSTRUCTION

Function decomposition can also be used for feature construction. This is defined as a process of augmenting the feature space by inferring or creating additional features. We here show how the single-step decomposition can be used both for finding appropriate combinations of original features and using them to construct new features which are then added to the original dataset. Again, the new features that use small number of values are preferred.

For example, consider again the dataset from Table 20.1. Suppose we want to augment it with a single new feature. For this purpose, single-step decomposition examines all pairs of original input features and for each pair derives a new feature. A new feature that has the fewest values is preferred. Such feature is $c = H(x_2, x_3)$, and is defined by the example set from Figure 20.3. Function H can now be used for each example from Table 20.1 to obtain the corresponding value for c ; such augmented dataset is shown in Table 20.3.

Obviously, an original dataset can be augmented with more than just a single new feature. We propose a schema where, using the single-step decomposition, m new features are added to a dataset: a candidate set of combinations of the original features are examined, and corresponding m new features that have the fewest values are selected. Ties are resolved arbitrarily. Only original features are used to generate new features. Note also that in the process no hierarchy of features is constructed. The feature construction and corresponding aug-

Table 20.3 A dataset from Table 20.1 augmented with a new feature $c = H(x_2, x_3)$.

x_1	x_2	x_3	c	y
lo	lo	lo	1	hi
lo	lo	med	0	lo
lo	med	lo	0	lo
lo	med	med	1	hi
lo	med	hi	0	lo
lo	hi	lo	0	lo
lo	hi	med	0	lo
lo	hi	hi	1	hi
hi	lo	lo	1	hi
hi	lo	med	0	hi
hi	med	med	1	hi
hi	hi	med	0	hi

mentation of the original dataset results in a new dataset which may then be further processed by some machine learning algorithm.

We evaluate this schema on the above-mentioned domains. For new features, only those that depend only on two original input features are considered. For MONK1 and MONK2 domains, the benefit of adding new features was assessed by a 10-fold cross validation. For HOUSING, 10 experiments were performed with a learning set consisting of a random sample of 1% of the problem space, while the test set consisted of the remaining 99%. Each learning set was first augmented with m new features as described above, and then given to C4.5 (Quinlan, 1993) which was used to induce the target concept from the augmented dataset. Again, C4.5 was run with default options except for `-m1`. The induced decision tree was then tested on a test set, which was also augmented by the same set of new features, i.e., using their definition as derived from the learning set.

In all the cases, a considerable increase of classification accuracy (Figure 20.8) and considerable decrease of the size of induced decision trees (Figure 20.9; error bars in both graphs indicate standard deviations) were observed when new features are added. In other words, adding new features improves both the classification accuracy and the transparency of decision trees. For example, for MONK1, HINT discovered the feature $c = \text{EQUAL}(x_1, x_2)$ which enabled C4.5 to construct the decision tree with only five nodes:

```

if c=1 then y=1
else
  if x5=1 then y=1
  else y=0

```

This decision tree was also substantially less complex than the decision trees with an average of 77 nodes constructed from the original MONK1 data.

The results indicate that function decomposition discovers features that are relevant for these domains. It is further interesting to note how well a simple feature selection criterion based on the number of new feature's values performs

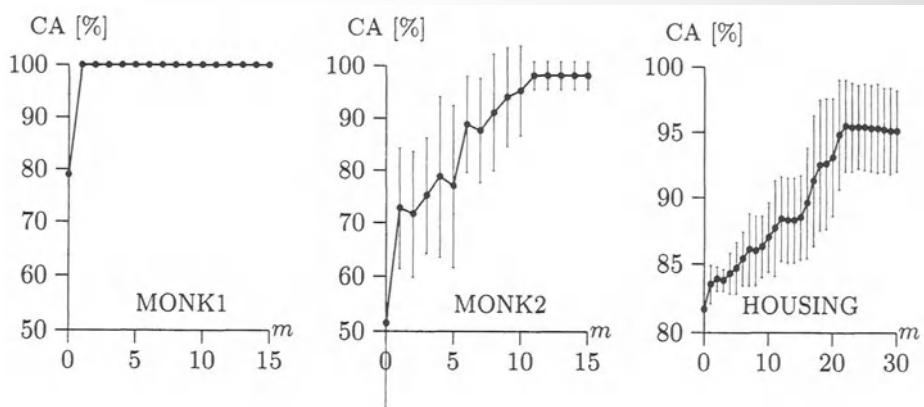


Figure 20.8 Classification accuracy of C4.5 as a function of number of new features m added to MONK1, MONK2, and housing loans dataset.

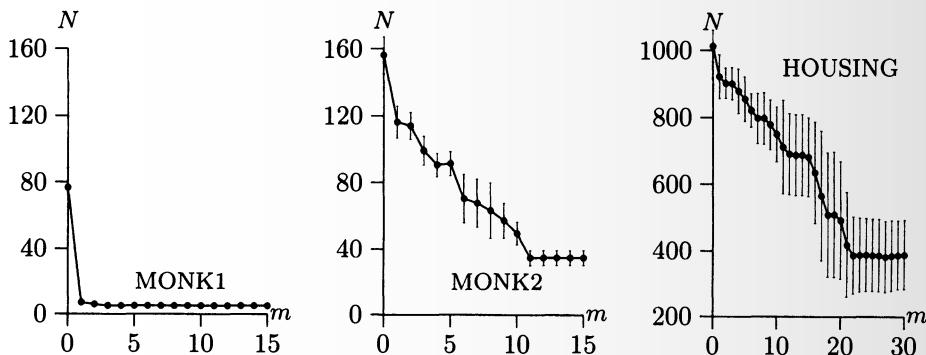


Figure 20.9 Number of nodes N of C4.5-induced decision trees as a function of number of new features m added to MONK1, MONK2, and housing loans dataset.

in terms of yielding augmented datasets that enable C4.5 to derive classifiers with high classification accuracies.

20.7 RELATED WORK

A learning method that uses feature hierarchies has been proposed already by a pioneer of artificial intelligence, (Samuel, 1967). His method used a signature table system, which is essentially the same representation mechanism as the one used in this chapter. However, Samuel's method was able to learn only functions, requiring the hierarchy to be given in advance. Samuel's approach was later studied and improved by (Biermann et al., 1982) but still relied on the predefined feature hierarchy.

While, within machine learning, Samuel and Biermann et al. may be the first to realize the power of decomposition, the fundamentals of the approach

were defined earlier in the area of switching circuit design. (Curtis, 1962) reports that in the late 1940's and 1950's several switching circuit theorists considered this subject and (Ashenhurst, 1952) reported on a unified theory of decomposition of switching functions. Most of related work of those times is reported and reprinted in (Curtis, 1962), where he compares the decomposition approach to other switching circuit design approaches and further formalizes and extends the decomposition theory. Curtis' method is defined over Boolean variables and requires a set of examples that completely cover the problem space.

Recently, the Ashenhurst-Curtis approach was substantially improved by research groups of M. A. Perkowski, T. Luba, and T. D. Ross. (Perkowski and Uong, 1987) and (Wan and Perkowski, 1992) propose a graph coloring approach to the decomposition of incompletely specified switching functions. A different approach is presented by (Luba and Selvaraj, 1995). Their decomposition algorithms are able to generalize. Generalization of function decomposition when applied to a set of simple Boolean functions was studied by (Ross et al., 1994b) and (Goldman, 1994). Goldman also indicates that the decomposition approach to switching function design might be termed knowledge discovery since features not previously anticipated can be discovered.

The above mentioned decomposition approaches typically deal with noiseless data and Boolean features. Various extensions towards dealing with nominal-valued features were proposed by (Biermann et al., 1982), (Luba, 1995), (Zupan et al., 1997), and (Files et al., 1997). To handle noisy data, a minimal-error decomposition was recently proposed (Zupan, 1997). It is based on a representation of learning examples with class distributions and uses successive column merging of partition matrix, so that the expected error of classification is minimized. For the decomposition of real-valued functions some preliminary methods were proposed by (Ross et al., 1994a) and (Demšar et al., 1997).

There are other machine learning approaches that either use or construct feature hierarchies. DUCE (Muggleton, 1987; Muggleton, 1990) uses transformation operators to compress the given examples by successive generalization and feature construction. Query-based PAC-learning of tabulated functions within feature hierarchies is described by (Tadepalli and Russell, 1998). Queries are also used in PAC-learning described by (Bshouty et al., 1995). Their algorithm identifies both concept structures and their associated tabulated functions, but can deal only with Boolean functions with symmetric and constant fan-in gates. Within PAC-learning, (Hancock et al., 1994) learn non-overlapping perceptron networks from examples and membership queries. A structured induction approach (Shapiro and Niblett, 1982; Michie, 1995) is based on a manual decomposition of the problem and an expert-assisted selection and classification of examples to construct functions that define intermediate features in the hierarchy.

The feature hierarchy has also been used by a multi-attribute decision support expert system shell DEX (Bohanec and Rajković, 1990) which has its roots in DECMAK methodology (Efstathiou and Rajković, 1979; Bohanec et al.,

1983). There, a tree-like structure of variables is defined by an expert, and several tools assist in the acquisition of decision tables.

The above related work refers primarily to the use and discovery of feature hierarchies. As outlined in this chapter, a function decomposition may also be useful to manufacture features that are then used by some other machine learning algorithms. To the best of our knowledge, there has been no related work that would use function decomposition in this way.

20.8 SUMMARY

The distinctive property of function decomposition is that it can discover new features together with their corresponding functions that are not predefined in any way. As such and in the framework of feature transformation, function decomposition is a promising approach to discover and construct new features that are either added to the original dataset, or transform this dataset to a hierarchy of less complex datasets. The experiments show that decomposition can:

- discover useful and relevant features,
- develop relevant feature hierarchies, and
- identify and remove redundant features and/or their values.

Furthermore, we have shown that discovery of feature hierarchies by function decomposition may generalize well, which is especially important when using function decomposition within machine learning framework. Also, adding the features derived by function decomposition to the original datasets can improve the performance of other machine learning algorithms both in terms of accuracy and size of induced classifiers.

Acknowledgments

This chapter is a revised and extended version of the article ‘Feature transformation by function decomposition’ by the same authors, *IEEE Intelligent Systems*, March/April 1998 (© 1998 IEEE). We acknowledge the kind permission of IEEE to use portions of this article.

References

- Ashenhurst, R. L. (1952). The decomposition of switching functions. Technical report, Bell Laboratories BL-1(11), pages 541–602.
- Biermann, A. W., Fairfield, J., and Beres, T. (1982). Signature table systems and learning. *IEEE Trans. Syst. Man Cybern.*, 12(5):635–648.
- Bohanec, M., Bratko, I., and Rajković, V. (1983). An expert system for decision making. In Sol, H. G., editor, *Processes and Tools for Decision Support*. North-Holland.
- Bohanec, M., Cestnik, B., and Rajković, V. (1996). A management decision support system for allocating housing loans. In Humphreys, P., Bannon, L.,

- McCosh, A., and Migliarese, P., editors, *Implementing System for Supporting Management Decisions*, pages 34–43. Chapman & Hall, London.
- Bohanec, M. and Rajković, V. (1990). DEX: An expert system shell for decision support. *Sistemica*, 1(1):145–157.
- Bshouty, N. H., Hancock, T. R., and Hellerstein, L. (1995). Learning boolean read-once formulas over generalized bases. *Journal of Computer and System Sciences*, 50(3):521–542.
- Curtis, H. A. (1962). *A New Approach to the Design of Switching Functions*. Van Nostrand, Princeton, N.J.
- Demšar, J., Zupan, B., Bohanec, M., and Bratko, I. (1997). Constructing intermediate concepts by decomposition of real functions. In van Someren, M. and Widmer, G., editors, *Proc. European Conference on Machine Learning, ECML-97*, pages 93–107, Prague. Springer.
- Efstathiou, J. and Rajković, V. (1979). Multiattribute decisionmaking using a fuzzy heuristic approach. *IEEE Trans. on Systems, Man and Cybernetics*, 9:326–333.
- Files, C., Drechsler, R., and Perkowski, M. (1997). Functional decomposition of MVL functions using multi-valued decision diagrams. In *International Symposium on Multi-Valued Logic*.
- Goldman, J. A. (1994). Pattern theoretic knowledge discovery. In *Proc. the Sixth Int'l IEEE Conference on Tools with AI*.
- Hancock, T. R., Golea, M., and Marchand, M. (1994). Learning nonoverlapping perceptron networks from examples and membership queries. *Machine Learning*, 16(3):161–183.
- Kononenko, I. (1994). Estimating attributes. In Bergadano, F. and Raedt, L. D., editors, *Proc. of the European Conference on Machine Learning (ECML-94)*, pages 171–182. Springer-Verlag.
- Luba, T. (1995). Decomposition of multiple-valued functions. In *25th Intl. Symposium on Multiple-Valued Logic*, pages 256–261, Bloomington, Indiana.
- Luba, T. and Selvaraj, H. (1995). A general approach to boolean function decomposition and its application in FPGA-based synthesis. *VLSI Design*, 3(3–4):289–300.
- Michie, D. (1995). Problem decomposition and the learning of skills. In Lavrač, N. and Wrobel, S., editors, *Machine Learning: ECML-95*, Notes in Artificial Intelligence 912, pages 17–31. Springer-Verlag.
- Muggleton, S. (1987). Structuring knowledge by asking questions. In Bratko, I. and Lavrač, N., editors, *Progress in Machine Learning*, pages 218–229. Sigma Press.
- Muggleton, S. (1990). *Inductive Acquisition of Expert Knowledge*. Addison-Wesley, Workingham, England.
- Perkowski, M. and Uong, H. (1987). Automatic design of finite state machines with electronically programmable devices. In *Record of Northcon '87*, pages 16/4.1–16/4.15, Portland, OR.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

- Ross, T. D., Goldman, J. A., Gadd, D. A., Noviskey, M. J., and Axtell, M. L. (1994a). On the decomposition of real-valued functions. In *3rd Int'l Workshop of Post-Binary VLSI Systems*.
- Ross, T. D., Noviskey, M. J., Gadd, D. A., and Goldman, J. A. (1994b). Pattern theoretic feature extraction and constructive induction. In *Proc. ML-COLT '94 Workshop on Constructive Induction and Change of Representation*, New Brunswick, New Jersey.
- Samuel, A. (1967). Some studies in machine learning using the game of checkers II: Recent progress. *IBM J. Res. Develop.*, 11:601–617.
- Shapiro, A. D. and Niblett, T. (1982). Automatic induction of classification rules for a chess endgame. In Clarke, M. R. B., editor, *Advances in Computer Chess 3*, pages 73–92. Pergamon, Oxford.
- Tadepalli, P. and Russell, S. (1998). Learning from examples and membership queries with structured determinations. *Machine Learning* (to appear).
- Thrun et al., S. B. (1991). A performance comparison of different learning algorithms. Technical report, Carnegie Mellon University CMU-CS-91-197.
- Wan, W. and Perkowski, M. A. (1992). A new approach to the decomposition of incompletely specified functions based on graph-coloring and local transformations and its application to FPGA mapping. In *Proc. of the IEEE EURO-DAC '92*, pages 230–235, Hamburg.
- Zupan, B. (1997). *Machine learning based on function decomposition*. PhD thesis, University of Ljubljana. Available at <http://www-ai.ijs.si/BlazZupan/papers.html>.
- Zupan, B., Bohanec, M., Bratko, I., and Demšar, J. (1997). Machine learning by function decomposition. In D. H. Fisher, J., editor, *Proc. Fourteenth International Conference on Machine Learning*, pages 421–429, San Mateo, CA. Morgan Kaufmann.

VI Applications of Feature Transformation

21 CONSTRUCTIVE INDUCTION OF CARTESIAN PRODUCT ATTRIBUTES

Michael J. Pazzani

Department of Information and Computer Science
University of California, Irvine
Irvine, California 92697

pazzani@ics.uci.edu

Abstract: Constructive induction is the process of changing the representation of examples by creating new attributes from existing attributes. In classification, the goal of constructive induction is to find a representation that facilitates learning a concept description by a particular learning system. Typically, the new attributes are Boolean or arithmetic combinations of existing attributes and the learning algorithms used are decision trees or rule learners. We describe the construction of new attributes that are the Cartesian product of existing attributes. We consider the effects of this operator on three learning algorithms and compare two different methods for determining when to construct new attributes with this operator.

21.1 INTRODUCTION

In classification learning, a variety of methods have been explored for changing the representation of examples to improve the accuracy of learned concepts. One method is to select a subset of the attributes(Caruana and Freitag, 1994; John et al., 1994) . In the subset selection problem, examples are represented by a set of attributes (e.g., {height, age, shoe-size, weight}) and the goal is to find a subset of these attributes (e.g., {height, age, weight}) such that when a classifier is applied to examples represented by a subset of the attributes the accuracy of the classifier is maximized. This problem has also recently received attention in the context of Bayesian classifiers(Langley and Sage, 1994) and nearest neighbor algorithms(Aha and Bankert, 1995) . The subset selection problem has also been studied in pattern recognition (e.g., Kittler, 1986). It has proved important in applying machine learning to real world

tasks(Salzberg et al., 1995) in which the attributes were not selected by hand by experts. Constructive induction is another method for improving the accuracy of learned concepts. Constructive induction is the process of changing the representation of examples by creating new attributes from existing attributes. The new attributes are typically Boolean or arithmetic combinations of existing attributes and the learning algorithms used are typically decision trees or rule learners(Matheus and Rendell, 1989; Ragavan and Rendell, 1993; Sanger, 1992). Here, we consider the use of a different operator for constructive induction: the Cartesian product. We will use the term “joining” to refer to the process of creating a new attribute whose values are the Cartesian product of two other attributes. For example, if `height` with values `tall` and `short` were joined to `weight` with values `heavy` and `light`, a new attribute would be formed `height_weight` with values `tall_heavy`, `tall_light`, `short_heavy` and `short_light`.¹

In this chapter, we first propose an algorithm based on the “wrapper model” (John et al., 1994) for the construction of Cartesian product attributes and describe the effect of construction induction of Cartesian product attributes on three learning algorithms: a naive Bayesian classifier (Duda and Hart, 1973); ID3, a decision tree learner(Quinlan, 1986); and PEBLS, a nearest neighbor algorithm(Cost and Salzberg). We show experimentally that constructing Cartesian product attributes is not generally beneficial to decision tree learners but can be beneficial to Bayesian classifiers and PEBLS. Finally, we compare the wrapper approach to an alternative for detecting attribute dependencies.

21.2 A WRAPPER APPROACH FOR CREATING CARTESIAN PRODUCT ATTRIBUTES

A template of the general algorithm for changing example representation is shown in Table 21.1. In summary, it starts with the provided example representation and performs hill-climbing search to find a new representation by applying two operators:

- a. Replacing a pair of attributes in the representation with a new attribute that is the Cartesian product of the two attributes.
- b. Deleting an attribute used by the representation.

There are several items that require further elaboration and explanation about the algorithm in Table 21.1. First, to determine which single change of representation is best, the accuracy of the learned description that would result from this representation is estimated by using leave-one-out cross validation on the training data. We choose this particular method of accuracy estimation because it may be computed quickly for the naive Bayesian classifier and the nearest neighbor algorithm. Second, we terminate the process of changing the example representation when no change results in an improvement over the current representation as determined by leave-one-out cross-validation. Third, a new attribute replaces the attributes from which it was formed. This is

Table 21.1 A template of the algorithm for changing representation.

```

0. Start with the initial example representation
1. Consider joining each pair of attributes
2. Consider deleting each attribute
3. If there is a "good" change of representation
Then: Make the "best" change of representation and go to 1
Else: Return the current example representation.

```

done since some learners (e.g., Bayesian classifiers) assume independence of attributes within each class and the Cartesian product of two attributes is clearly dependent on the two attributes.² Fourth, we consider deleting attributes as well as joining attributes since this can easily be accomplished at the same time and it is useful both in eliminating original irrelevant attributes and attributes created with constructive induction. Finally, we use hill-climbing search to explore the search space, since an exhaustive search would not be feasible³. In the future we may explore other search algorithms such as best-first search or genetic algorithms but in this chapter we provide evidence that the Cartesian product operator is useful in constructive induction and that hill-climbing search guided by accuracy estimation can improve the accuracy of some learning algorithms.

There are two reasons for introducing the Cartesian product operator in constructive induction. First, by rerepresenting examples, some concepts that are not capable of being expressed in some learners representation language can now be expressed. For example, the naive Bayesian classifier is limited in expressiveness in that it cannot learn nonlinearly separable functions(Langley and Sage, 1994; Rachlin et al., 1994).In addition, it cannot learn with 100% accuracy some *m-of-n* concepts(Kohavi, 1995). Second, the naive Bayesian classifier and PEBLS make independence assumptions that are violated by some data sets. When attributes are treated individually, joint probabilities are computed by taking the product of individual probabilities estimated from data. When two or more attributes are replaced by the Cartesian product of the attributes, the joint probability is estimated from the training data.

