DATA CLASSIFICATION

Algorithms and Applications

Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series

SERIES EDITOR
Vipin Kumar
University of Minnesota
Department of Computer Science and Engineering
Minneapolis, Minnesota, U.S.A.

AIMS AND SCOPE

This series aims to capture new developments and applications in data mining and knowledge discovery, while summarizing the computational tools and techniques useful in data analysis. This series encourages the integration of mathematical, statistical, and computational methods and techniques through the publication of a broad range of textbooks, reference works, and handbooks. The inclusion of concrete examples and applications is highly encouraged. The scope of the series includes, but is not limited to, titles in the areas of data mining and knowledge discovery methods and applications, modeling, algorithms, theory and foundations, data and knowledge visualization, data mining systems and tools, and privacy and security issues.

PUBLISHED TITLES

ADVANCES IN MACHINE LEARNING AND DATA MINING FOR ASTRONOMY
Michael J. Way, Jeffrey D. Scargle, Kamal M. Ali, and Ashok N. Srivastava

BIOLOGICAL DATA MINING
Jake Y. Chen and Stefano Lonardi

COMPUTATIONAL BUSINESS ANALYTICS
Subrata Das

COMPUTATIONAL INTELLIGENT DATA ANALYSIS FOR SUSTAINABLE DEVELOPMENT
Ting Yu, Nitesh V. Chawla, and Simeon Simoff

COMPUTATIONAL METHODS OF FEATURE SELECTION
Huan Liu and Hiroshi Motoda

CONSTRAINED CLUSTERING: ADVANCES IN ALGORITHMS, THEORY, AND APPLICATIONS
Sugato Basu, Ian Davidson, and Kiri L. Wagstaff

CONTRAST DATA MINING: CONCEPTS, ALGORITHMS, AND APPLICATIONS
Guozhu Dong and James Bailey

DATA CLASSIFICATION: ALGORITHMS AND APPLICATIONS
Charu C. Aggarawal

DATA CLUSTERING: ALGORITHMS AND APPLICATIONS
Charu C. Aggarawal and Chandan K. Reddy

DATA CLUSTERING IN C++: AN OBJECT-ORIENTED APPROACH
Guojun Gan

DATA MINING FOR DESIGN AND MARKETING
Yukio Ohsawa and Katsutoshi Yada

DATA MINING WITH R: LEARNING WITH CASE STUDIES
Luís Torgo

FOUNDATIONS OF PREDICTIVE ANALYTICS
James Wu and Stephen Coggeshall

GEOGRAPHIC DATA MINING AND KNOWLEDGE DISCOVERY, SECOND EDITION
Harvey J. Miller and Jiawei Han

HANDBOOK OF EDUCATIONAL DATA MINING
Cristóbal Romero, Sebastian Ventura, Mykola Pechenizkiy, and Ryan S.J.d. Baker

INFORMATION DISCOVERY ON ELECTRONIC HEALTH RECORDS
Vagelis Hristidis

INTELLIGENT TECHNOLOGIES FOR WEB APPLICATIONS
Priti Srinivas Sajja and Rajendra Akerkar

INTRODUCTION TO PRIVACY-PRESERVING DATA PUBLISHING: CONCEPTS AND TECHNIQUES
Benjamin C. M. Fung, Ke Wang, Ada Wai-Chee Fu, and Philip S. Yu

KNOWLEDGE DISCOVERY FOR COUNTERTERRORISM AND LAW ENFORCEMENT
David Skillicorn

KNOWLEDGE DISCOVERY FROM DATA STREAMS
João Gama

MACHINE LEARNING AND KNOWLEDGE DISCOVERY FOR ENGINEERING SYSTEMS HEALTH MANAGEMENT
Ashok N. Srivastava and Jiawei Han

MINING SOFTWARE SPECIFICATIONS: METHODOLOGIES AND APPLICATIONS
David Lo, Siau-Cheng Khoo, Jiawei Han, and Chao Liu

MULTIMEDIA DATA MINING: A SYSTEMATIC INTRODUCTION TO CONCEPTS AND THEORY
Zhongfei Zhang and Ruofei Zhang

MUSIC DATA MINING
Tao Li, Mitsunori Ogihara, and George Tzanetakis

NEXT GENERATION OF DATA MINING
Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeev Motwani, and Vipin Kumar

RAPIDMINER: DATA MINING USE CASES AND BUSINESS ANALYTICS APPLICATIONS
Markus Hofmann and Ralf Klinkenberg

RELATIONAL DATA CLUSTERING: MODELS, ALGORITHMS, AND APPLICATIONS
Bo Long, Zhongfei Zhang, and Philip S. Yu

SERVICE-ORIENTED DISTRIBUTED KNOWLEDGE DISCOVERY
Domenico Talia and Paolo Trunfio

SPECTRAL FEATURE SELECTION FOR DATA MINING
Zheng Alan Zhao and Huan Liu

STATISTICAL DATA MINING USING SAS APPLICATIONS, SECOND EDITION
George Fernandez

SUPPORT VECTOR MACHINES: OPTIMIZATION BASED THEORY, ALGORITHMS, AND EXTENSIONS
Naiyang Deng, Yingjie Tian, and Chunhua Zhang

TEMPORAL DATA MINING
Theophano Mitsa

TEXT MINING: CLASSIFICATION, CLUSTERING, AND APPLICATIONS
Ashok N. Srivastava and Mehran Sahami

THE TOP TEN ALGORITHMS IN DATA MINING
Xindong Wu and Vipin Kumar

UNDERSTANDING COMPLEX DATASETS: DATA MINING WITH MATRIX DECOMPOSITIONS
David Skillicorn

Data Classification
Algorithms and Applications

Edited by

Charu C. Aggarwal

IBM T. J. Watson Research Center

Yorktown Heights, New York, USA

CRC Press

Taylor & Francis Group

Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20140611

International Standard Book Number-13: 978-1-4665-8674-1