We call the constructive induction algorithm, BSEJ, for Backward Sequential Elimination and Joining by analogy with the backward sequential elimination algorithm for attribute selection. In the remainder of this chapter, we first report on experiments using BSEJ on both naturally occurring and artificial datasets with three learning algorithms and explain why it is effective with some algorithms but not others. Finally, we compare BSEJ to related work and elaborate on several design choices made in BSEJ:

- Using accuracy estimation rather than a test for conditional independence to decide which attributes to join.
- Creating Cartesian product attributes rather than deleting attributes.

Table 21.2 Summary of domains used in the experiments.

<i>Domain</i>	<i>Training Examples</i>	<i>Attributes</i>
glass	150	9
krkp	300	36
mushroom	800	22
voting	300	16
credit	250	15
horse-colic	200	21
iris	100	4
pima-diabetes	500	8
wine	125	13
wisc-cancer	500	9
xor	128	8
4-Parity	128	8

- Searching backward rather than forward.

21.3 EXPERIMENTS

To investigate the effects of BSEJ on the three learning algorithms we will use 10 databases from the UCI Repository of machine learning databases and two artificial concepts. The artificial concepts used are exclusive-or with two relevant attributes and 6 irrelevant attributes and parity with 4 relevant attributes and 4 irrelevant attributes. Both artificial concepts are cases where there are dependencies among attributes. The parity concept should be more difficult for all algorithms to deal with since four attributes are involved in the interaction rather than just two in exclusive-or. The experiments on the naturally occurring databases are included to determine whether the problems that BSEJ is intended to address occur in practice as well as in artificial concepts.

The domains we used are summarized in Table 21.2. In each experiment, we ran 24 trials of randomly selecting training examples, running the learning algorithm with and without the BSEJ algorithm for constructive induction, and evaluating the accuracy of the algorithm on a test set consisting of all examples not in the training set. We will use a paired, two-tailed t-test at least at the .05 level to determine whether BSEJ has a significant effect on the accuracy of each learning algorithm.

21.3.1 Naive Bayesian Classifier

The Bayesian classifier(Duda and Hart, 1973) is a probabilistic method for classification. It can be used to determine the probability that an example j belongs to class C_i given values of attributes of an example represented as a set of n nominally-valued attribute-value pairs of the form $A_1 = V_{1,j}$:

Table 21.3 The accuracy of the naive Bayesian classifier with and without constructive induction.

<i>Domain</i>	<i>Naive</i>	<i>BSEJ</i>
xor	.461	1.00 +
4-parity	.424	.475 +
glass	.417	.768 +
krkp	.868	.931 +
mushroom	.940	.992 +
voting	.904	.920 +
credit	.840	.833
horse-colic	.810	.802
iris	.931	.938
pima	.755	.749
wine	.980	.975
wisc-cancer	.973	.971

A “+” indicates that using BSEJ results in a significant increase in accuracy. There are no significant decreases in accuracy.

$$P(C_i | A_1 = V_{1,j} \& \dots A_n = V_{n,j})$$

If the attributes are independent, this probability is proportional to:

$$P(C_i) \prod_k P(A_k = V_{k,j} | C_i) \quad (21.1)$$

This formula is well suited for learning from data, since the probabilities $\hat{P}(C_i)$ and $\hat{P}(A_k = V_{k,j} | C_i)$ may be estimated from the training data. To determine the most likely class of a test example, the probability of each class is computed. A classifier created in this manner is sometimes called a simple (Langley, 1993) or naive(Kononenko, 1990) Bayesian classifier.

Table 21.3 shows the accuracy of the Bayesian classifier with and without the constructive induction algorithm.

The exclusive-or problem is a dramatic example of why adding Cartesian product attributes results in a significant increase in accuracy. For simplicity, in this discussion we'll assume there are 2 relevant attributes, A_1 and A_2 , and only 1 irrelevant attribute, A_3 . Without constructive induction, the probability of class C_i is computed to be proportional to

$$\hat{P}(A_1 = V_{1,j} | C_i) \hat{P}(A_2 = V_{2,j} | C_i) \hat{P}(A_3 = V_{3,j} | C_i) \hat{P}(C_i) \quad (21.2)$$

However, on every trial the BSEJ algorithm forms the Cartesian product of A_1 and A_2 and the probability of class C_i is proportional to:⁴

$$\hat{P}(A_1 = V_{1,j} \& A_2 = V_{2,j} | C_i) \hat{P}(A_3 = V_{3,j} | C_i) \hat{P}(C_i) \quad (21.3)$$

A more accurate classifier can result from forming the Cartesian product of A_1 and A_2 . For example, consider classifying an example in which A_1 , A_2 and A_3 were equal to 1. In this case, $\hat{P}(A_1 = 1|C_i = \text{True})$, $\hat{P}(A_2 = 1|C_i = \text{True})$, $\hat{P}(A_3 = 1|C_i = \text{True})$, $\hat{P}(A_1 = 1|C_i = \text{False})$, $\hat{P}(A_2 = 1|C_i = \text{False})$, $\hat{P}(A_3 = 1|C_i = \text{False})$, $\hat{P}(C_i = \text{True})$, and $\hat{P}(\text{Class} = \text{False})$ would all be close to 0.5 if estimated from randomly selected examples, and the naive Bayesian classifier that did not use a Cartesian product attribute would perform at near chance levels. However, if Equation 21.3 were used, the classifier would be accurate since $\hat{P}(A_1 = 1 \& A_2 = 1|C_i = \text{True})$ would be 1 and $\hat{P}(A_1 = 1 \& A_2 = 1|C_i = \text{False})$ would be 0.

Although in this simple example, Cartesian product attributes work out well, in practice, it may cause problems. If independent attributes are joined and the probabilities of joined attributes are estimated from training data, a less accurate classifier may result because the probability estimates of the joined attributes are less reliable than the estimates of individual attributes. There is a trade-off between inaccuracies caused by not having enough data to reliably estimate joint probabilities from the data and using more accurate estimates of individual probabilities, but incorrectly computing the joint probability as the product of dependent individual probabilities. The wrapper approach is one way of dealing with this trade-off since it only joins attributes when the joint probability is estimated accurately enough to improve the accuracy of the learned concept as measured by leave-one-out cross validation.

Forming Cartesian product attributes results in a significant increase in accuracy on four of the naturally occurring domains. This suggests that there are violations of the conditional independence assumption made by the naive Bayesian classifier in naturally occurring domains, and that BSEJ can detect and correct for violations of this assumption by changing the example representation.

The parity example illustrates a potential shortcoming of the hill-climbing approach to building Cartesian product attributes. Although constructive induction results in a small statistically significant increase in accuracy, it is not better than simply guessing randomly. In this case, a Cartesian product of four attributes is needed and none of the Cartesian products of two or three of these attributes appear beneficial to the evaluation metric. Although it would be possible to consider joining three (or more attributes) in one step, the computational complexity makes it impractical for most databases.

21.3.2 ID3

To conserve space, we'll assume that the reader is familiar with decision tree learning algorithms such as ID3 (Quinlan, 1986). In order to minimize differences between algorithms in our experiments, the decision tree learner uses the exact same example representation as the naive Bayesian classifier. Although there are better ways of dealing with numeric attribute values in decision trees (Dougherty et al., 1995) we concentrate solely on the issue of constructive in-

Table 21.4 The effect of constructive induction of Cartesian product attributes on ID3.

<i>Domain</i>	<i>Tree</i>	<i>BSEJ</i>
xor	.926	1.00 +
4-parity	.648	.971 +
glass	.786	.788
krkp	.961	.959
mushroom	.996	.996
voting	.930	.934
credit	.798	.806
horse-colic	.783	.780
iris	.912	.931 +
pima-diabetes	.709	.702
wine	.934	.926
wisc-cancer	.952	.951

duction of Cartesian product attributes. Similarly, we will not use any of the more advanced decision tree options such as post-pruning or value subsetting. Therefore, the differences between algorithms that we report are the effects of the representation and search bias.

Table 21.4 shows the results of using constructive induction of Cartesian product attributes in the context of decision trees. These experiments are run in the same manner as the earlier experiment. A decision tree learner with constructive induction is significantly more accurate on only one naturally occurring problem (iris) and the two artificial problems.

A closer investigation of the significant difference on the iris data set revealed that the deletion of an attribute and not the creation of a Cartesian product attribute is responsible for the improvement. Section 21.4 demonstrates that this is not the case with the naive Bayesian classifier.

Although Cartesian product attributes substantially improve the accuracy of the decision tree on exclusive-or and parity they do not help on any real problems encountered so far. Parity and exclusive-or are extreme cases of attribute interaction where no single attribute has information while combinations of certain attributes are very informative. Note that unlike the naive Bayesian classifier Cartesian product attributes do not change the expressiveness of the decision tree classifier. A test on a Cartesian product attribute in a decision tree is equivalent to a test on one of the attributes followed by a test on the other attribute on all branches. Unless individual attributes have no information gain, it would be capable of learning such a sequence of tests as well as having the flexibility to choose a different test under some branches. Furthermore, the change in the search biase caused by Cartesian product attributes could be achieved by allowing the decision tree to exploit look-ahead when building tests. Like Cartesian product attributes, look-ahead in decision trees has been shown to be beneficial on artificial problems, but it is not usu-

ally helpful and sometimes harmful on naturally occurring problems(Murthy and Salzberg, 1995). Cartesian product attributes require that continuous attributes be discretized, losing the ordering information that can be exploited by the decision tree learner. Therefore, we would not recommend using Cartesian product attributes with decision tree learners.

One final observation is worth noting on the results of the previous two experiments. On the six problems where constructive induction improved the accuracy of the naive Bayesian classifier, the naive Bayesian classifier is also significantly less accurate than a decision tree learner. Since the naive Bayesian classifier is the Bayes optimal classifier when attributes are conditionally independent given the class, this suggests that the reason the naive Bayesian classifier is not as accurate as the decision tree learner on these problems is the violation of the independence assumption.

21.3.3 PEBLS

PEBLS(Cost and Salzberg) is a nearest neighbor algorithm that makes use of a modification of the value difference metric, MVDM,(Stanfill and Waltz, 1986) for computing the distance between two examples. This distance between two examples is the sum of the value differences of all attributes of the examples. The value difference between two values V_{k_x} and V_{k_y} of attribute A_k is given by Equation 21.4:

$$\delta(V_{k_x}, V_{k_y}) = \sum_i |\hat{P}(C_i | A_k = V_{k_x}) - \hat{P}(C_i | A_k = V_{k_y})| \quad (21.4)$$

PEBLS assigns a test example to the class of the training example that has the minimum distance to the test example as measured by the value difference metric. PEBLS makes an independence assumption and its accuracy degrades on problems in which this assumption is violated(Rachlin et al., 1994). In Table 21.5, we report on experiments run in the same manner as the previous experiments.

Equation 21.4 is designed to be an estimate of the relevance of an attribute toward making a classification. Difference in values of relevant attributes greatly affect the distance measured by the similarity metric, while difference in values of irrelevant attributes do not. However, without constructive induction of Cartesian product attributes, it considers the relevance of an attribute in isolation. A constructed Cartesian product attribute allows the relevance of a combination of attributes to be determined.

The nature of the independence assumption in PEBLS and the naive Bayesian classifier is quite different. In the Bayesian classifier, joint probabilities are computed as the product of individual probabilities. In PEBLS, the overall distance is the sum of the individual attribute distances which are a function of the conditional probability of the class given the attribute. Nonetheless, Cartesian product attributes are beneficial since they change how the distance is computed. For example, on the exclusive-or problem discussed

Table 21.5 The effect of constructive induction of Cartesian product attributes on PEBLS.

<i>Domain</i>	<i>PEBLS</i>	<i>BSEJ</i>
xor	.740	.999 +
4-parity	.475	.989 +
glass	.821	.816
krkp	.941	.946
mushroom	.999	.999
voting	.882	.946 +
credit	.805	.803
horse-colic	.451	.710 +
iris	.882	.887
pima-diabetes	.479	.676 +
wine	.960	.956
wisc-cancer	.957	.960

earlier, as the number of examples increases, PEBLS without BSEJ will assign a distance of 0 between all examples, since for all attributes and values the probability of a class given an attribute value will be 0.5(Rachlin et al., 1994). However, if a Cartesian product attribute is formed from the two relevant attributes, a test example will be considered most similar to those training examples that share the two relevant attributes. For example, consider classifying an example in which the relevant attributes A_1 and A_2 are 1. In this case, $|\hat{P}(C_i = \text{True}|A_1 = 1 \& A_2 = 1) - \hat{P}(C_i = \text{True}|A_1 = 1 \& A_2 = 0)|$ would be approximately 0. In contrast, for other pairs of values of A_1 and A_2 such as 1 and 0 $|\hat{P}(C_i = \text{True}|A_1 = 1 \& A_2 = 1) - \hat{P}(C_i = \text{True}|A_1 = 1 \& A_2 = 0)|$ would be approximately 1. Therefore, PEBLS will correctly classify exclusive-or examples if the two relevant attributes are joined. Inspection of traces running the algorithm and the near perfect accuracy show that the two relevant attributes are joined by BSEJ.

The accuracy improvement in PEBLS is not restricted to just artificial problems. On three of the naturally occurring databases, there is a significant and substantial increase in accuracy. For example, on the diabetes diagnosis problem, the accuracy increased from .479 to .676 by using the BSEJ algorithm.

21.4 ATTRIBUTE DELETION

(Langley and Sage, 1994) have used a Forward Sequential Selection (FSS) method for eliminating attributes from Bayesian classifiers. They advocate the use of attribute selection to deal with dependencies in Bayesian classifier. Another alternative to eliminating attributes is Backwards Sequential Elimination (BSE)(Kittler, 1986). In the next experiment, we compare BSEJ to FSS and BSE to see if there is an advantage in using constructive induction to create attributes. In addition, we investigate a “forward” approach to constructive induction that we call Forward Sequential Selection and Joining (FSSJ). FSSJ

initializes the set of attributes to be used by the classifier to the empty set. Next, two operators are used to generate new example representations until no improvement is found:

- a. Consider adding each attribute not used by the current example representation.
- b. Consider joining each attribute not used by the current example representation with each attribute currently used.

Table 21.6 shows the results of these algorithms run in the same manner as the previous experiments. For brevity, we present results only on the Bayesian classifier, but similar results were obtained with PEBLS. Table 21.6 indicates when one of the methods for improving upon the naive Bayesian classifier is significantly better than another at the .05 level by placing a “+” after the accuracy. None of the algorithms do particularly well on the parity problem, since they are limited by hill-climbing search. The BSEJ method is more accurate than BSE on 6 problems and there are no significant differences on the remainder. Note that the problems on which BSEJ is more accurate than BSE are the same problems in which a decision tree learner is more accurate than the naive Bayesian classifier, giving additional support for the hypotheses that the constructive induction operator is needed to achieve the increased accuracies. The comparison between FSS and BSEJ is similar. BSEJ is usually more accurate, or at least as accurate, except FSS is significantly more accurate on the congressional voting problem. In this problem there is a single very predictive attribute and forward methods have an advantage over backward when there are many attributes that must be eliminated. Although Langley and Sage advocate using the FSS method for dealing with attribute dependencies, the experimental results indicate that joining a pair of correlated attributes is more useful than ignoring one of the attributes. Like BSEJ, FSSJ has the capability of constructing Cartesian product attributes. However, it is more limited, since to use this operator, one of the attributes must be beneficial individually. Since this is not true on exclusive-or, FSSJ does not perform as well on this problem nor three naturally occurring problems.

21.5 RELATED WORK

(Kononenko, 1991) implemented a method for identifying that attributes are conditionally independent based upon a statistical test that determines the probability that two attributes are not independent. The algorithm joins two attributes if there is a greater than 0.5 probability that the attributes are not independent. Experimental results with this method were disappointing. On two domains, the modified Bayesian classifier had the same accuracy as the naive Bayesian classifier, and on the other two domains tested, the modified Bayesian classifier was one percent more accurate. It is not clear whether this difference is statistically significant.

Table 21.6 A comparison of the subset selection methods (without joining) and the constructive induction selection methods with backward and forward search.

<i>Domain</i>	<i>BSEJ</i>	<i>BSE</i>	<i>FSS</i>	<i>FSSJ</i>
xor	1.00	.450 –	.481 –	.789 –
4-Parity	.475	.442 –	.471	.471
glass	.768	.706 –	.737 –	.765
krkp	.931	.862 –	.739 –	.793 –
mushroom	.992	.952 –	.984 –	.984 –
voting	.920	.902 –	.956 +	.949 +
credit	.833	.847	.830	.836
horse-colic	.802	.806	.811	.809
iris	.938	.939	.941	.942
pima-diabetes	.749	.756	.741	.740
wine	.975	.980	.957 –	.954 –
wisc-cancer	.971	.972	.956	.959

A “+” or “–” indicates that the method is significantly more or less accurate than BSEJ.

(Langley, 1993) proposes the use of “recursive Bayesian classifiers” to address the non-linearly separable problem of Bayesian classifiers. Although the algorithm worked on an artificial problem, it did not provide a significant benefit on any naturally occurring databases including ones on which BSEJ had considerable increases in accuracy.

Our work differs from most in constructive induction in two related ways. First, the constructed attributes are not Boolean or arithmetic combinations of existing attributes and the learning algorithms used are not decision trees or rule learners (Matheus and Rendell, 1989; Ragavan and Rendell, 1993; Sanger, 1992). Cartesian product attributes were shown to have little value on decision trees, but they were important in improving the accuracy on naive Bayesian classifiers and PEBLS.

21.6 CONCLUSIONS

We have investigated the constructive induction of Cartesian product attributes. We proposed an algorithm based on the “wrapper model” for deciding which attributes to join. We showed that this approach was superior to forming Cartesian product attributes guided by an entropy-based measure of conditional independence. This occurs because BSEJ is sensitive to representational biases in addition to violations of independence assumptions. We showed that a backward search method is more appropriate than a forward for creating Cartesian product attributes, although like all hill-climbing algorithms it can have a problem with local minima. We used three artificial concepts to illustrate the types of problems for which BSEJ is beneficial and showed that

BSEJ can result in substantial increases in accuracy on a variety of naturally occurring problems for two different learning algorithms.

Acknowledgments

The research reported here was supported in part by NSF grant IRI-9310413 and ARPA grant F49620-92-J-0430 monitored by AFOSR. I'd like to thank Pedro Domingos, Dennis Kibler, Ron Kohavi, Pat Langley and Kamal Ali for advice on Bayesian classifiers, Cullen Schaffer for advice on cross-validation, Steven Salzberg and David Aha for advice on instance-based learners and Catherine Blake, David Schulenburg and Clifford Brunk for comments on an earlier draft of this chapter.

Notes

1. Note that if the value of either attribute is unknown, then the value of the joined attribute is unknown. Joining is only defined on discrete attributes. If an attribute has continuous values, it must be discretized to be joined. In this chapter, all continuous attributes are discretized into five values.
2. On decision trees, which do not make the same independence assumption, retaining the original attributes had no significant effect in our experiments.
3. The number of possible changes to representation grows faster than exponential in the number of attributes. One way to view this problem is to find a partitioning of the attributes, such that each attribute appears in at most one block of the partition. For example, one partitioning of the attributes {height, age, shoe-size, weight} would be {{height, weight}, {age}}. When two or more attributes appear in the same block, the cross product of those attributes is used in the example representation. There is not a closed form for the number of partitionings but for a fixed number of blocks, this is a Stirling number of the second kind. In contrast, the hill-climbing algorithm considers at most $O(n^3)$ Cartesian product attributes where n is the number of attributes.
4. While BSEJ also has the capability of deleting irrelevant attributes, this never occurs on the exclusive-or problem since it is possible to achieve 100% accuracy on the training set without deleting attributes.

References

- Aha, D. & Bankert, R. (1995). A Comparative evaluation of sequential feature selection algorithms. *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*. Ft. Lauderdale, Fl.
- Almuallim, H., and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Ninth National Conference on Artificial Intelligence*, 547-552. *MIT Press*.
- Caruana, R., & Freitag, D. (1994). Greedy attribute selection. In Cohen, W., and Hirsh, H., eds., *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann.
- Cost, S. & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features *Machine Learning*, 10, 57-78.
- Danyluk, A. & Provost, F. (1993). Small disjuncts in action: Learning to diagnose errors in the telephone network local loop. *Machine Learning Conference*, pp 81-88.

- Dougherty, J., Kohavi, J., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Machine Learning Conference*, pp 194–202.
- Duda, R. & Hart, P. (1973). *Pattern classification and scene analysis*. New York: John Wiley & Sons.
- John, G. Kohavi, R., & Pfleger, K. (1994). Irrelevant features and the subset selection problem. *Proceedings of the Eleventh International Conference on Machine Learning*. New Brunswick, NJ.
- Kittler, J. (1986). Feature selection and extraction. In Young & Fu, (eds.), *Handbook of pattern recognition and image processing*. New York: Academic Press.
- Kohavi, R. (1995). *Wrappers for performance enhancement and oblivious decision graphs*. Ph.D. dissertation. Stanford University.
- Kononenko, I. (1990). Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition. In B. Wielinga (Eds.), *Current trends in knowledge acquisition*. Amsterdam: IOS Press.
- Kononenko, I. (1991). Semi-naive Bayesian classifier. *Proceedings of the Sixth European Working Session on Learning*. (pp. 206–219). Porto, Portugal: Pittman.
- Langley, P. (1993). Induction of recursive Bayesian classifiers. *Proceedings of the 1993 European Conference on Machine Learning*. (pp. 153–164). Vienna: Springer-Verlag.
- Langley, P. & Sage, S. (1994). Induction of selective Bayesian classifiers. *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*. Seattle, WA.
- Matheus C.J., & Rendell L.A. (1989). Constructive Induction On Decision Trees, in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 645–650.
- Murphy, P. M. , & Aha, D. W. (1995). UCI Repository of machine learning databases. Irvine: University of California, Department of Information & Computer Science.[Machine-readable data repository <ftp://ics.uci.edu:/pub/machine-learning-databases/>]
- Murthy, S. & Salzberg, S. (1995). Lookahead and Pathology in Decision Tree Induction *Proceedings of the International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1025–1031.
- Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1, 81-106.
- Rachlin, Kasif, Salzberg & Aha, (1994). Towards a better understanding of memory-based reasoning systems. *Proceedings of the Eleventh International Conference on Machine Learning*. New Brunswick, NJ.
- Ragavan, H. & Rendell, L. (1993). Lookahead feature construction for learning hard concepts. *Machine Learning: Proceedings of the Tenth International Conference*. Morgan Kaufmann.
- Salzberg, S., Chandar, R., Ford, H., Murthy, S. & White, R. (1995). Decision Trees for Automated Identification of Cosmic Ray Hits in Hubble Space Telescope Images. *Publications of the Astronomical Society of the Pacific*.

- Sanger T.D., Sutton R.S., & Matheus C.J.(1992). Iterative Construction of Sparse Polynomial Approximations, in Moody J.E., et al.(eds.), *Neural Information Processing Systems 4*, Morgan Kaufmann, San Mateo, CA, pp.1064–1071.
- Stanfill, C. & Waltz, D. (1986). Towards memory-based reasoning. *Communications of the ACM*, 29, 1213-1228.
- Wan, S. & Wong S. (1989). A measure for concept dissimilarity and its application in Machine Learning. In R. Janicki and W. Koczkodaj (eds.) *Computing and Information* 267–274.

22 TOWARDS AUTOMATIC FRACTAL FEATURE EXTRACTION FOR IMAGE RECOGNITION

Matteo Baldoni, Cristina Baroglio,
Davide Cavagnino and Lorenza Saitta

Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
`{baldoni|baroglio|davide|saitta}@di.unito.it`

Abstract: Fractal encoding based on *Iterated Function Systems* is a well-known method which is currently exploited mainly for image compression tasks. The reason is that it allows to produce very compact encodings without any meaningful loss of precision. In this work, we present the results of a study aimed at showing that, within the field of image analysis, it is possible to use fractal encodings based on Iterated Function Systems also to extract information that is relevant in object recognition tasks. In particular, we propose a new kind of features, called *fractal features*, to be used jointly with machine learning techniques in isolated 2D image recognition tasks and show that it is possible to extract them in an automatic way.

22.1 INTRODUCTION

One of the central problems in many artificial vision tasks is the classification of the visual representations of objects with respect to a given set of classes. The possible applications of automatic recognition systems are many and appealing (see (O'Gorman, 1994)); for instance, CAD graphical interfaces could be enabled to automatically interpret sketches that are drawn by human operators. Similar systems could also be used both for interpreting graphical queries to image databases and for finding the items that satisfy them.

In human beings image recognition requires the cooperation of cognitive processes at different levels of abstraction. In the case of artificial systems, instead, one of the main problems to solve is to find *proper representations* of the

data. Supposing that a good representation method is available, it is possible to build a set of object prototypes that are used during classification. When an image is to be classified, the system decides which of them is the most similar to it and returns the corresponding class as an answer. Since the classification is done according to the knowledge encoded by the prototypes, one necessary condition for building an image classifier is to be able to build a *model* for each target class (Besl and Jain, 1985; Chin and Dyer, 1986; Grimson et al., 1990). Model generation can be done either by hand (Nakamura et al., 1991; Pentland et al., 1994) or by means of a *learning* process, in which an adaptive classifier is *trained* to distinguish instances of different classes (Rumelhart and McClelland, 1986). Usually, in the former case model generation requires quite a big effort while in the latter it is sufficient to have a set of examples (i.e., a set of images) to use for training.

Generally speaking, the approach to image recognition can be summarized by the following steps: finding a way that allows one to *extract*, in an automatic way, a small set of *features* characterizing the images of the objects that belong to the target classes; building a system that maps the image encodings to the target classes; using the obtained classifier for recognition purposes. Of these three steps, the first is the most critical. The “ideal” features we would like to use are easy to extract in an automatic way, robust, and well suited to a fast and reliable interpretation. Furthermore, since we deal with image recognition, they should also be invariant with respect to the size of the objects. Last but not least, they should be as few as possible. The reason is that, beside the fact that more compact representations have the advantage of making the classification problem more tractable in terms of computational effort, in the particular case in which adaptive classifiers are used, compact representations have also the advantage of widening the range of the machine learning methods that can be applied for building the classifiers (Michie et al., 1994).

With this work we propose a new type of feature which shows the above mentioned good properties. These features are derived from a fractal description of the images obtained by means of *Iterated Function Systems* (IFS). Image encoding by means of IFSs, first proposed by Barnsley in the field of image compression (Barnsley and Sloan, 1988), allows to obtain extremely concise encodings of highly complex images, such as landscapes, clouds, and trees (Pentland, 1984). Nevertheless, IFSs were never used in image *classification* tasks. The novelty of our work consists, then, in showing that there exist IFS-based features, which retain the characteristic information of isolated shapes, that can be automatically extracted and, then, used to train adaptive classifiers to distinguish images of different kinds of objects. In particular, in this chapter we also describe two simple systems (one evolution of the other) that allow discriminant fractal feature extraction.

In order to check the goodness of the fractal features, we initially worked on a real-world well-known task: learning to recognize hand-written digits. The data consist of a collection of images, which was made available by the German Federal Post (Michie et al., 1994). More recently, we have also used fractal

features for learning to classify different kinds of trees. In this case, since the shapes have a very complex structure, the use of traditional feature extraction methods does not allow to obtain compact representations.

Due to the fact that the discriminant features extracted by means of IFSs have a numerical nature, we implemented the classifier as a *neural network* (NN).

22.2 FRACTALS AND IFS

The basic idea underlying fractal generation is that a fractal contains, at any scale, smaller copies of itself, i.e., it is *self-similar*. One way of generating fractal images in the plane is to *iterate* simple geometrical transformations, starting from a predefined shape (Falconer, 1990). For instance, from a geometrical

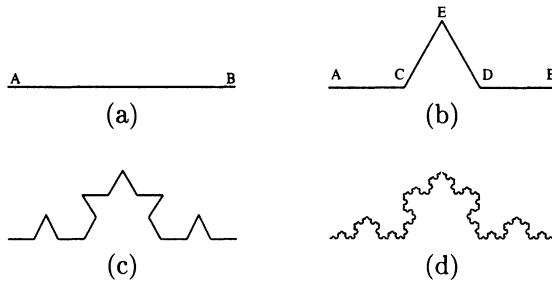


Figure 22.1 Successive stages in the construction of the von Koch curve. At each step, the middle third of each existing segment is removed and replaced by the two sides of an equilateral triangle. (a) Initial segment. (b) Basic transformation. (c) Curve after 2 steps. (d) The von Koch curve.

point of view, the rule that allows the curve of Figure 22.1(d) (the von Koch curve) to be built consists in *mapping* the initial segment AB, Figure 22.1(a), to each of the segments AC, CE, ED and DB respectively, Figure 22.1(b), then, repeating the same process for each of the produced segments, Figure 22.1(c). Each mapping is a *contraction*, i.e., it maps a set of points in another set of points which are closer to each other (in the example, for instance, the length of the segment reduces to one third of the original).

A particular kind of mapping is the *affine contractive transformation*, which, in words, allows a smaller copy of an initial pattern to be produced, keeping its original shape. More formally, let us consider the n -dimensional space \mathbf{R}^n and let \vec{z} be a point in \mathbf{R}^n : a mapping $M : \mathbf{R}^n \rightarrow \mathbf{R}^n$ is an *affine contractive transformation* if it is a combination of *rotations*, *scalings* and *translations* of the coordinates. An affine contractive transformation has the general form $M(\vec{z}) = T(\vec{z}) + \vec{w}$, where T is a linear transformation on \mathbf{R}^n and \vec{w} is a vector in \mathbf{R}^n (Falconer, 1990).

In the case of a *two-dimensional space* (i.e. in the case of 2D images) an affine contractive transformation has the matrix form:

$$M\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} r \cos\theta & -s \sin\phi \\ r \sin\theta & s \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \quad (22.1)$$

where θ and ϕ are rotations and r and s are scalings on the x and y coordinates, respectively; an example is shown in Figure 22.2. An affine contractive transformation is univocally defined by means of its six parameters: a , b , c , and d , which define a *geometrical transformation*, and e and f which define a *translation* in the plane.

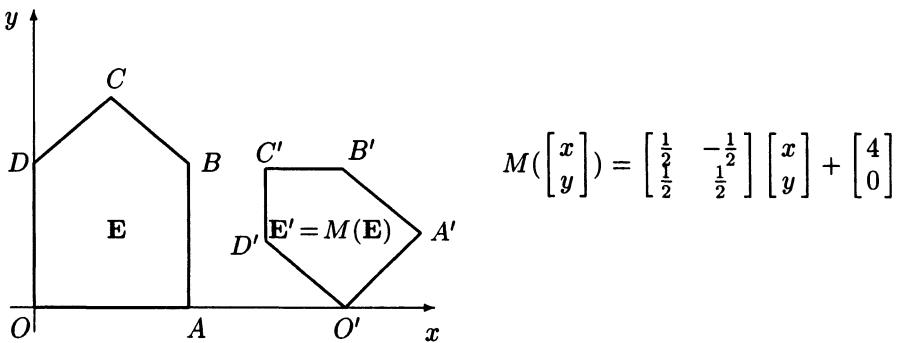


Figure 22.2 Example of affine contractive transformation: figure **E** is mapped on **E'** by mapping M .

A set of contractions is known as *Iterated Function System* (IFS for short). Two important properties of IFSs are the following. The first is that for every IFS in \mathbf{R}^n there exists a set F such that $F = \bigcup_{i=1}^m M_i(F)$ (F is a fixed-point of the IFS). F is called *invariant set*; it is *unique* and *non-empty* and is a *fractal* when its dimension is not integer. For example, the von Koch curve in Figure 22.1(d) is the invariant set for the mappings described in Figure 22.1(b); its dimension is $\log 4 / \log 3$.

The second property is that, given any subset E of \mathbf{R}^n and an arbitrary precision of approximation, it is always possible to find an IFS whose invariant set approximates E as closely as desired. This second property is known as the *Collage Theorem* (Barnsley and Hurd, 1993). In the plane the Collage Theorem provides a method for encoding an image, ensuring the accuracy of the subsequent reconstructions: the image is encoded by covering it with a set of contracted affine copies of itself (see Figure 22.3), the closer the union of the contractions to the object, the more accurate its reconstruction.

In the literature, there are many methods that exploit the Collage Theorem to compress images by means of IFS¹ (Barnsley and Hurd, 1993; Jacquin, 1990) (one of them, derived from the work by Fisher (Fisher, 1994), was used in some

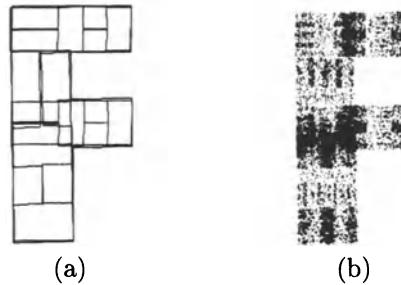


Figure 22.3 (a) It is possible to cover a capital “F” with copies of itself in many ways. One possibility is shown here: (from (Oliver, 1992)): the “F” is covered with six contracted copies of itself, each “bar” being covered with two contracted copies placed side by side. (b) The corresponding invariant set.

of our experiments). We, instead, exploit the Collage Theorem for extracting the characteristic information of an image for classification purposes.

Table 22.1 IFS code of the capital “F” in Figure 22.3.

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
-0.03	-0.37	-0.52	0.02	0.17	-0.50
-0.03	-0.37	-0.52	0.02	0.39	4.38
0.50	-0.06	0.00	-0.80	1.23	-2.48
-0.50	-0.02	0.01	-0.80	1.69	-2.56
0.01	-0.37	0.52	-0.02	0.15	-0.99
0.01	-0.37	0.52	-0.02	0.38	3.93

22.3 EXTRACTION OF FRACTAL FEATURES

The task we dealt with in our work is *image classification* and the method we applied for finding IFS-based image encodings is the *Collage Theorem*. The use of this theorem is not new and was already investigated in the field of image compression but, as it turned out in our work, the particular techniques developed in this field to make the Collage Theorem operational do not produce representations that allow to discriminate among different classes of objects. The reason is that, given an image, there is an infinite number of IFSs that encode it in many different ways; depending on the task in which the image is to be used, different IFS encodings may produce different results. On the other hand, given a specific task, the optimal IFS encoding is generally unknown and can be only approximated. Therefore, given a task, the problem is to find an algorithm that allows to produce proper encodings in an automatic

way. Supposing that an IFS encoding procedure is available, let's consider an image ξ in 256 grey levels. Let $M(\xi) = \{M_i | 1 \leq i \leq r\}$ be an IFS which covers the image according to this procedure. Since an IFS is completely defined by the values of the parameters of its transformations, we can use the vector $\vec{x} = \langle x_1, x_2, \dots, x_n \rangle$, containing the values of the parameters of the r transformations M_i , as a representation of ξ . In the following, we will call the x_i 's *fractal features*. The number of fractal features n that characterize an object is equal to the number of parameters defining each transformation times the number r of transformations applied (see Table 22.1). Anyhow, n is generally *much smaller* than the number of pixels of the original image (for instance, our trees, whose images are made of 240×320 pixels, have meaningful encodings that consist of just 12 numbers) and, maybe even more important, the theoretical results about IFSs ensure that n does not depend on the image size.

As mentioned in the introduction, the most common way in which descriptive features are used in image classification tasks is to define a set of prototypes, based on their values. When an image is to be classified, its encoding is compared to the prototypes so to find the most similar to it: the class of that prototype will be the class of the image. We also said that, since finding a set of prototypes is not trivial, it is preferable to use *adaptive classifiers* because they can extract a body of classification knowledge from a set of instances of the various classes. However, adaptive classifiers are applicable only in the case in which the features that describe the learning instances are *robust* and *stable*, i.e., varying in a continuous way the shape represented in an image, the features that describe it vary in a continuous way as well. Then, images having both close shapes and close topological characteristics should have similar encodings. IFSs show these properties (Falconer, 1990), as a consequence, fractal features are *suitable* for training adaptive classifiers. In Section 22.4 we informally describe two systems that allow to extract discriminant fractal features in an automatic way.

22.3.1 Related Work

The above described approach to image representation is quite different than the classical one (see (R. Kasturi, 1995; Gonzales and Woods, 1992) for a survey), which consists in making the pixel matrix undergo a sequence of successive processing steps, aimed at representing the information contained in it by means of more and more abstract descriptors. On the contrary, image representation through fractal features consists of a single step. For the sake of completeness and of a better understanding, in this section we briefly describe some of the best-known kinds of features that are commonly used in image classification tasks.

Leaving apart the lowest levels of processing, which include binarization and segmentation, image analysis is done either by using *line features*, *shape descriptors*, and *critical points*, or by exploiting *geometrical models* of the objects to recognize. *Line features* describe lines, curves, polygons and circles. The meth-

ods used to extract them include thinning, chain coding and polygonalization, which are usually applied one after the other. The resulting representation is an unlabeled geometrical sketch of the image. *Critical points* and *shape descriptors* are higher-level geometrical features. Critical point detection allows one to locate the geometrical components of a given image. Then, as in the previous case, it can be used to produce a sketch. Sketches can, then, be compared with a set of templates (or models) defined a priori.

While these techniques are general, in the sense that they can be applied to any kind of image, the choice of *shape descriptors* is application dependent. Some typical features are the shape area, the length of its contour and the Fourier descriptors. An alternative consists in using the *moments* of the shape, which encode characteristics like the shape elongation, non-roundness and eccentricity. These features are particularly interesting also because they can be combined into functions that are *invariant* w.r.t. image scaling and rotation. The disadvantage is that they have some difficulties in representing shapes characterized by concavities. A different approach consists in using *geometrical models*, in which numerical vectors represent relative positions of characteristic parts of the shape. As usual, the problem is that models are to be built and require a lot of *a priori* knowledge about the objects in the image.

Differently than the classical kinds of features, the *fractal features* are simple to extract, i.e. they do not require any preprocessing since they are obtained directly from the pixel matrix. Moreover, they do not require any model of the classes of interest to be supplied; actually, one of the reasons for which they were investigated is that, due to their robustness and stablyness, they allow the use of adaptive classifiers for building the classification knowledge in an automatic way.

22.4 EXPERIMENTS

Initially, all experiments were done on the *ten digits* test-bed. Briefly, this data set is a collection of hand-written instances of the ten digits, acquired with a resolution of 16×16 pixels with 256 grey levels. In some experiments the instances were binarized, i.e., all pixels whose value was higher than a certain threshold were considered as an element of the digit otherwise they were considered as background. Later on, the experiments were repeated on a second test-bed, a collection of images of three different kinds of trees. In this case, 150 images made of 240×320 pixels were available (50 per class). By working on these two very different kinds of images (digits are linear shapes spread across a planar surface while trees are very complex shapes) we meant to show that IFS-based fractal features are suitable to characterize a wide family of images and not only some very specific application.

Two main series of experiments were carried on. The first had the exploratory aim of verifying, in a short time, whether IFSs capture characterizing information about images; to this purpose, the instances were hand-coded and, then, processed by a neural net. The second investigates the possibility of extracting in an automatic way the features we propose. To this aim, we have

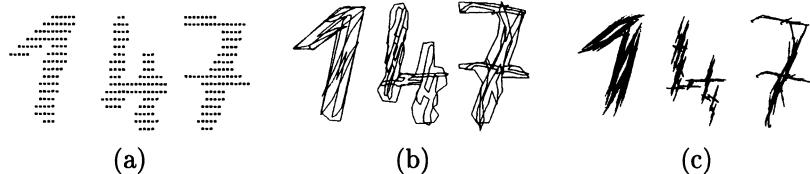


Figure 22.4 (a) Binarized instances of the digits “1”, “4” and “7”. (b) Digit covering with four self-affine transformations. (c) Digit fractal reconstruction.

both used an image compression algorithm, (*enc* by Fisher (Fisher, 1994)), and developed an original simple feature extraction algorithm. This algorithm was implemented in two versions: one uses contractions of fixed simple shapes that are not self-similar copies of the images to classify, while the other produces transformations based on contractions that are affine to the represented image.

The adaptive classifier always consisted in a three-layered feed-forward neural network². Two different learning rules were used: classical back-propagation (Rumelhart and McClelland, 1986) and the Scaled-conjugate Gradient learning rule (SCG for short) (Moller, 1993), which exploits second order derivatives in order to find a better way to a (local) minimum. The recognition rates are very close, but SCGs converge much faster.

When not otherwise specified, in the experiments on digits the learning set contained 300 instances (30 instances per class), while the test set contained 700 instances (70 per class), all different than those processed during training. Cross-validation was used to check the independence of the results from the particular learning and test sets used: each experiment was repeated on three different learning/test set pairs; the results we report are always the average of those obtained in the different runs. In the experiments on trees, instead, the learning and test sets consisted of 75 instances (25 per class) and six-fold cross-validation was applied.

22.4.1 Using a Hand-made Covering

The aim of this experiment (which was done on the digits only) was to show that, given a *proper* method for finding an IFS whose invariant set approximates a given image, the extracted fractal features allow an adaptive classifier to discriminate between different digits. The method we adopted was to cover *by hand* each binarized instance with a set of contractions of it. Only digits “1”, “4”, and “7” were taken into account, because of their similarity in hand-writing. Each digit was encoded by covering it with four self-affine contractions (see Figure 22.4). Then, each digit was represented by 24 values (4 transformations \times 6 parameters). Afterwards, a three-layered Multi Layer Perceptron (MLP) with 24 input units, 80 hidden neurons³ and 3 output neurons was trained to distinguish among the three classes. The learning set was made of 30 examples (10 per class) while the test set consisted of 120 examples (40

per class) different than those used during training. Results show a 100 % recognition rate both on the learning and on the test sets.

The conclusion we can draw is that the fractal features *are* discriminant and are, then, worth to be deeply studied also in the case of image classification. The problem resides in finding proper fractal feature extraction algorithms.

22.4.2 Automatic Fractal Feature Extraction

The previous experiment confirmed that fractal features capture image discriminant information, however, in order to allow their use in practical applications, it is fundamental to find some *automatic* feature extraction method. The first attempt we did consisted in looking whether an algorithm of this kind was already discussed in the literature. The result was that the only image encoding systems, based on IFSs, that had already been developed, were *image compression* systems. Then, we chose one of the best-known, *enc* (Fisher, 1994), which is representative of most of fractal compression systems, and investigated if the encodings it produces are sufficiently informative for classification tasks.

This algorithm exploits the Collage Theorem for *local IFSs* (see (Barnsley and Hurd, 1993)): instead of covering the whole image with contracted affine copies of itself, it covers the image with contracted affine copies of *parts* of itself. Briefly, *enc* partitions a *grey scale* image into a set of non-overlapping square blocks R_i (*range blocks*). Then, it finds a set of larger blocks D_i (*domain blocks*) in the same image and a set of affine contractions M_i , each of which, applied to a domain block D_i , provides a good approximation of the corresponding range block R_i . According to the Collage Theorem, the set of contractions M_i is the *fractal code* for the original image. By iterating it, starting from *any* initial image, this code generates a reconstructed image, which is quite close to the original. *enc* imposes some restrictions on the D_i 's and R_i 's to limit the computational cost in finding these blocks: the D_i 's are squares whose sides are twice as long as those of the R_i 's. Only 8 geometric transformations are allowed, namely the identity, 90° , 180° and 270° anticlockwise rotations, the reflection in the two axis (x and y) and in the two diagonals. In this way, a transformation can be specified by *five numbers*: a number t between 0 and 7 that codes the selected geometrical transformation, two numbers, e and f , for the translation, and two real numbers, s and o , indicating the luminance transformation (these two numbers are necessary to compression because *enc* works on grey scale images).

In these experiments each digit was approximated by 16 transformations (each R_i being a 4×4 square), making 80 features in all (16 IFSs \times 5 parameters). The reason for using 16 transformations is that, on one hand, by using smaller encodings the adaptive classifiers did not show any learning performance while, on the other, the next level of detail causes the number of fractal features to drastically increase because of the quad-tree partitioning (64 transformations \times 5 parameters makes 320 features, which is more than the number of pixels that make the image). The production of many transformations is, actually, a disadvantage of using “normal” fractal encoders, and, as

experiments have shown, the use of too many transformations decreases the recognition rates.

Due to the number of dimensions of the input and output spaces (in these experiments we tried to learn a classification knowledge for all ten classes), three NNs had to be used. The first, having 250 hidden neurons (as well as the third one), was trained to recognize digits from “0” through “3” using as counterexamples also all digits from “4” through “9”. The second had only 200 hidden neurons and was trained to classify digits “4”, “5”, and “6”. The third one learned to classify the remaining digits. The learning set was made of 500 digits (50 per class) while the test set was made of 2500 instances (250 per class). Results are reported in column (d) of Table 22.2. As can be observed, for five classes out of ten we achieved a recognition rate higher than 70%, however, the average performance is quite low (65.2%).

We used *enc* also to produce IFS-based representations of the trees. Each tree was, again, represented by means of 16 transformations (i.e. by 80 features). An average 64% recognition rate was achieved on separate test sets.

The conclusion we can draw from these results is that the fractal encoding method applied by image compression systems like *enc* is too much tailored to the application (i.e. to compression) and is not good for capturing the kind of information that is necessary in classification tasks. Compression systems better capture what we can call a “surface” information, more related to changes of texture than to the depicted shape. This result motivates the search for different fractal encoding systems, more shape-oriented.

Using a Simple Fractal Feature Extraction Algorithm. The next step was developing two original fractal feature extraction systems, based on the Collage Theorem. The systems we developed are based on the observation that, since we are looking for some discriminant features, we do not need to perfectly cover a given image: an approximation of its shape will be sufficient. The first system investigates the use of contractions that are not affine to the image to represent and was applied to the digits test-bed. The second, instead, is characterized by the use of self-affine transformations and was applied to the trees classification problem (one example for each tree class is reported in Figure 22.6).

Note that this is the greatest difference between the two systems. The underlying methods are very similar to each other, being both an operationalization of the Collage Theorem. In the following we will call them respectively *System One* and *System Two*.

System One. System One covers an image by means of a set of contractions of a shape that is *different* (and simpler) than the one of the digit itself. This is possible because the IFSs capture a kind of information that is *intrinsic* to the covered digit and does *not* depend on the particular shape used for covering (see the definition of IFS). Moreover, using predefined simple shapes has also the advantage of simplifying the covering operation.

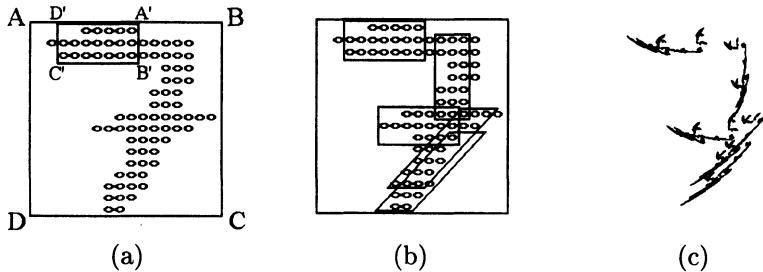


Figure 22.5 (a) Sample digit inside its square box $ABCD$ (the "o"'s represent black pixels). Application of the first affine contraction resulting in $A'B'C'D'$; (b) Final coverage using 5 contractions; (c) Fractal reconstruction: note how the digit structure is captured, even though the reconstruction is far from being accurate.

In particular, System One is based on covering the *binarized* image of a digit by means of a *fixed* number of affine contractions of the *square box* containing it. In order to reduce the complexity of the computation, only a *limited number* of geometric transformations is allowed. With reference to Equation 22.1, all rotations belong to the set $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$, the scaling is fixed a priori and depends on the experiment, while the translations are not limited. Consequently, we were able to use an even *more compact* IFS representation; each transformation is encoded by means of *three* numbers only: the translations (e and f) plus a number t , between 0 and 3, which encodes the rotation.

The *feature extraction* procedure was implemented in *ECLⁱPS^e*, a Constraint Logic Programming (CLP) language, which allows constraints on finite domains to be defined⁴. CLP was used because it allows a fast prototyping (this was the first feature extraction program we implemented) and also because it allows the search space to be cut in a very simple way.

The system works as follows. First of all, a set of geometric transformations of a contracted copy of the square box, which are parametric with respect to e and f (the parameters encoding the translation), are produced. We will call them *proto-transformations*. Then, the feature extraction procedure iteratively tries to cover a yet uncovered part of the digit by means of the proto-transformations. At each iteration a new transformation M_i is added to the IFS that will represent the image. M_i is chosen out of the set of all possible transformations by means of a *scoring procedure* (in this system, the scoring procedure simply consists in counting the number of the newly covered pixels). Then, the feature extraction procedure marks the newly covered pixels and starts a new iteration. The loop ends when a predefined number of transformations has been produced. Figure 22.5 shows the whole process.

The results of this series of experiments are reported in Table 22.2. Columns (a), (b) and (c) contain the average recognition rates obtained on the test sets for each digit. All the NNs used had 500 hidden neurons and applied the SCG learning rule. They were trained and tested on the same data, the only

Table 22.2 Recognition rates of the experiments: (a) 5 transformations with a scaling to 7×3 pixels; (b) 6 transformations with a scaling to 5×3 pixels; (c) 6 transformations with a scaling to 5×3 pixels (no binarized images); (d) 16 transformations obtained by means of enc.

<i>digit</i>	(a)	(b)	(c)	(d)
“0”	89.05 %	84.95 %	79.52 %	79.60 %
“1”	99.05 %	96.67 %	97.14 %	78.00 %
“2”	86.67 %	91.90 %	90.48 %	71.60 %
“3”	86.67 %	84.19 %	82.38 %	60.80 %
“4”	84.95 %	81.43 %	83.33 %	65.60 %
“5”	78.57 %	86.67 %	86.66 %	81.60 %
“6”	80.00 %	90.19 %	83.33 %	72.40 %
“7”	96.67 %	92.38 %	96.19 %	69.20 %
“8”	47.62 %	62.38 %	62.86 %	26.00 %
“9”	79.52 %	79.00 %	80.00 %	47.20 %
<i>avg.</i>	82.86 %	85.00 %	84.19 %	65.20 %

difference consisting in the values of the parameters used by the algorithm described above: the results of column (a) were obtained on digits covered by means of 5 transformations, where the 16×16 pixel frame square box is scaled to a 7×3 pixel rectangle. Column (b) refers to digits covered by means of 6 transformations with a scaling to a 5×3 pixel rectangle. Column (c) refers to 6 transformations with a scaling to a 5×3 pixel rectangle, the difference with the previous case being that now the digit images were not binarized any more, i.e., the 256 grey levels were maintained, and the search for the best matching transformations took into account also the intensity level of the pixels. In the first case each image is, then, encoded by 15 numbers, in the second and in the third by 18 numbers.

If we compare these results to those obtained using *enc*, we note that the average recognition rate was increased of about 20 percentage points; moreover, we achieved a recognition rate greater than 80% for eight classes out of ten. As a consequence, we can state that it is, actually, possible to develop fractal feature extraction systems that capture discriminant information and that IFSs may have a wider range of applications than what thought in the past.

The recognition rates achieved are very close to those obtained by AC2 (84.5%), INDCART (84.5%), C4.5 (85.1%), NewID (85.5%), and CN2 (86.6%) (in (Michie et al., 1994)) with the advantage that the fractal features are invariant with respect to the size of the shape. Of course, we are still far from the 95% recognition rate of *k-n-n* (Michie et al., 1994) but our aim was to verify that this kind of system can be built, opening the way to deeper investigations.



Figure 22.6 Three instances of, respectively, class elm, oak and lime-tree.

System Two. System Two basically works in the same way as System One (it is an extension of it) but it produces contractions that are affine to the shape to be encoded (following more closely what stated by the Collage Theorem) and can deal with grey-level images. It has also a different scoring criterion, which takes into account the pixels intensity levels. This criterion selects, as a first choice, those transformations that cover more uncovered pixels; only as a second choice, it allows the selection of transformations that cover also pixels that are already covered by previously selected transformations. In no case it selects transformations that cover a part of the background bigger than a predefined little threshold. This last possibility was left because of the difficulty of precisely cutting a grey-level shape out of a scene. System Two was written in C.

We did various experiments, using 3, 4 or 5 transformations to encode each tree. Since 3 parameters (one encoding the proto-transformation plus two parameters, e and f for the translations – see Section 22.2 –) are sufficient to encode each transformation, each instance was represented by 9, 12, or 15 numbers. Another constraint was that, for the sake of computational simplicity, the only rotations allowed were those in the set $\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$. The best results were obtained using 4 transformations; in this case, the proto-transformations were scaled copies of the original image with a side scaling factor equal to 0.35. The average recognition rates achieved for the three classes were, respectively 88.61% (class elm), 91.38% (class oak), and 75.38% (class lime-tree). The average recognition rate on the three classes was, then, equal to 85.12%.

The main observation we can draw from these experiments is that the performance achieved is very good, especially taking into account the number of fractal features extracted to encode each instance (just 12 numbers for representing an image made of 240×320 pixels) and that, due to the very limited number of rotations allowed, the most informative datum, i.e. the inclination of the branches, was not supplied to the adaptive classifier. The second observation, that makes these results even more interesting, is that the features were extracted in an automatic way, obtaining again an average recognition rate that is about 20 percentage points greater than the one achieved using a compression system. These results confirm those obtained in the experiments on digits done with System One.

We are strongly convinced that much higher recognition rates will be achieved as soon as some better heuristic for cutting the search space will be found. In fact, if such a heuristic were available, a richer set of proto-transformations could be used.

22.5 DISCUSSION AND FUTURE WORK

Even though IFSs have been extensively studied and used for image compression, they were never investigated as a means for characterizing visual representations of objects in image classification tasks. In the literature, in fact, there are very few examples of works where fractals have been used for classification purposes. The only ones that we could find are (Freisleben et al., 1993), where a neural network is trained to distinguish fractal images from non-fractal images, (Erkmen and Stephanou, 1990), where fractals are used to build the belief functions of a Bayesian classifier, and (Imiya et al., 1996), where a metric for planar self-similar forms is proposed.

In this last work, which is probably the closest to the topics we explored, a metric is defined and used for measuring the distance between IFSs: the smaller the distance the more similar the represented shapes. Note that image classification is a natural application of similar metrics, however, in order to use this metric in recognition tasks some models are to be given for comparison. Furthermore, it is necessary to find a technique for producing IFS encoding for the learn/test instances. Only then, it is possible to apply a metric to compute the similarity between instances or between instances and prototypes.

The greatest novelty of our work is that we focused on finding the IFS representations of a set of given images and, then, we used these representations to learn a classification knowledge. Then, by attacking both image feature extraction and classification knowledge acquisition, our method is to some extent *complementary* to (Imiya et al., 1996). Moreover, it is simpler than the classical approach to image feature extraction because it allows features to be extracted in one single processing step, producing extremely concise encodings.

With our work we have proved that fractal features capture discriminant information of images of different objects and can then be used also in classification tasks. In fact, as experiments show, when a perfect encoding is produced the recognition rates achieved are optimal (100%). The problem is that specific algorithms for fractal feature extraction are to be developed. In fact, even though the theory ensures that it is always possible to approximate and, then, encode any image by means of some IFS, for each image there are infinite alternative IFS-based encodings. Then, the problem is to find an algorithm that allows to select the IFS that better characterizes the given image. Unfortunately, the only fractal encoders that can currently be found were developed specifically for image compression. So, for instance, even though *enc* allowed a recognition rate greater than 70% to be achieved for five digits out of ten, the overall performance is quite poor. In order to show that such algorithms can be developed, we have designed and implemented two of them. Even though they are very simple, experiments showed a remarkable increase of the recognition

rates, achieving an average 85%. Despite to the limitations imposed on the two systems, we believe that the results obtained motivate a deeper investigation of the heuristics that guide IFS selection for producing characteristic features to be exploited in classification.

The digits application was chosen because it is a test-bed that is well-known in the machine learning literature; in particular, it was used in the StatLog Project, which was aimed at the comparison of many learning algorithms (Michie et al., 1994). One interesting observation is that, as explained in (Michie et al., 1994), most of the learning algorithms cannot deal with applications having hundreds of dimensions, while those that can take a too long time before converging. Then, another reason that motivates further studies about the topic we proposed is that the use of fractal features reduces both the number of space dimensions and the set of possible values for each input feature. The consequence is that the input space becomes very small and the learning time diminishes.

The trees application, on the other hand, proves that concise fractal encodings can be produced so to capture the discriminant aspects of different kinds of very complex (and also very similar) objects. It also demonstrates that this kind of features can practically be used to characterize a wide variety of images and not only some specific application.

After the described experiments, we have also started a research aimed at comparing the performances obtained by using learning methods that are different than SCGs. We have alternatively applied regression trees (CART) (Breiman et al., 1984), genetic algorithms (Goldberg, 1989) (REGAL)⁵, and Cascade-correlation (SNNS) to the data encoded by the algorithm we developed. Even though the results collected are only preliminary, as expected, *numerical* methods (i.e. neural networks) gave the best results. Depending on the parameters of the encoding algorithm, SCG networks achieved a recognition rate in between 82% and 85%. Cascade-correlation networks (as an example of a different network model) achieved 79%. Symbolic methods, instead, achieved a maximum 72%. We believe that the reason for which neural nets (and, in particular, full-connected neural nets) perform better is that they can capture the structure of a instance, leaving out the particular order of the transformations in it. This is particularly true in the case of digits. Due to the noise typical of hand-writing and to the fact that the encoding algorithm selects first those transformations that cover more pixels on the instance, the following phenomenon can arise. Consider two instances of a same digit, say two “4”, and suppose, for instance, that in the first “4” the vertical bar is strongly marked while in the second, the oblique line is strongly marked. Then, it is very likely that the first transformations generated to encode the first “4” cover the vertical line whereas those created to cover the second “4” cover the oblique line. This means that the two instances will be encoded by two sequences T_{11}, \dots, T_{1n} and T_{21}, \dots, T_{2n} where, for example, T_{11} covers the vertical bar in the first instance while in the second example, T_{21} covers the oblique line and T_{23} covers the vertical one. Note that the information contained in the two sequences

will be the same, the only difference is the *order* of the transformations. Now, our algorithm imposes a simple ordering of the transformations which is based on the position of the baricentre of the contractions; however, due to the non-regularity of hand-writing, the above mentioned problem can still arise (it is easy to imagine examples of instances of a same digit were such an ordering rule produces different sequentializations). Unfortunately, the symbolic methods we used are not able to generalize over information placed at different points in the sequences, e.g. over T_{11} and T_{23} in the example. To overcome this limit, we mean to try *First Order* learning tools. Two alternatives are either to search for more sophisticate ordering rules or to produce richer learning sets that take into account any possible ordering of the transformations for any given instance.

Acknowledgments

The authors thank Prof. Charles Taylor for all the information about the StatLog project and Giuseppe Lo Bello for helpful discussions.

Notes

1. More precisely, compression systems exploit the Collage Theorem for Local IFSs.
2. All the neural networks used were implemented by means of SNNS, the Stuttgart Neural Network Simulator, developed at the Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, Germany.
3. Actually, a wide experimentation was carried on to decide the number of hidden neurons to use. The numbers reported here for each experiment are those of the networks that showed the best performances.
4. ECLⁱPS^e was developed at the European Computer-Industry Research Centre (ECRC), Germany.
5. Systems CART and REGAL were implemented at the Dipartimento di Informatica, University of Turin, Italy.

References

- Barnsley, M. and Hurd, L. P. (1993). *Fractal Image Compression*. AK Peters, Ltd., Wellesley, Massachusetts.
- Barnsley, M. and Sloan, A. (1988). A Better Way to Compress Images. *BYTE Magazine*, 13(1):215–223.
- Besl, P. and Jain, R. (1985). Three Dimensional Object Recognition. *ACM Computing Surveys*, 17:75–154.
- Breiman, L., Friedman, J., Ohlsen, J., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA.
- Chin, R. and Dyer, C. (1986). Model-Based Recognition in Robot Vision. *ACM Computing Surveys*, 18:67–108.
- Erkmen, A. M. and Stephanou, H. E. (1990). Information Fractals for Evidential Pattern Classification. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(5):1103–1114.
- Falconer, K. (1990). *Fractal Geometry, Mathematical Foundations and Applications*. John Wiley & Sons Ltd., Chichester, UK.

- Fisher, Y. (1994). *Fractal Compression: Theory and Application to Digital Images*. Springer Verlag, New York.
- Freisleben, B., Greve, J. H., and Löber, J. (1993). Recognition of Fractal Images using a Neural Network. In Mira, J., Cabestany, J., and Prieto, A., editors, *New Trends in Neural Computation, IWANN '93*, volume 686 of *LNCS*, pages 632–637. Springer-Verlag.
- Goldberg, D. E. (1989). *Genetic Algorithms*. Addison-Wesley, Readings, MA.
- Gonzales, R. C. and Woods, R. E. (1992). *Digital Image Processing*. Addison - Wesley Publishing Company.
- Grimson, W., Lozano-Pérez, T., and Huttenlocher, D. (1990). *Recognition by Computer: The Role of Geometric Constraints*. The MIT Press, Cambridge, MA.
- Imiya, A., Fujiwar, Y., and Kawashima, T. (1996). A Metric of Planar Self-Similar Forms. In Perner, P., Wang, P., and Rosenfeld, A., editors, *Advances in Structural and Syntactical Pattern Recognition, SSPR'96*, volume 1121 of *LNCS*, pages 100–109. Springer-Verlag.
- Jacquin, A. (1990). Image Coding Based on Fractal Theory of Iterated Contractive Image Transforms. In *Proc. of SPIE, Visual Communications and Image Processing '90*, volume 1360.
- Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in Artificial Intelligence. Prentice Hall.
- Moller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525:533.
- Nakamura, O., Mathur, S., and Minami, T. (1991). Identification of Human Faces Based on Isodensity Maps. *Pattern Recognition*, 24:263–272.
- O’Gorman, L. (1994). Binarization and multi-thresholding of document images using connectivity. *CVGIP: Graphical Models and Image Processing*, 56(6):494–506.
- Oliver, D. (1992). *Fractal Vision, Put Fractals to Work for You*. Sams Publishing, Indiana, USA.
- Pentland, A., Moghaddam, B., and Starner, T. (1994). View-Based and Modular Eigenspaces for Face Recognition. In *Proc. of Conf. on Computer Vision and Pattern Recognition*.
- Pentland, A. P. (1984). Fractal-Based Description of Natural Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):661–674.
- R. Kasturi, K. T., editor (1995). *Basic Techniques and Symbol-Level Recognition – An Overview*, volume 1072 of *LNCS*. Springer-Verlag.
- Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Parts I & II*. MIT Press, Cambridge, Massachusetts.

23 FEATURE TRANSFORMATION STRATEGIES FOR A ROBOT LEARNING PROBLEM

Luis Seabra Lopes¹ and Luis M. Camarinha-Matos²

¹Departamento de Electrónica e Telecomunicações
Universidade de Aveiro
3810 Aveiro, Portugal
lsl@ua.pt

²Departamento de Engenharia Electrotécnica
Universidade Nova de Lisboa
2825 Monte da Caparica, Portugal
cam@uninova.pt

Abstract: This chapter illustrates, with a case study from the robotized assembly domain, the importance of feature transformation. The specific problem that is addressed is learning failure diagnosis models for a pick-and-place operation. Several feature transformation strategies are evaluated on flat as well as hierarchical learning problems. The SKIL learning algorithm, previously proposed by the authors, is used in most experiments. A comparison with an oblique tree learning algorithm is also included.

23.1 INTRODUCTION

The research described in this paper was motivated by the need to develop execution supervision functionalities for flexible assembly systems (Camarinha-Matos et al., 1996; Seabra Lopes and Camarinha-Matos, 1996).

Execution supervision is the activity of executing a plan while making sure that all its goals are achieved. Besides the lower (sub-symbolic/reactive) control levels, an architecture for intelligent supervision should possess high-level decision-making capabilities. This includes monitoring goal achievement during task execution, diagnosis when unforeseen situations are detected and planning

both for nominal task execution and for failure recovery. One main problem in developing an intelligent supervisor is the acquisition of knowledge about the task and the environment to support monitoring, diagnosis and recovery.

This is especially true in the product assembly area, where task execution must achieve complex goals (like the relationships between components in the completed assembly) defined at a symbolic level. In order to alleviate this sort of knowledge acquisition bottleneck, the use of machine learning techniques was investigated. The general problem we have been investigating can be described as *robot learning at the task level* (Seabra Lopes, 1997). An important part of the work, concerned with the induction of structured knowledge for diagnosis, evolved in the framework of the B-LEARN II project (Kaiser et al., 1995).

During the research, the authors came across the problem of having to transform the space of features that characterize each training example, in order to improve the quality of the learned models. The applied feature transformation strategies and the improvements they produced are described in this chapter. In the initial sections, the programming by demonstration framework, the applied learning algorithm and the experimental situation will be described.

23.2 ROBOT PROGRAMMING BY DEMONSTRATION

In real execution, when the monitoring function detects a deviation in the behavior of the robot that may correspond to a failure, the diagnosis function is called to confirm and characterize the failure. Diagnosis is a decision process that requires a sophisticated model of the task, the system and the environment. Based on the failure description, recovery planning can be attempted.

The problem of building such sophisticated model is not easily solved. Even the best domain expert will have difficulty in specifying the necessary mappings between the available sensors on one side and the monitoring conditions, failure classifications, failure explanations and recovery strategies on the other. Also, a few less common errors will be forgotten.

The paradigm of Robot Programming by Demonstration (RPD), adopted in the B-LEARN II project, seems indicated to overcome this type of difficulties. According to this paradigm, complex systems are programmed by showing examples of their desired behavior. In our approach, interaction with the human, seen as a tutor is fundamental. Usually, emphasis is put on robots learning from their own perception of how humans perform certain tasks (Kuniyoshi et al., 1994). In our approach, RPD is broader and includes any interaction with the human that leads to improvement in future robot performance. An adequate user interface facilitates transfer of the human's knowledge to the robot and learning capabilities enable it to generate new knowledge.

The human will carry out an initial training phase for the nominal plan execution. The traces of all testable sensors will be collected in order to generate primitive skills and the corresponding monitoring knowledge. Also in the initial training phase, the human operator may decide to provoke typical failures and collect sensor data about them. Failure classification knowledge is subsequently generated by induction. When a new failure is detected during real execution

of the assembly system, the human operator is called to classify and explain that failure and to provide a recovery strategy for the situation.

The classification phase of the diagnosis task can be performed based on knowledge generated automatically by inductive learning. A new algorithm, SKIL, was developed to perform this task. SKIL generates hierarchies of structured concepts. As far as diagnostic knowledge is concerned, the most important evaluation criterion is accuracy. Information on hierarchical problem decomposition, given to SKIL, leads to significant improvements in prediction accuracy. Another way to improve accuracy, emphasized in this chapter, is feature transformation.

The crucial step in diagnosis seems to be training the system to understand the meaning of sensor values and learn a qualitative and hierarchical model of the behavior of each operation. Programming such model would be nearly impossible. Since the human defines the "words" (discrimination features, classification attributes and respective values) used in the model, the human is capable of understanding the more or less detailed description that the model provides for each situation. It is then easier to hand-code explanations for the situations described in the model.

23.3 LEARNING STRUCTURED CONCEPTS

Having as motivation the automatic construction of the models required for the assembly supervisor, the idea of generating a concept hierarchy became attractive. The problem of learning at multiple levels of abstraction has not yet been adequately considered in the literature. In some approaches, a fixed decomposition of concepts is used, and learning is applied at each level (Koller and Sahami, 1997). However this is not flexible enough. Fixed decompositions have also been used for feature values. A new algorithm, SKIL (*structured knowledge generated by inductive learning* (Seabra Lopes and Camarinha-Matos, 1995)), was developed to perform this task.

The concepts in the hierarchy learned by SKIL are characterized by a set of symbolic classification attributes. At the lower levels of the hierarchy, concepts are described in more detail, i.e., more attribute values are specified. Moreover, in detailing or refining a concept, in which attributes take certain values, it may make sense to calculate other attributes. Therefore, the user may provide a set of attribute enabling statements of the form (A_i, a_{ij}, A_k) , meaning that when the value of A_i is determined to be a_{ij} , then attribute A_k should be included in the set of attributes to consider in the continuation of the induction process. For example, when learning the behavior of a robotized *Transfer* operation, if a collision is found, it may make sense to determine some characteristics of the colliding object, such as its size. This could be expressed by the following attribute enabling statement: $(failure_type, collision, obj_size)$.

The attribute values of the concepts in the hierarchy are determined inductively based on training data specified in terms of a set of discrimination features, which can be numerical or symbolic. Each example in the training

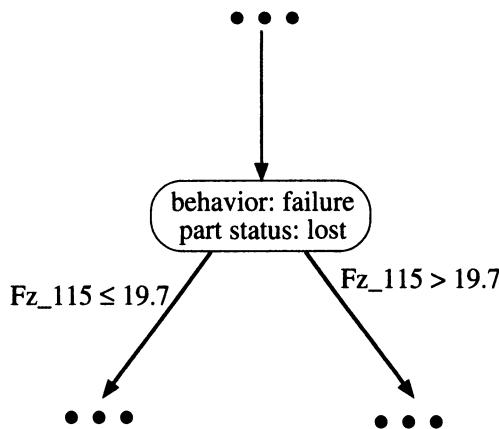


Figure 23.1 Example of a node in a concept hierarchy generated by SKIL

set is composed of the concept instance, represented as a list of attribute-value pairs, followed by a vector of feature values.

The algorithm is a recursive procedure that takes as parameters a list of examples, a list of classification attributes, a list of attribute enabling triples and a list of discrimination features. In each stage of the induction, the main goal is to predict the values of as many classification attributes as possible. For each attribute, the discrimination power of features is evaluated, using information theoretic measures. For continuous features, segmentation of feature values is done at each decision node, in a way that maximizes its discrimination power. The feature that, for some attribute, gives the highest discrimination power is selected to be test feature. Expansion stops when the values of all attributes have been predicted or when it is not possible to extract more information from the data.

The basic knowledge transmutation used by SKIL is, therefore, empirical inductive generalization, only that at multiple levels of abstraction. The generated knowledge structure is a hierarchy, where nodes represent structured concepts. A concept consists of a set of predictions. The number of predictions defines the abstraction level (concepts with less predictions are more abstract). The hierarchy is, simultaneously, a decision tree that can be used to recognize instances of the concepts. The formation of these concepts, guided by the attribute enabling statements, depends highly on the training data.

In Figure 23.1 an example of a node with two predictions and two child nodes is presented. The predictions identify a failure situation and assert that

the handled part was lost. In order to know which of the child nodes is the correct specialization of the displayed node, the feature Fz_115 must be tested.

As a special case, SKIL can also work as any traditional classification algorithm. This happens when only one classification attribute is defined. In such situations, SKIL performs equally well as (or even better than) some of the most popular classification algorithms, including CART (Breiman et al., 1984), ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993) and OC1 (Murthy et al., 1993).

23.4 CONSTRUCTIVE INDUCTION

In robotics applications, information about the status of the system must often be obtained from complex sensors that provide numerical data difficult to analyze. From the supervision point of view, the results obtained in the initial experiments carried out within B-LEARN II were not satisfactory, due to the low classification accuracies on unseen examples. The most typical accuracy results were between 50% and 70%. One obvious explanation for these poor results was that the relevant features in each situation were not explicit in the sensor data, and, if the relevant features were somehow implicit in the data, the applied learning algorithms were not powerful enough to extract them.

One solution for this problem would be to use constructive induction. The problem of constructive induction, which aims to generate hypotheses described in terms of features that do not appear in the training data, has been raised by several authors, starting with Michalski (Michalski, 1983). In this case, a method should be capable of inventing new features, by applying known functions to the initial features, and evaluating the relevance of these new features to the learning task.

Only recently, constructive induction started to grow steadily and it is certainly a good promise for the future. It is unclear if the existing techniques are already mature for complex real-world problems. Experience within B-LEARN II indicates that they are not. In the project activities, the SMART+ system (Botta and Giordana, 1993), was applied to problems from the assembly application domain. SMART+ is a sophisticated learner, capable of combining different types of inference, such as inductive, deductive and abductive inference, and able to handle numerical features and to explore background knowledge. The concept description language used in SMART+ is an extended first order Horn clause language. The predicates that make up the body of a clause can be operational (those defined by the teacher) or non-operational (invented by the learner). The possibility of the teacher defining operational predicates enables the algorithm to perform a shift of bias, i.e. enables the generation of hypotheses expressed in terms of predicates not used in the training data. The application of these predicates to the examples in the training set establishes new features potentially more discriminative with respect to the learning problem. With this idea in mind, SMART+ was applied to diagnostic problems in the assembly domain. Several serious disadvantages became apparent. To start with, the system was too complex to parameterize. Some of the learned rules were equally complex. The system was not able to handle problems with

many features. Finally, the system took a long time, typically many hours, to produce the rule set.

In numerical feature spaces, *oblique decision trees* are an alternative method of automatically deriving new features. The CART system (Breiman et al., 1984) was extended in this direction. During tree growth and in each step of the recursive partitioning procedure, instead of evaluating only the features provided in the training set, linear combinations of these features are evaluated.

The linear feature combination can be regarded as a new feature and therefore this approach can be thought of performing constructive induction. While tests on single features are equivalent to partitioning the input space by an axis-parallel hyperplane, linear feature combinations define oblique hyperplanes.

The vector of coefficients is initialized with random values and then a hill-climbing style search procedure will determine the vector of coefficients that maximizes class discrimination. The major drawback of this approach is the combinatorial explosion of possibilities that it must face. For a training set with N examples described in terms of p features, there are $\binom{N}{p}$ different splitting regions. This means that the number of alternative hyperplanes grows exponentially with the size of the training set. CART has the additional drawback that it does not necessarily find the best vector of coefficients because the search procedure can get stuck in local maxima. The more recent OC1 algorithm (Murthy et al., 1993), that will be used for comparison later in this chapter, uses randomization to escape local maxima. The specific strategy of OC1 makes it polynomial.

23.5 EXPERIMENTAL SITUATION: THE PICK AND PLACE TASK

A part pick-and-place macro-operation has been chosen as case study. For the experiments, three of the primitives involved in the operation were selected:

1. approach to grasp position (*Approach-Grasp*),
2. *Transfer* (of a part), and
3. approach to the final position (*Approach-Ungrasp*).

During the training phase, each of the selected operations was executed many times and several external exceptions were simulated. In most cases an object was placed, either in motion or stopped, in the robot arm motion path. A trace of forces and torques, covering the course of events from failure detection to stabilization of the system, was collected and a detailed failure description was assigned to the trace by the operator.

The force/torque sensor provides the features that will enable the learner to discriminate among alternative failure descriptions. Each time the sensor is consulted, the values of six variables, namely the forces and torques in the X, Y and Z directions, are obtained. This information constitutes a *sensor sample*. During the training phase, when a failure is detected, sensor samples are collected at regular time increments until the state of the system stabilizes.

During the observation window (a time interval covering the typically encountered failures) 15 sensor samples are collected. The values of a certain sensor variable (a force or a torque) in the successive samples in a failure trace make up a *profile* of that sensor variable, which is an element of \Re^{15} . Each failure trace is therefore composed of three force profiles and three torque profiles, providing a total of $(3 + 3) \times 15 = 90$ features to the learning process.

The failure description appended by the teacher to each example is the dependent variable in the learning process. When a similar failure is encountered, the robot should consider a similar failure description for further decision-making. However, these descriptions are not simple labels, as is customary in most learning systems, but structured descriptions, containing considerable detail. In the style of the previously described SKIL algorithm, each failure description consists of a set of attribute-value pairs. These attributes correspond to the classification attributes of SKIL and should not be confused with discrimination features.

In this way, 88 examples were collected for the *Approach-Grasp* operation, 47 examples for the *Transfer* operation and 117 examples for the *Approach-Ungrasp* operation. Based on these examples, it is possible to formulate a variety of learning problems.

23.6 FEATURE TRANSFORMATION STRATEGIES

The alternative followed in the experiments described in this chapter is to apply, in a pre-processing phase, a set of pre-defined feature transformation strategies. This is a highly domain-dependent approach, that remains an art and has not received enough attention from the machine learning researchers, traditionally more concerned with feature selection only. These strategies involve applying complex functions, including the discrete Fourier transform, to the raw features in order to obtain new features, hopefully more discriminative with respect to the learning problems. Most of these complex functions would hardly ever be discovered by existing constructive induction algorithms. The basic elements manipulated by feature construction functions are the profiles of the sensor variables, i.e. elements $\bar{p} = (p_1, \dots, p_{15}) \in \Re^{15}$. Each feature transformation strategy receives the profiles of forces and torques and produces a feature vector, which results of joining together (or appending) the feature vectors returned by the applied feature construction functions.

In what follows, appending two feature vectors, $\bar{f} \in \Re^m$, and $\bar{g} \in \Re^n$, denoted by (\bar{f}, \bar{g}) , is an element $\bar{h} \in \Re^{m+n}$ such that $h_i = f_i$ for $i = 1, \dots, m$ and $h_i = g_{i-m}$ for $i = m + 1, \dots, m + n$.

In general, in a numerical domain, a feature transformation strategy is a function $f : \Re^d \rightarrow \Re^k$ which transforms d -dimensional feature vectors into k -dimensional feature vectors.

It is desirable that the execution supervisor reasons about the evolution of force and torque values, not in terms of the individual numerical values, but in terms of its overall characteristics, in short, as humans do. A human, making a qualitative description of such behavior, would probably divide it

into intervals, and would mention roughly how long these intervals were, which were the average values in each interval, as well as the average derivatives. Dealing with time intervals is not an easy task, mainly when the goal is to apply feature-based learning algorithms to generate new knowledge. In the field of qualitative physics, the representations for numbers, frequently proposed, include signs, inequalities and orders of magnitude. Qualitative formalisms to describe changes in a system have also been proposed. Nevertheless, the feature transformation strategies that will be used do not produce qualitative features. The constructed features are all numeric, but they were designed to summarize potentially important aspects of the physical phenomenon under consideration. The numerical to symbolic conversion is a job for the learning algorithm itself.

One of the feature construction functions calculates the average of a certain sensor variable between two instants, m and n ($1 \leq m \leq n \leq 15$), in its profile:

$$A(\bar{p}, m, n) = \frac{\sum_{i=m}^n p_i}{n - m + 1} \quad (23.1)$$

Another function calculates the average derivative or rate of change between two instants in a given profile:

$$D(\bar{p}, m, n) = \frac{p_n - p_m}{n - m} \quad (23.2)$$

Another function tries to assess the degree of monotonicity of a profile. In this case, the general trend of the profile must be identified. The function $trend(\bar{p})$ determines if the general trend displayed by the profile is *ascending* (1) or *descending* (-1):

$$trend(\bar{p}) = \begin{cases} 1 & \text{if } A(\bar{p}, 1, 5) < A(\bar{p}, 11, 15), \\ -1 & \text{otherwise.} \end{cases} \quad (23.3)$$

The monotonicity of a profile \bar{p} is then defined as the number of increases in consecutive instants of the profile, if the general trend is ascending, or the number of decreases in consecutive instants, if the general trend is descending. Monotonicity of \bar{p} is computed by the following function:

$$M(\bar{p}) = \sum_{i=1}^{14} mon(\bar{p}, i) \quad (23.4)$$

where the function $mon(\bar{p}, i)$, that checks if the general trend of \bar{p} is followed between instants i and $i + 1$, is given by

$$mon(\bar{p}, i) = \begin{cases} 1 & \text{if } p_i \cdot trend(\bar{p}) < p_{i+1} \cdot trend(\bar{p}), \\ 0 & \text{otherwise.} \end{cases} \quad (23.5)$$

These three measures are summary measures. They are an approximation to the kind of features humans would extract from the profiles.

The measurable sensor variables are axis-parallel forces and torques. However, the resulting forces in the orthogonal planes and in 3D space might emphasize some important aspects of the physical process. For instance, the amplitude of the resulting torque in the XY plane might be more relevant to learning about vertical arm motion than the individual torques in the X and Y directions. In some of the feature transformation strategies that were experimented, two profile combination functions were applied. In both cases, the result is a profile of amplitudes of the resulting forces or torques.

The first of these functions, R_{2D} , takes as inputs two profiles, $\bar{a}, \bar{b} \in \Re^{15}$, and returns the amplitude profile $\bar{r} \in \Re^{15}$:

$$\bar{r} = R_{2D}(\bar{a}, \bar{b}) \Leftrightarrow r_i = \sqrt{a_i^2 + b_i^2}, \quad \text{for } i = 1, \dots, 15 \quad (23.6)$$

The other function, R_{3D} , computes a similar profile in 3D. It takes three profiles $\bar{a}, \bar{b}, \bar{c} \in \Re^{15}$ as input and returns the amplitude profile $\bar{r} \in \Re^{15}$:

$$\bar{r} = R_{3D}(\bar{a}, \bar{b}, \bar{c}) \Leftrightarrow r_i = \sqrt{a_i^2 + b_i^2 + c_i^2}, \quad \text{for } i = 1, \dots, 15 \quad (23.7)$$

In the selected case-study, most of the failure situations to be learned are related to different types of collisions of the robot arm with its environment. When a collision occurs, the arm enters a state of abnormal vibration that virtually disappears after a short time interval. In addition to the study in the time domain, it might be interesting to study the force and torque profiles in the frequency domain. A function of time for which an analytical expression is known can be transposed to the frequency domain by applying the *Fourier transform*. Waveforms are mathematical entities that describe important real-world phenomena (in optics, electricity, acoustics, etc.) as functions of time. In the meantime, spectra, that are the Fourier transformations of waveforms, also find physical correspondence. In several branches of science, it often happens that analysis of a given phenomenon is much simpler in the frequency domain.

There are several mathematical conditions that a function must meet in order for the Fourier transform to be applicable, but physical possibility is a valid sufficient condition for the existence of a transform.

In this case-study, the analytical expressions of the time functions represented by the profiles are not known. However, the measurements in each profile have been obtained at regular time intervals which enables the application of the *discrete Fourier transform*. For a given profile, what is obtained is a certain number of points of the Fourier transform of the underlying time function.

Let's consider the general case of a profile of a time function with an even number N of measurements taken at regular time intervals ($\bar{p} \in \Re^N$). By definition, the discrete Fourier transform of \bar{p} is (Bracewell, 1986):

$$dft(\bar{p}, \nu) = \frac{1}{N} \sum_{\tau=1}^N e^{-j2\pi(\nu/N)\tau} \cdot p_\tau \quad (23.8)$$

In this expression, ν/N is an analog of frequency. The function $dft(\bar{p}, \nu)$ is a periodic function of ν with period N . Furthermore, half of the produced values are complex conjugates of the other half. This means that, for a profile with N values, the discrete Fourier transform produces $N/2$ different amplitudes.

The feature transformation strategies, that have been used in the experiments described below, make use of the function $F(\bar{p})$, which returns the vector of amplitudes produced by the discrete Fourier transform for the profile $\bar{p} \in \Re^{15}$. Since the length of the measured profiles is 15, the period of the transform will be $N=16$ and the last value of the measured profile is repeated in position 16. $F(\bar{p})$ therefore returns an element $\bar{s} \in \Re^8$:

$$F(\bar{p}) = \bar{s} \Leftrightarrow s_\nu = \|dft((\bar{p}, p_{15}), \nu)\|, \quad \text{for } \nu = 1, \dots, 8 \quad (23.9)$$

A computer implementation of the fast Fourier transform (FFT), a method of calculating the discrete Fourier transform, has been used to obtain the amplitudes.

It is now possible to describe rigorously the feature transformation strategies that were applied. The starting point for feature transformation are the measured profiles, namely the force profiles \bar{f}_x , \bar{f}_y and \bar{f}_z and the torque profiles \bar{t}_x , \bar{t}_y and \bar{t}_z . The applied feature transformation strategies are the following (for simplicity, each strategy is identified by a code):

Strategy S1: Raw Features. The measured force and torque values constitute the features in this strategy. No feature transformation is performed.

$$S1 \equiv (\bar{f}_x, \bar{f}_y, \bar{f}_z, \bar{t}_x, \bar{t}_y, \bar{t}_z) \quad (23.10)$$

In total, $6*15=90$ features are considered.

Strategy S2: Resulting Forces and Torques. In this case, the profiles of the amplitudes of forces and torques in the orthogonal planes and in 3D space are considered:

$$\begin{aligned} \bar{f}_{xy} &= R_{2D}(\bar{f}_x, \bar{f}_y) & \bar{t}_{xy} &= R_{2D}(\bar{t}_x, \bar{t}_y) \\ \bar{f}_{xz} &= R_{2D}(\bar{f}_x, \bar{f}_z) & \bar{t}_{xz} &= R_{2D}(\bar{t}_x, \bar{t}_z) \\ \bar{f}_{yz} &= R_{2D}(\bar{f}_y, \bar{f}_z) & \bar{t}_{yz} &= R_{2D}(\bar{t}_y, \bar{t}_z) \\ \bar{f}_{xyz} &= R_{3D}(\bar{f}_x, \bar{f}_y, \bar{f}_z) & \bar{t}_{xyz} &= R_{3D}(\bar{t}_x, \bar{t}_y, \bar{t}_z) \end{aligned} \quad (23.11)$$

These profiles together with the measured profiles define this strategy:

$$S2 \equiv (\bar{f}_x, \bar{f}_y, \bar{f}_z, \bar{f}_{xy}, \bar{f}_{xz}, \bar{f}_{yz}, \bar{f}_{xyz}, \bar{t}_x, \bar{t}_y, \bar{t}_z, \bar{t}_{xy}, \bar{t}_{xz}, \bar{t}_{yz}, \bar{t}_{xyz}) \quad (23.12)$$

The number of features is in this case $14 \times 15 = 210$.

Strategy S3: Summary Features. This strategy consists of extracting the summary features of each profile. First of all, a long list of averages will be calculated for each profile \bar{p} by

$$\text{averages}(\bar{p}) = (A(\bar{p}, 1, 3), A(\bar{p}, 4, 6), A(\bar{p}, 7, 9), A(\bar{p}, 10, 12), A(\bar{p}, 13, 15), \\ A(\bar{p}, 1, 5), A(\bar{p}, 6, 10), A(\bar{p}, 11, 15), A(\bar{p}, 1, 15)) \quad (23.13)$$

where $A(\bar{p}, m, n)$ was defined in Equation 23.1. These sets of averages covers the profile \bar{p} at different granularities (3 instants, 5 instants and 15 instants). The goal is to capture global aspects of the profile as well as more localized incidents. The summary features that will be extracted from each profile are given by the following function:

$$S(\bar{p}) = (\text{averages}(\bar{p}), D(\bar{p}, 1, 8), D(\bar{p}, 8, 15), M(\bar{p})) \quad (23.14)$$

where $D(\bar{p}, m, n)$ and $M(\bar{p})$ are the derivative and monotonicity functions defined in Equations 23.2 and 23.4. Finally, the summary features extracted from each example are:

$$S3 \equiv (S(\bar{f}_x), S(\bar{f}_y), S(\bar{f}_z), S(\bar{t}_x), S(\bar{t}_y), S(\bar{t}_z)) \quad (23.15)$$

The number of features is in this case $6 \times 12 = 72$.

Strategy S4: Fourier Features. Extract the amplitudes of the harmonics generated by the fast Fourier transform for each profile:

$$S4 \equiv (F(\bar{f}_x), F(\bar{f}_y), F(\bar{f}_z), F(\bar{t}_x), F(\bar{t}_y), F(\bar{t}_z)) \quad (23.16)$$

where the function $F(\bar{p})$ was defined in Equation 23.9. This strategy produces $6*8=48$ features.

Strategy S5: All Features. Finally, all features used in the previous strategies are joined together, producing a large feature vector. The features extracted from each profile include all measurements in that profile as well as summary features and Fourier features.

The function

$$B(\bar{p}) = (\bar{p}, S(\bar{p}), F(\bar{p})) \quad (23.17)$$

returns such features for a profile $\bar{p} \in \Re^{15}$ (functions $S(\bar{p})$ and $F(\bar{p})$ were defined in Equations 23.14 and 23.9). Strategy S5 is given by

$$S5 \equiv (B(\bar{f}_x), B(\bar{f}_y), B(\bar{f}_z), B(\bar{f}_{xy}), B(\bar{f}_{xz}), B(\bar{f}_{yz}), B(\bar{f}_{xyz}), \\ B(\bar{t}_x), B(\bar{t}_y), B(\bar{t}_z), B(\bar{t}_{xy}), B(\bar{t}_{xz}), B(\bar{t}_{yz}), B(\bar{t}_{xyz})) \quad (23.18)$$

This strategy produces $14 \times (15 + 12 + 8) = 490$ features. The goal of proposing and evaluating this strategy, defined simply as the reunion/combination of the previous strategies, is to confirm (or not) the general rule that the more features are provided to the learner the better will be the result.

23.7 EXPERIMENTAL RESULTS

The experimental results that will now be presented illustrate the importance of feature transformation in improving the quality of the learned concept hierarchies. The first results were obtained on five classification problems that were designed, based on the training data collected in the experimental setup and previously described. In each of them, the goal is to learn to recognize some relevant aspect of an execution failure, maximizing accuracy and compactness of the learned knowledge. That relevant aspect that is to be learned in each case can be captured by a single classification attribute. These learning problems could, therefore, be addressed by classical concept learning algorithms, such as ID3 or CART. The presented results were obtained with SKIL and OC1. For simplicity of presentation, the learning problems are identified by a code:

- **LP-1: failures in *approach-grasp*.** Four classes of system behavior, that will be learned, are considered: *normal*, *collision*, *front collision* and *obstruction*; 88 training examples are available.
- **LP-2: failures in *transfer* of a part.** Five classes of system behavior are considered: *normal*, *front collision*, *back collision*, *collision to the right* and *collision to the left*; 47 examples are available.
- **LP-3: failures in *transfer* of a part.** Four situations of the handled part are considered: *ok*, *slightly moved*, *moved* and *lost*; the same 47 cases were used.
- **LP-4: failures in *approach-ungrasp*.** The classes of failures to be learned are: *normal*, *collision* and *obstruction*; 117 examples are available.
- **LP-5: failures in motion with part.** Five classes of system behavior are considered: *normal*, *bottom collision*, *bottom obstruction*, *collision in part* and *collision in tool*; 164 examples are available.

The results, in terms of classification accuracy (measured experimentally following a leave-one-out scheme) and size of the learned classification trees, are presented in Tables 23.1 and 23.2. In general, the more discriminative the available features are, the more accurate and compact will be the learned hypothesis. As expected, the results obtained with strategy S1, in which the sensor measurements (raw features) are directly used, are clearly the worst results. It is also clear that problems LP-2 and LP-5 are very difficult, even after feature transformation.

In terms of accuracy, strategy S3 (summary features) delivered the best results in problems LP-1, LP-3 and LP-4. In all the three cases, the improvements were substantial. In LP-1, summary features enabled an accuracy of 96%, which is 18% above the accuracy obtained with the raw features. In problem LP-4, the summary features produced an accuracy of 95%, representing an improvement of 30%. A major improvement (38%) was enabled by summary features in problem LP-3. In this case, the accuracy raised from the 49% obtained with

Table 23.1 Classification accuracy for different feature transformation strategies

<i>Learning problem</i>	<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>S4</i>	<i>S5</i>
LP-1	78	80	96	85	89
LP-2	45	57	51	68	64
LP-3	49	75	87	85	83
LP-4	65	60	95	77	83
LP-5	69	63	72	49	77
Average	61	67	80	79	79

the first strategy to 87% obtained with summary features. The best result in problem LP-2 was obtained with strategy S4 (Fourier features). In this case, accuracy was improved from 45% with the raw features to 68% with the Fourier features. Finally, problem LP-5, which turned out to be the least sensitive to feature transformation, got the best result with strategy S5 (all features).

In average terms, considering all the learning problems, strategy S3 was also the most successful, leading to an average accuracy improvement of 19%, immediately followed by strategy S5 with an average improvement of 18%. Strategy S4 was not so successful, producing an average accuracy improvement of 11.6%. Strategy S2 (measured forces and torques plus the resulting forces and torques in 2D and 3D) produced the smallest average improvement, only 5.8%. The fact that strategy S5, which includes all features used in strategy S3 and many more (490 features in total), has performed a little worse than strategy S3 can only be a contingency. It is not the general rule. In fact, our experience shows that the more features are provided to the algorithm the greater is the probability of obtaining a good result.

Table 23.2 Tree size for different feature transformation strategies

<i>Learning problem</i>	<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>S4</i>	<i>S5</i>
LP-1	9	9	6	9	8
LP-2	18	12	12	11	12
LP-3	14	10	6	10	8
LP-4	11	10	9	8	7
LP-5	28	26	14	20	11
Average	16	13	9	12	9

As far as the compactness of the learned tree is concerned (Table 23.2), strategy S5 was, actually, the most successful on the average of all learning problems, immediately followed by strategy S3. The two major improvements in compactness were precisely obtained with these strategies. Strategy S3 en-

abled the generation of a tree with 6 nodes for problem LP-3, less than half the size of the tree obtained with the raw features. Strategy S5 enabled the generation of a tree with 11 nodes for problem LP-5, much smaller than the 28 nodes tree obtained with the raw features.

In Table 23.3, a comparison with OC1 (Murthy et al., 1993) is presented. This algorithm can be used to generate oblique trees, whose tests are based on linear combinations of features, as well as classical axis-parallel trees. The linear feature combinations can be seen as automatically learned features. It is, therefore, interesting to compare this automated approach to the approach followed in this chapter, based on pre-defined feature transformation strategies. From the results obtained, we see that the performances of OC1 (axis-parallel and oblique) and SKIL on the five learning problems previously presented are equivalent: the average accuracy is of 61% in the three cases. Therefore, the constructive nature of OC1-oblique didn't bring the expected improvements. This is even more disappointing considering the long running time required by OC1. In fact, to generate oblique trees and perform the leave-one-out test for the five learning problems, OC1 took more than 30 hours. To generate its hierarchies, SKIL needed around one hour only. In contrast with the lack of success of oblique trees, the feature transformation strategies that were applied enabled significant improvements in accuracy. As can be seen from Table 23.3 it was not difficult to get improvements of 20% or more.

Table 23.3 Comparison of various approaches

Approach	LP-1	LP-2	LP-3	LP-4	LP-5	Average
OC1 (axis-parallel)	65	43	70	70	58	61
OC1 (oblique)	65	49	57	80	53	61
SKIL (no feature transf.)	78	45	49	65	69	61
SKIL (2nd best results)	89	64	85	83	72	79
SKIL (best results)	96	68	87	95	77	85

Another experiment was conducted, in this case with a structured concept learning problem. This is the kind of problem that classical concept learning algorithms cannot handle. The training data used in this experiment consists of 88 examples of failures in the *approach_grasp* operation.

Ten classification attributes were considered:

- *behavior*: generic information about the operation behavior; it can be *normal* or *failure*; in the second case, the attributes *failure type*, *affected body*, *object size*, *object hardness* and *object weight* become enabled.
- *phase*: phase of the failed operation; it can be *initial*, *middle* or *terminal*.
- *failure type*: a classification or identification of the failure, for instance *collision*, *front collision* or *obstruction*.

- *affected body*: what was involved in or more affected by the failure: it can be the *tool*, the *tool tubes* or the *fingers*; if it is *tool* then the attribute *tool region* becomes enabled and if it is *fingers* then the attribute *fingers region* becomes enabled.
- *tool region*: region of the tool that was affected: it can be *front*, *left*, *right* and *back* sides, and *bottom*; in the last case the attribute *bottom subregion* becomes enabled.
- *bottom subregion*: if the failure affected the bottom part of the tool, where exactly was it; it may be *front*, *middle* or *back*.
- *fingers region*: region of the fingers that was affected, including *left finger*, *right finger* and *both fingers*.
- *object size*: size of object causing failure: *small*, *large*.
- *object hardness*: can be *soft* or *hard*.
- *object weight*: can be *low* or *high*.

Since not all attributes are enabled at the same time, the maximum size of a concept is of 8 predictions. The average concept size in the leaf nodes of a fully grown hierarchy is of 4.7 predictions (the method of calculating this figure is omitted here). The goal of this experiment is to evaluate the impact of feature transformation, not only in terms of the accuracy and compactness of the learned hierarchy, but also in terms of its *coverage* of the conceptual space. Coverage is defined as the ratio between the average concept size in the leaf nodes of the generated hierarchy and the average concept size in the leaf nodes of a fully grown tree. For instance, if the average concept size in the leaf nodes of the hierarchy generated for the problem above is 2.35, its coverage will be $2.35/4.7 = 50\%$.

Coverage may be much less than unity when the training set does not contain enough information. In that case, expansion of the tree is stopped before predictions have been determined for all attributes. The particular stopping criterion that was used is to discard every prediction for which the anticipated accuracy is not above 70%. The results obtained for this structured learning problem are shown in Table 23.4. This time, strategy S5 (all features) was consistently the most successful, followed by strategies S3 (summary features) and S4 (Fourier features). Since the minimum allowed accuracy for a prediction is of 70%, the average accuracy is high, even in the case of strategy S1. What varies more clearly is coverage of the conceptual space. While using only the raw features the coverage is only 46%, using the Fourier features (strategy S4) or all features (strategy S5) raises coverage to 66%.

Typically, the predictions added in the lower levels of a concept hierarchy have lower accuracy. This means that, if the stopping criterion is very permissive, the hierarchy will be large but not very accurate. On the other end, if the stopping criterion is very severe, the hierarchy will be more compact

and accurate, but less informative (less predictions, smaller coverage). In the meantime, if the number and the discrimination power of the features used in describing the examples increase, it will be possible to accept more predictions and create more nodes. But, simultaneously, it will be possible to optimize the size of the hierarchy. For this reason, there is no clear rule to explain the impact of features in the size of the hierarchy. Accuracy increases (between 3% and 10% in Table 23.4) when better features are provided. This is the obvious effect of providing more information for learning while keeping the threshold for prediction acceptance. In structured problems, coverage is the measure with the more direct dependence on the quality of discrimination features.

Table 23.4 Impact of feature transformation on a structured problem

	<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>S4</i>	<i>S5</i>
Experimental accuracy (%)	82	85	90	87	92
Coverage (%)	46	59	52	66	66
Size (nodes)	18	25	21	19	22

23.8 CONCLUSIONS

This chapter described the application of feature transformation to the problem of learning diagnosis knowledge about execution failures in robotics. The original features are time dependent and the transformation strategies take that into account.

Some experiments with constructive induction algorithms were performed (and only partially described above). Since no good results were obtained, we tried the alternative approach of using pre-programmed feature transformation strategies. These strategies were developed specifically for our application. The size of the feature vectors after feature transformation varies between 48 and 490 features. Summary features, like averages and slopes, extracted from sensor traces, were quite successful in improving the accuracy and compactness of the learned hierarchies. Frequencies, extracted by the fast Fourier transform, did also contribute to improve accuracy, although to a lesser extent. In some learning problems, feature transformation led to accuracy improvements of 30% or more and to reductions in tree size to half the size of the trees generated with raw features only.

It must be noted that feature transformation is a highly domain-dependent topic. The transformation strategies that we developed for force-based diagnosis in assembly can give inspiration to develop methods for other domains, but probably won't give the best results if directly applied. What was described is no more than a case study. We believe that future developments in feature transformation should be concerned in identifying major classes of objects and major feature transformation strategies for those objects. One example of such

objects in our particular case study are the force/torque profiles. Profiles appear in many domains and, independently of the particular variables whose evolution they describe, there are a certain number of features that may make sense to extract. It seems, therefore, that a good methodology for feature transformation should look at examples not as feature vectors but as collections of standard objects. Precisely, in our case study an example was a collection of six profiles.

References

- Botta, M. and Giordana, A. (1993). Smart+: A multi-strategy learning tool. In *Proc. International Joint Conference on Artificial Intelligence*, pages 937–943.
- Bracewell, R. (1986). *The Fourier Transform and its Applications*. McGraw-Hill, Singapore.
- Breiman, L., Friedman, J., Oleshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsforth International Group.
- Camarinha-Matos, L., Seabra Lopes, L., and Barata, J. (1996). Integration and learning in supervision of flexible assembly systems. *IEEE Transactions on Robotics and Automation*, 12:202–219.
- Kaiser, M., Giordana, A., Nuttin, M., and Seabra Lopes, L. (1995). B-learn ii: Combining sensing and action (final project report: Short version). Technical report, ESPRIT BRA 7274 B-LEARN II.
- Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the International Conference on Machine Learning*, pages 170–178.
- Kuniyoshi, Y., Inaba, M., and Inoue, H. (1994). Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10:799–822.
- Michalski, R. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161.
- Murthy, S., Kasif, S., Salzberg, S., and Beigel, R. (1993). Oc1: Randomized induction of oblique decision trees. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*.
- Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.
- Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Francisco, California.
- Seabra Lopes, L. (1997). *Robot Learning at the Task Level. A Case Study in the Assembly Domain (Ph.D. Thesis)*. Universidade Nova de Lisboa, Portugal.
- Seabra Lopes, L. and Camarinha-Matos, L. (1995). Inductive generation of diagnostic knowledge for autonomous assembly. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2545–2545.
- Seabra Lopes, L. and Camarinha-Matos, L. (1996). Learning failure recovery knowledge for mechanical assembly. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*.

24 INTERACTIVE GENETIC ALGORITHM BASED FEATURE SELECTION AND ITS APPLICATION TO MARKETING DATA ANALYSIS

Takao Terano¹ and Yoko Ishino²

¹ Graduate School of Systems Management,
Tsukuba University
terano@gssm.otsuka.tsukuba.ac.jp

² Graduate School of Engineering,
University of Tokyo
ishino@ai.rcast.u-tokyo.ac.jp

Abstract: Interactive Genetic Algorithms (IGAs) are subjective and interactive methods to evaluate the qualities of offspring generated by genetic operations. We adopt IGA in order to select relevant features in inductive learning problems. The method we propose is used to extract efficient decision knowledge from noisy questionnaire data in marketing data analysis domain. Unlike popular learning-from-example methods, in such tasks, we must interpret the characteristics of the data without clear features of the data nor pre-determined evaluation criteria. The problem is how domain experts get simple, easy-to-understand, and accurate knowledge from noisy data. The proposed method has been qualitatively and quantitatively validated by a case study on consumer product questionnaire data: the acquired rules are simpler than the results from the direct application of inductive learning; a domain expert admits that they are easy to understand; and they are at the same level on the accuracy compared with the other statistical methods.¹

24.1 INTRODUCTION

Marketing decision making tasks require the acquisition of efficient decision rules from noisy questionnaire data. These are major application fields in data mining or knowledge discovery in databases (Fayyad, 1996). Unlike popular learning-from-example methods, in such tasks, we must interpret the characteristics of the data without clear features of the data nor pre-determined evaluation criteria.

Traditionally, statistical methods have been used for these analyses, however, conventional techniques in statistics are too weak because they usually assume the linearity of the models and the form of distributions of the data. For example, multivariate analysis (Kendall, 1980) has been a main statistical tool used for analysis of questionnaire data. During the analysis, emphasis has been placed on understanding trends after identifying target data. A marketing managers also requires quantitative as well as qualitative analysis (Aaker, 1980). There are no statistical tools to facilitate to satisfy both requirements simultaneously.

On the other hand, AI-based techniques for concept learning do not assume such conditions, however, they involve combinatorial feature selection problems. The objective of the data analysis is not to develop a consultation system for classificatory decision making, but to extract hidden simple knowledge to explain what the respondents think. Therefore, it is critical to interactively evaluate the resulting decision rules.

To overcome the problem, we have developed an interactive method **SIBILE**² to acquire efficient decision rules from questionnaire data using both *interactive Genetic Algorithms* (IGAs) and inductive learning techniques (Terano, 1995). IGA or simulated breeding is one of the GA-based techniques which subjectively or interactively evaluate the qualities of offspring generated by genetic operations with human-in-a-loop manner (Unemi, 1994).

SIBILE works well to extract simple, explainable, and accurate information from the data (Terano, 1995; Terano, 1996). However, one of the remaining issues to use **SIBILE** is that it requires knowledge intensive tasks to evaluate the offspring generated by inductive learning and genetic operations. To improve the difficulty of **SIBILE**, we develop a half-automated version of **SIBILE**: first, in the interactive phase, we subjectively evaluate decision trees or sets of decision rules to get the biases of features in the data as is used in (Terano, 1995), then in the automated phase, using the biases, conventional automated genetic operations are applied to develop offspring with 'effective' features. By the word 'effective', we mean the resulting decision trees or sets of rules are simple, accurate, and easy-to-understand.

This chapter is organized as follows: In Section 24.2, we briefly introduce the concepts and features of interactive GAs and the target domain. In Section 24.3, we explain the procedure of the proposed method. In Section 24.4, we demonstrate the results of experimental results conducted to validate the effectiveness of the method. In Section 24.5, we give some concluding remarks.

24.2 INTERACTIVE GAS, FEATURE SELECTION, AND KNOWLEDGE EXTRACTION

24.2.1 Interactive Genetic Algorithms

Genetic algorithms are considered to be a good gear to practically solve feature selection problems in the literature (e.g., (Bala, 1994; Bala, 1995; Vafaei, 1994)). To use GA techniques, it is necessary to clearly define the objective function in advance (e.g., (Davis, 1991)). However, for problems where human subjective judgments play an important role, the definition of evaluation function is not an easy task. If we were able to have well-defined objective functions to be optimized in the feature selection, we could apply conventional GA techniques to choose appropriate features. In our task domain, however, it is not the case.

The idea of IGAs or *simulated breeding*(Unemi, 1994) is similar to the ones of *simulated evolution* or *interactive evolution* in computer graphics arts (Sims, 1992) in the ALife literature. In both methods, individuals judged by human experts or users to have some *efficient* features are allowed to breed their offspring. The judgments are subjectively or interactively done. In such cases where the evaluation function is not clearly defined, IGAs are able to improve breeds by selecting the parent for the next generation from among the phenotypes developed based on human preference.

An example of an earlier system related to simulated breeding was developed by R. Dawkins in the Blind Watchmaker (Dawkins, 1986). Dawkins has shown that through use of form constraints, such as left and right side symmetry and segment structures, which are both often since in living objects, figures resembling living objects can be developed.

24.2.2 Feature Transformation in Machine Learning

The method most representative of inductive machine learning is ID3 (Quinlan, 1986) which gives a decision tree or a set of decision rules as an output for the results of classification learning analysis on features and attribute-value pairs. Our research adopts C4.5 (Quinlan, 1993) a noise tolerant successor of ID3. As are stated in (Weiss, 1991), inductive learning or concept learning techniques in Artificial Intelligence, and classification of data via statistical methods (for example, the linear discrimination method) will give similar classification results, if we are able to assume the linear distribution of sample data. The performance of the results are compatible. However, we believe that the explainability of the results for domain experts from machine learning is better than the one from statistical methods.

On the other hand, machine learning which attempts to incorporate all features in a decision tree is too complex (Mingers, 1989a; Mingers, 1989b). Hence, transformation of appropriate features becomes necessary. Various studies have been conducted to deal with the problem of interactions between features (

(Kira, 1992; Liu, 1994; John, 1994; Hirsh, 1994; Schaffer, 1993)). Feature transformation tasks intrinsically have combinatorial characteristics.

24.2.3 Task Domain Description

In a saturated market domain such as oral care products, marketing decision analysts as domain experts must determine the promotion strategies of new products according to the abstract image of the products to be produced. However, in the task domain, although we can only gather noisy sample data with complicated models, it is critical to get simple but clear rules to explain the characteristics of the products in order to make decisions for promotion.

Furthermore, the actual features of the product to realize the product image are often left to the discretion or intuition of the experts and are determined in a very unclear manner based on the questionnaire data of candidate users. It is necessary to organize the information in a simple and useful format in order to understand the relationship between product images and features from the results of a questionnaire, and in order to use this understanding in decision making processes.

Summing up, the difficult points of the research are that 1) the questionnaire data intrinsically involve noises, 2) a distribution of data cannot be previously assumed, 3) selection of appropriate features of the data is inevitable, because of the difficulty in interpreting the results of analysis incorporating all the various features, and 4) we do not know how to define the evaluation criteria in advance for effective explanation.

24.3 ALGORITHM OF SIBILE

The procedure of the proposed method is shown in Figure 24.1. Some additional descriptions are given in the following:

Step 1: Initialization We generate the initial population, that is, we select m sets of individuals with less than or equal to l features. The m and l respectively represent the number of individuals and the length of their chromosomes. In our current implementation, m and l are respectively set to 6 and 16. The chromosomes to represent the features are coded in binary strings, in which a '1' (respectively '0') means that a feature is (not) selected for inclusion in the inductive learning process.

Interactive Phase

Step 2: Apply Inductive Inferences Inductive learning is applied to each of the m individuals with the selected features suggested as '1' in the chromosome. The data acquired from the questionnaire is aggregated, each of which has the corresponding features in it. Then the m sets of the data are processed by inductive learning programs. As stated earlier, we use C4.5 programs with

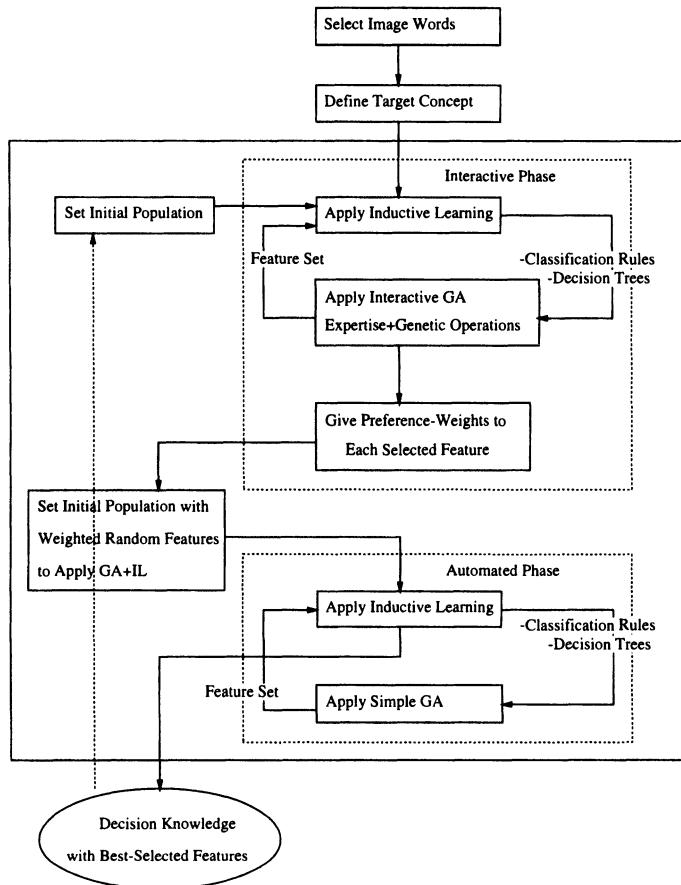


Figure 24.1 SIBILE Algorithm with Interactive- and Automated-Phases

standard parameters. As a result, m sets of decision trees with selected features or the corresponding set of decision rules are generated.

Step 3: Interaction with Users In this step, a user or a domain expert must interact with the system. This is a highly knowledge-intensive task. Observing the forms of the decision trees, set of decision rules, and combinations of selected features, the domain expert subjectively and interactively evaluate the intermediate results to explain the characteristics of the predetermined n image words.

He or she specifies 'good' features following two ways:

- Selection of decision trees
He or she selects 'good' two decision trees. The feature set in the se-

lected trees are set to as parents for genetic operation, and new product characteristics are determined by them.

The selected two parents are preserved for the next generation. The rest $m-2$ offspring are replaced by the corresponding new $m-2$ offspring.

- Selection of good rules

He or she selects 'good' rules generated from decision trees. Features in the decision rules become candidates of the next generations. The results will reflect the preference weight of each feature.

We apply modified uniform-crossover operations to them in order to get new sets of features to broaden the variety of offspring. The modified point is that the features are stochastically selected based on their preference weight values.

Step 4: Give Preference Weights to Each Selected Feature Based on the judgment in Step 3, the preference weights are modified. If the user selects decision trees with good features, the preference weights of the corresponding features are equally increased. If the user selects some of the decision rules, the preference weights of the corresponding features in the rules are increased based on the user-specified preference values (in our current implementation) from -3 to +3.

Step 5: Repeat the Steps Steps 2 to 4 are repeated until an appropriate decision tree or set of decision rules is obtained. As are illustrated in (Dawkins, 1986) and (Unemi, 1994), the steps required to obtain the appropriate results are very small. In our experiments, it usually takes only less than 10 steps.

Automated Phase

Step 6: Set Initial Population with Weighted Random Features Based on the preference weights interactively determined in *Interactive Phase*, the chromosomes or features are randomly selected to apply genetic operations. We generate the initial population, that is, we select m ' sets of individuals with less than or equal to l features. The In our implementation, m ' and l are respectively set to 12 and 16.

Step 7: Apply Inductive Learning The procedure is same as **Step 2**.

Step 8: Apply Simple Genetic Algorithm Each individual is evaluated by the following simple fitness function:

$$F(T) = \alpha A + \beta B + \gamma C + \delta D,$$

where, T means a corresponding decision tree, A = (number of tree nodes with all features)/(number of nodes of T), B = (number of rules to classify the data positive), C = (accuracy of T to classify the data positive), D = (accuracy of top three rules generated from T to classify the data positive). α , β , γ , and δ

are parameters experimentally determined. In our experiments, we set $\alpha = 0.3$, $\beta = 1.5$, $\gamma = 28.0$, and $\delta = 0.2$. As a result, A , B , C , and D contribute $F(t)$ at the same level of magnitudes.

We adopt the following genetic operations based on Simple GA in (Goldberg, 1989): Conventional uniform crossover, Mutation rate: 0.1, Generational replacement, and The best two parents preserved.

The mutation rate is higher than the one with conventional GAs. This is because the features the user selected will tend to be employed in the offspring. The probability that the features which the user specified in *Interactive Phase* appear in the next generation is proportional to both the mutation rate and the times they are selected in the interaction.

Step 9: Repeat the Steps Steps 7 to 8 are repeated until an appropriate decision tree or set of decision rules is obtained. The steps required to obtain the appropriate results are approximately equal to 10-30. If the results are not satisfactory when *Automated Phase* converges, return to *Interactive Phase*.

24.4 EXPERIMENTS

To validate the effectiveness of the proposed method, we have carried out intensive experiments from a practical case study on consumer product questionnaire data. The data set is the same one used in (Terano, 1995; Terano, 1996). This section describes the experimental results.

24.4.1 Method

Questionnaire data to investigate the features of new products in a manufacturing company was used as a case study of the proposed method. The experimental methods are summarized as follows.

Questionnaire used: Questionnaire survey conducted with 2,300 respondents by a manufacturing company in 1993 regarding oral care products. The number of cases is usual for the questionnaire data in the task domain, and is large enough to apply the proposed method.

Domain Expert: The resulting knowledge was evaluated by a domain expert who is concerned with marketing analysis on the task domain at the manufacturing company. She knows the basic principles of inductive learning programs, statistical techniques, and is able to understand the output results. Using the output forms of decision trees and corresponding rule sets from C4.5 programs, she has been required to interactively and subjectively evaluate the quality of the acquired knowledge from the viewpoints of simplicity, understandability, accuracy, reliability, plausibility, and applicability of the knowledge.

Experimental Methods and Implementation:

- 16 image words were selected to define product image. Respondents of the questionnaire evaluated how well each of the 16 image words fit the categories (*Fit*, *Moderate*, and *Does not Fit*, which will be respectively denoted as O, M, and X in the following sections, figures, and tables) of the toothpaste brand they mainly use.
- 16 features words were selected for the evaluation. Respondents of the questionnaire evaluated whether they were *satisfied* or *not satisfied* with their toothpaste brand with regards to each of the 16 features. Therefore, the size of the search space is 2^{16} , which seems small to use Genetic Algorithms, however, it is large enough for using Simulated Breeding. For example, the size is as same as the ones in the researches which solve feature selection problems using GA techniques. This means that the selection of appropriate features within only 16 candidates requires various kinds of efforts.
- The current experimental system **SIBLE** is implemented on a Windows-Based Personal Computer. The GA programs were written in C++ language, and C4.5 programs are used as an Inductive learning tool. The sample display is shown in Figure 24.2.

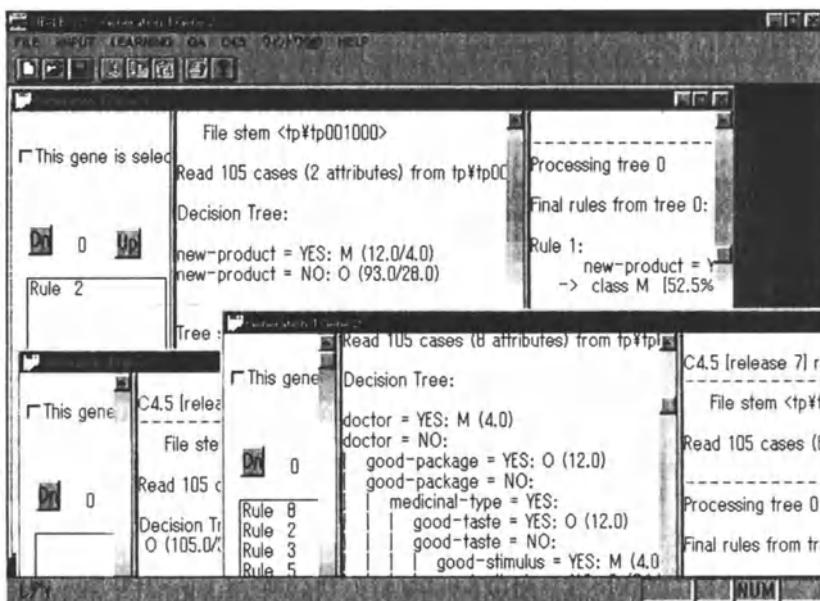


Figure 24.2 Sample Display of **SIBLE** on a Window Based PC

24.4.2 Results

This subsection presents the experimental results to extract knowledge to simultaneously explain both selected images: *innovative* and *effective*.

Prior to the experiment, as an initial investigation, we have applied C4.5 programs to the original data with all 16 features. As a result, we have got a huge *pruned* decision tree with 113 nodes, which was impossible for even an experienced human expert to correctly interpret. Furthermore, on the other hand, using only *Automated Phase* of the algorithm in Section 24.3, that is, using Simple GA, we have obtained a decision tree with 50-70 nodes. These results suggest that simple application of inductive learning and/or genetic algorithms do fail to acquire 'good' knowledge from the questionnaire data.

		FEATURE SET														Selected or Best Offspring
Generations																
Gen 0	Good for Combination	Good Packaged	Good for Family Use	Recommended by Friends	Recommended by Dentists	Stimulating Taste	Familiar Goods	Good Taste and Flavors	New Products	Reliable Manufacturer	Favorite Commercials	Frequent Commercials	Bubbly Paste	Liquid Type	Medical Type	Have Unique Characteristics.
	0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0	▲	0 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0	▲	0 1 0 1 0 1 1 1 1 0 1 0 0 0 0 0	▲	0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1	▲	0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0	▲	1 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1	▲	0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0	▲	0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0	▲
	0 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0	▲	0 1 0 1 1 0 1 1 1 0 0 0 0 1 0 0	▲	0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0	▲	0 1 0 1 0 1 1 0 0 0 1 1 1 1 0 0	▲	0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0	▲	1 1 0 1 0 1 1 1 1 0 0 0 0 1 0 0	▲	0 1 0 0 1 0 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0	▲
	0 1 0 0 1 0 1 1 1 1 0 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0	▲
	0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0	▲
	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0	▲
	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0	▲	1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0	▲	1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0	▲
Gen 1	Good for Combination	Good Packaged	Good for Family Use	Recommended by Friends	Recommended by Dentists	Stimulating Taste	Familiar Goods	Good Taste and Flavors	New Products	Reliable Manufacturer	Favorite Commercials	Frequent Commercials	Bubbly Paste	Liquid Type	Medical Type	Have Unique Characteristics.
	1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0	▲
	1 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 1 1 1 1 1 0 0 0 0 1 0	▲
	1 0 1 0 1 1 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 1 1 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 1 1 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0	▲	1 0 1 0 1 1 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0	▲
	1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0	▲	1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0	▲	1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0	▲	1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0	▲	1 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0	▲
	1 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0	▲	1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 0	▲	1 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0	▲	1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 0	▲	1 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0	▲	1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 0	▲	1 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0	▲	1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 0	▲
Gen 2	Good for Combination	Good Packaged	Good for Family Use	Recommended by Friends	Recommended by Dentists	Stimulating Taste	Familiar Goods	Good Taste and Flavors	New Products	Reliable Manufacturer	Favorite Commercials	Frequent Commercials	Bubbly Paste	Liquid Type	Medical Type	Have Unique Characteristics.
	0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0	○	1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0	○	1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0	○	1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0	○	1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0	○	1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0	○	1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0	○	1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0	○
	1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0	○	1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0	○	1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0	○	1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0	○	1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○
	1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0	○	1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○
	1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0	○	1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0	○	1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0	○	1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0	○	1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0	○
Gen 3	Good for Combination	Good Packaged	Good for Family Use	Recommended by Friends	Recommended by Dentists	Stimulating Taste	Familiar Goods	Good Taste and Flavors	New Products	Reliable Manufacturer	Favorite Commercials	Frequent Commercials	Bubbly Paste	Liquid Type	Medical Type	Have Unique Characteristics.
	1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0	●	1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0	●	1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0	●	1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0	●	1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0	●	1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0	●	1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0	●	1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0	●
	1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0	●	1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0	●	1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0	●	1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0	●	1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0	●	1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0	●	1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0	●	1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0	●
	1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0	●	1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0	●	1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0	●	1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0	●	1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0	●	1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0	●	1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0	●	1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0	●
	1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0	●	1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1	●	1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1	●	1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1	●	1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1	●	1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1	●	1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1	●	1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1	●

Figure 24.3 Summary of Offspring of Interactive Phase

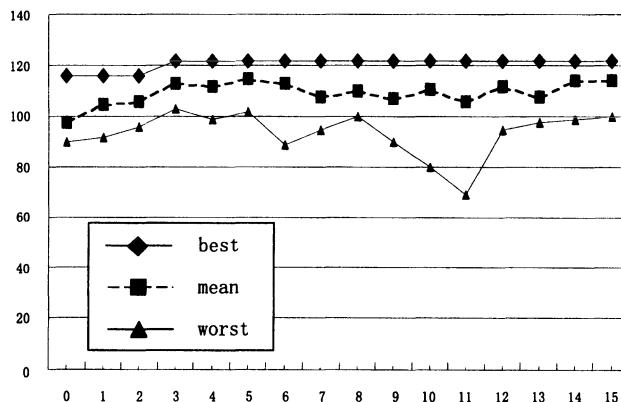
**Figure 24.4** Changes of Fitness Values in Automated Phase

Figure 24.3 shows the results of *Interactive Phase* of the experiment. The figure shows the changes in the selected product characteristics (bit strings) The figure also shows the characteristics of offspring in each generation (Gen-x): '1' or '0' respectively mean selected or unselected features in each offspring. Black triangles and circles respectively mean selected rules and decision trees to make offspring In the experiment, we iterate 3 generations in *Interactive Phase*. Figure 24.4 shows the changes of fitness functions in *Automated Phase*. Although the highest fitness values do not change remarkably, the feature set in the decision trees does not converge until 15 iterations.

Table 24.1 Accuracy Comparison of Direct Application of C4.5 and the Proposed Method

Method	C4.5 All	C4.5 + IGA + GA
Number of Features	16	5
Size of Decision Tree	113	6
Total Accuracy	57.3%	53.2%
Class O Accuracy	51.8%	42.8%
Class M Accuracy	81.2%	80.3%
Class X Accuracy	0.0%	0.0%

The final results of the decision tree and the corresponding rules are shown in Figure 24.5. The decision tree is represented in the form of standard outputs of C4.5 programs. The top left node means the top decision node. The lower decision nodes follow to down and right directions. Please refer to (Quinlan,

Resulting Decision Tree:

```

characteristics = YES: O      (293.0/117.3)
characteristics = NO:
|   medical - type = NO:
|   |   liquid = YES: O      (32.0/16.5)
|   |   liquid = NO: M      (1560.0/782.2)

```

Resulting Decision Rules:

Rule	Rule
medical - type = YES	medical - type = YES
liquid = Yes	maker - value = YES
family - use = YES	-> class 0 [61.5%]
-> class 0 [85.7%]	
Rule 10:	
characteristics = NO	characteristics = YES
liquid = YES	maker - value = NO
maker - value = YES	family -
family - use = YES	-> class 0 [58.5%]
-> class 0 [77.7%]	
Rule	Rule
characteristics = YES	medical - type = YES
liquid = NO	family -
family - use = YES	-> class 0 [57.0%]
-> class 0 [63.0%]	
Rule	Rule 13:
characteristics = YES	liquid = YES
maker - value = YES	maker - value = NO
-> class 0 [62.2%]	family -
	-> class 0 [53.9%]

Figure 24.5 Resulting Decision Tree and Corresponding Rules

1993) for the detail explanations. A human expert judges that these acquired rules are easy-to-understand and simple enough to make practical marketing decisions. The resulting rules coincide with some actual decisions made by the other experts in the company.

Table 24.1 shows the accuracy comparison results of the tree with all features and the resulting tree generated by the experiment. About the further accuracy comparisons, we have demonstrated the superiority of the proposed method in the other literature (Terano, 1996), in which we have compared the proposed method with the other statistical methods: linear discrimination (LD) and automatic interaction detection (AID). The experimental results using *SAS* (*SAS*, 1985) and *S* (Clark, 1992) packages are also summarized in Table 24.2.

In our task domain, the total accuracy of the resulting rules and the accuracy for Class 0 are critical to get decision knowledge. Keeping this in mind, the figure suggests that the proposed method shows the same level of accuracy

Table 24.2 Accuracy Comparison of C4.5, LD, and AID

<i>Method</i>	<i>C4.5</i>	<i>LD</i>	<i>AID</i>	<i>C4.5</i>	<i>LD</i>	<i>C4.5</i>	<i>LD</i>
<i>Number of Features</i>	16	16	16	7 (ex. 1)	7 (ex. 1)	8 (ex. 2)	8 (ex. 2)
Total Accuracy	57.3%	41.4%	56.0%	51.4%	40.6%	52.4%	33.9%
Class O Accuracy	51.8%	48.2%	61.3%	41.3%	37.2%	45.3%	50.4%
Class M Accuracy	81.2%	34.5%	69.4%	77.5%	43.8%	76.1%	8.2%
Class X Accuracy	0.0%	43.5%	0.9%	0.0%	40.0%	0.0%	66.5%

compared with the tree with all features, in spite that the resulting tree is so simple and very easy to understand.

24.5 CONCLUDING REMARKS

We have proposed a novel method to acquire efficient decision rules from questionnaire data by using inductive learning and genetic algorithms with interactive- and automated-phases. Using practical questionnaire data on marketing decision making, we have also shown the evolutionary computation techniques such as IGAs can be applied to practical knowledge engineering and data mining problems.

Researches to integrate inductive learning and genetic algorithms are not quite novel in multistrategy learning literatures (De Jong, 1991; Vafaie, 1994; Bala, 1994; Bala, 1995). For example, (Vafaie, 1994) has investigated automated feature selection for inductive learning program AQ with GA-based techniques. (Bala, 1994) has proposed another multistrategy approach to use GA to improve the performance of classification rules generated by AQ. The task domain is the classification of texture images with 18 features. The objective function of GA is defined based on the accuracy of the decision rules or classifiers. However, these researches differ from our problem in the aspect that we must determined the form of the objective functions to be evaluated in the interactive phase.

The prerequisites of the proposed method are quite simple and the algorithm is easy to implement. Future directions of the work include to generalize **SIBILE** for a portable tool applicable to the other decision making problems and to specify **SIBILE** for efficient decision knowledge acquisition by improving the inductive learning techniques.

Notes

1. This research is supported in part by System Theory of Function Emergence under the Grant-in-Aid for Scientific Research on Priority Area of the Ministry of Education, Science, Sports and Culture of Japan.

2. **SIBILE** is a sibyl in old French, and stands for Simulated Breeding and Inductive Learning Environment.

References

- Aaker, D. A., Day, G. S. (1980). *Marketing Research: Private and Public Sector Decisions*. John-Wiley.
- Bala, J. W., De Jong, K, Pachowicz, P. W. (1994). Multistrategy Learning from Engineering Data by Integrating Inductive Generalization and Genetic Algorithms. in Michalski, R., Tecuci, G. (eds.) (1994). *Machine Learning IV: A Multistrategy Approach*. Morgan-Kaufmann, pp. 471-488.
- Bala, J., De Jong, K., Huang, J., Vafaie, H., and Wechsler, H. (1995). Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification. *14th Proceedings of the International Joint Conference on Artificial Intelligence* . pp. 719-724.
- Clark, L. A., Pregibon, D.(1992). Tree-Based Model. in Chambers, J.M., Hastie T. J., (eds.) *Statistical Models in S* AT&T Bell Laboratories.
- Davis, L. (ed.).(1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.
- Dawkins, R.(1986). *The Blind Watchmaker*. W. W. Norton.
- De Jong, K. A., Spears, W. M.(1991). Learning Concept Classification Rules Using Genetic Algorithms. *12th Proceedings of the International Joint Conference on Artificial Intelligence* , pp. 651-656.
- Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (eds.). (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press.
- Goldberg, D. E.(1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- Hirsh, H., Noordewier, M.(1994). Using Background Knowledge to Improve Inductive Learning: A Case Study in Molecular Biology. *IEEE Expert*, Vol. 9, No. 5, pp. 3-6.
- John, G. H., Kohavi, R., Pfleger, K.(1994). Irrelevant Feature and the Subset Selection Problem. *Proc. Machine Learning-94*, pp. 121-128.
- Kendall, M.(1980). *Multivariate Analysis, 2nd Edition*. Charles Griffin.
- Kira, K., Rendell, L. A.: The Feature Selection Problem.(1992). Traditional Methods and a New Algorithm. *Proc. AAAI'92*, pp. 129-134.
- Liu, W. Z., White, A. P.(1994). The Importance of Attribute Selection Measures in Decision Tree Induction. *Machine Learning*, Vol. 15, pp. 25-41.
- Mingers, J.(1989a). An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning*, Vol. 3, pp. 319-342.
- Mingers, J.(1989b). An Empirical Comparison of Pruning Methods for Decision-Tree Induction. *Machine Learning*, Vol. 4, pp. 227-243.
- Quinlan, J. R. (1986). Induction for Decision Trees. *Machine Learning* 1-1: 81-106
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan-Kaufmann.

- SAS (1985). *SAS User's Guide*. SAS Institute Inc.
- Schaffer, C.(1993). Overfitting Avoidance as Bias. *Machine Learning*, Vol. 10, pp. 153-178.
- Sims, K.: Interactive Evolution of Dynamical Systems. in Varela, F. J., Bourgine, P. (eds.).(1992). *Toward a Practice of Autonomous Systems - Proc. 1st European Conf. Artificial Life*, MIT Press, pp. 171-178.
- Terano , T., Ishino, Y., Yoshinaga, K.(1995). Integrating Machine Learning and Simulated Breeding Techniques to Analyze the Characteristics of Consumer Goods. in Biethahn, J., Nissen, V. (eds.): *Evolutionary Algorithms in Management Applications*, Springer-Verlag, pp. 211-224.
- Terano, T., Ishino, Y.(1996). Knowledge Acquisition from Questionnaire Data Using Simulated Breeding and Inductive Learning Methods. *Expert Systems with Applications*, Vol. 11, No. 4, pp. 507-518.
- Unemi, T.(1994). Genetic Algorithms and Computer Graphics Arts (in Japanese). *Journal of Japanese Society for Artificial Intelligence*, Vol. 9, No. 4, pp. 42-47.
- Vafaie, H., De Jong, K. A.(1994). Improving a Rule Induction System Using Genetic Algorithms. in Michalski, R., Tecuci, G. (eds.): *Machine Learning IV: A Multistrategy Approach*. Morgan-Kaufmann, pp. 453-470.
- Weiss, S. M., Kulikowski, C. A.(1991). *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems*. Morgan-Kaufmann.

Index

- A priori probability, 103
- AASM, 22
- AC2, 368
- AQ algorithm, 57
- AQ17-DCI, 53
- AQ17-HCI, 252, 320
- Adaptive classifier, 362
- Affine contractive transformation, 359
- Attribute deletion, 349
- Attribute interaction, 55
- Attribute spectrum, 260
- Automatic interaction detection, 403
- BP, 252
- Base-level variable, 221
- Bayes rule, 34
- Bayesian classifier, 14, 342, 370
 - naive Bayesian classifier, 38, 43, 47, 342, 344
- Bhattacharyya distance, 112
- Bias-variance tradeoff, 39
- Boolean operator, 55, 259
- Boolean variable, 220, 337
- C4.5, 34, 40–41, 91, 145, 207, 213, 243, 246, 267, 291, 315, 333, 368, 379, 395
- CARMA, 27
- CART, 34, 298, 379
- CITRE, 258
- CN2, 257, 368
- CS-IBL, 20
- Cartesian product, 342
 - cartesian product attribute, 348
- Case-base reasoning, 14
- ChiMerge algorithm, 55, 58
- Classification accuracy, 335
- Classification, 15, 184, 307
 - classification accuracy, 334–335
- Collage theorem, 360
- Collocational information, 160
- Collocational lexical analysis, 157
- Complexity measure, 326
- Compound operator, 39–40
- Comprehensibility, 10, 118, 269
- Concept hierarchy, 377
- Concept representation space, 53
- Conjunction, 241
- Conjunctive normal form, 240, 259
- Conjunctive representation, 241
- Construction algorithm, 192
- Constructive induction, 52, 56, 137–138, 240, 273–274, 290, 293, 301–302, 342, 379
 - data-driven, 53, 55, 259, 290
 - hypothesis-driven, 55, 259, 290
 - knowledge-driven, 55, 290
 - multistrategy, 55, 290
- Constructor, 53
- Contextual information, 163
- Correlated feature, 38
- Cost, 142, 145
 - training cost, 14
- Cross entropy error function, 194
- Cross entropy, 45
- Cross validation, 38, 195, 199, 245, 282, 299, 320, 342, 346
- DCFringe, 258
- DIET, 24
- DNC, 223
- DUCE, 337
- Data mining, 5, 10, 71, 118, 132, 269, 394
- Decision tree, 14, 192, 200, 290–291, 293–294, 296–297, 301, 335, 346
 - multivariate decision tree, 244
 - pruning, 297
 - oblique decision tree, 293, 298, 380
 - probabilistic decision tree, 293, 301
 - size, 335
- Destructor, 53
- Diagnosis, 375
- Dimensional analysis, 277
- Dimensionality reduction, 176

- Discourse structure, 160
- Discretization, 44
- Discriminant analysis, 176, 291
 - linear discriminant, 295, 298, 301
 - linear discrimination, 403
- Disjunction, 241
- Disjunctive normal form, 240, 259
- Disjunctive representation, 241
- DistAI, 121, 125
- Divide-and-conquer, 309
- Dynamic operator, 40
- EACH, 18
- ELF, 224
- Eager learning, 4, 14
- Entropy function, 162, 207, 209–210
- Evaluation bias, 26
- Evaluation function, 219
- Evolutionary algorithm, 122
- Evolutionary computation, 308
- Example representation space, 52–53
- Exclusive-or, 345
- Expected loss, 15
- Extracted feature, 193–195, 200, 202
- FOCUS, 36, 119
- FRINGE, 252, 258, 320
- Face recognition, 316
- Feature construction, 4, 132, 273, 308–309, 334
- Feature discovery, 4, 325
- Feature extraction, 4, 103, 177, 192, 195, 197
- Feature hierarchy, 325, 331
- Feature overlap, 222
- Feature selection, 13, 72–73, 90, 103, 308
 - feature subset selection, 34, 118
- Feature space, 221
- Feature transformation, 3, 192, 377
- Feature warp, 226
- Feature weighting, 4, 13, 86, 118
- Filter model, 4, 16, 36, 86, 121, 309
- First-order logic, 163
- Force/torque profile, 381, 391
- Formal concept analysis, 207
- Fourier transform, 381
- Fractal, 360
 - fractal feature extraction, 365
 - fractal feature, 362
- Function approximation, 219
- Function decomposition, 325
- GADistAI, 122
- GALA, 55, 261, 268
- GCM-ISW, 22
- GREEDY3, 252, 258, 320
- GROVE, 320
- Gain ratio, 266
 - relative gain ratio, 266
- Generalization, 329, 333
- Genetic algorithm, 122, 310, 395
 - fitness function, 126, 314
 - genetic operator, 123, 311–312, 399
 - crossover, 123, 311, 399
 - mutation, 123, 311, 399
 - Genetic programming, 132
 - HINT, 326
 - IB3-CI, 26
 - IB4, 18
 - IBL, 34
 - ICET, 145
 - ID2-of-3, 252
 - ID3, 346, 379
 - IDX, 258
 - IGLUE, 205, 207, 213, 215
 - ILLM, 143
 - INDCART, 368
 - ISAC, 23
 - Image classification, 361–362
 - Incompatibility graph, 328
 - Inconsistency rate, 73
 - Incremental learning, 14
 - Inductive logic programming, 138
 - Information retrieval, 158
 - Inheritance hierarchy, 277
 - Instance-based learning, 14, 206
 - Interactive genetic algorithm, 395
 - Interestingness, 160
 - Interleaving vs. preprocessing, 258
 - Intra-collocation measure, 162
 - Intrinsic property, 222
 - Invariant set, 360
 - Irrelevant feature, 35, 207
 - Iterated function system, 358, 360
 - Joint probability, 343, 346
 - Knowledge acquisition, 158
 - Knowledge discovery, 5, 56, 132, 158, 163, 269
 - Knowledge, 273
 - background knowledge, 260
 - contingency knowledge, 276
 - correlation knowledge, 275
 - domain knowledge, 7
 - fragmentary knowledge, 7, 273
 - normalization knowledge, 276
 - proximity knowledge, 276
 - relevance knowledge, 274
 - strong knowledge, 7
 - support knowledge, 275
 - weak knowledge, 7
 - LEGAL, 205, 207, 213, 215
 - LFC, 55, 252
 - LINUS, 138, 140
 - LMDT, 298
 - LOOCE, 19
 - LVF, 120
 - LVW, 120
 - Latent model, 108

- modified Gaussian model, 108
- Lattice, 205, 207–208, 210, 215
 - Galois connection, 208
 - Galois lattice, 207–208
- Lazy learning, 4, 13
- Lazy similarity algorithm, 14
- Learning information, 88
- Literal, 139
- Ltree, 293
- M-of-N, 241, 291
- MARS, 252
- MRP, 258
- Machine learning, 87, 309, 333, 338
- Mahalanobis distance, 207, 212
- Marketing decision making, 394
- Maximum likelihood estimation, 109
 - EM algorithm, 109
- Measure, 259
 - absolute measure, 259
 - dependence measure, 5
 - distance measure, 5
 - error rate, 5
 - inconsistency rate, 5, 72–73
 - information measure, 5
 - relative measure, 259
- MineSet, 45
- Minimum description length, 89, 119
- Minimum encoding length, 89
- Misclassification risk, 15
- Missing value, 298
- Monotonicity condition, 105, 120
- Multi-criteria optimization, 8, 122
- Multimodal, 105
- Multivariate data analysis, 176
- Multivariate variable transformation, 177
- Mutual information, 88, 96, 160, 162
- NODAL, 26
- NP-complete, 90
- NP-hard, 34
- Natural language, 163
- Natural scene interpretation, 63
- Nearest neighbor, 15, 207, 348
 - k-nearest neighbor, 15
- Necessity axiom, 89
- Negative literal, 140
- Neural network, 120, 123, 192–193, 364
 - construction algorithm, 124, 192
 - feature extraction, 195
 - pruning algorithm, 192
- NewID, 368
- Noise, 150
 - noise handling, 150–151
- Non-linear classifier, 120
- Non-linear fitting, 301
- Nonparametric method, 108
- OC1, 298, 379
- Occam’s razor, 88
- Oligoterm attribute, 261
- Optimal feature subset, 35, 95
- Orthogonal feature, 222
- Overfitting, 44
- P/n pair, 141
- PEBLS, 342, 348
- PRESET, 120
- Pairwise t-test, 246
 - one-tailed pairwise t-test, 268
 - two-tailed pairwise t-test, 246, 344
- Partition matrix, 328
- Pattern language, 224
- Polyterm attribute, 261
- Predicate invention, 137–138
- Primitive feature, 240
- Principal components analysis, 24
- Principle of parsimony, 88
- Programming by demonstration, 376
- Prolog, 145
- Prototypical structure, 261
- Pruning algorithm, 192
- Pruning, 75, 297
 - QM2, 24
 - QM2y, 24
- Qualitative state, 382
- Questionnaire data analysis, 399
- RC, 19
- REDUCE, 143
- REDWOOD, 320
- RELIEF, 19, 36, 120, 331
- RL-ICET, 139, 145
- ROBBIE, 27
- RULEARNER, 208
- Redundant feature, 330
- Regression, 176, 183
 - regression tree, 230
- Relevance, 6, 35, 90, 142–143
 - of features, 143
 - of literals, 142
 - relevancy filter, 138
 - strong relevance, 35, 86
 - weak relevance, 35, 86
- Relevancy filter, 143
- Relevant term, 138
- Replication problem, 201
- Robotics, 390
- SCYTHE, 23
- SIBILE, 394
- SKIL, 377
- SMART+, 379
- Sample space, 24
- Scale type, 9
- Search, 17, 76, 106
 - exhaustive search, 119
 - heuristic search, 119
 - randomized search, 120
 - backward elimination, 119, 343

- best-first, 39
- branch and bound, 105
- breadth-first search, 76
- depth-first search, 76
- evaluation criteria, 110
- floating search (SFFS and SBFS), 106
- forward selection, 119
- generate and test, 58
- generate-and-test, 264
- hill-climbing, 18, 342
- hybrid search, 72, 76
- Selective induction, 239
- Selective learning system, 52
- Self similar, 359
- Simulated breeding, 395
- Soft threshold, 296
- Spectral data, 176
- Speech recognition, 113
- Stacked generalization, 302
- Statistical pattern recognition, 103
- Structured concept, 378
- Subset selection, 3, 102, 114, 182, 341
- Sufficiency axiom, 88
- Sufficiency feature set, 88
- TDIDT, 145
- Text categorization, 60
- Textual feature, 161
- Texture discrimination, 112
- Time-dependent feature, 381, 390
- Transformation matrix, 103
- Truth-value, 144
- Unfaithful pattern, 120
- Unimodal, 105
- Unsupervised feature analysis, 163
- VC dimension, 245
- Variable relevance, 96
- Variable-resolution lookup table, 230
- Vertical compactness, 71–72
- Wavelet, 177
 - adaptive wavelet, 177
 - adaptive wavelet matrix, 177
 - discrete wavelet transform, 180
- Weight normalization, 262
- Weight space, 21
- Wrapper model, 4, 16, 38, 86, 121, 309, 342
- X-of-N, 241, 291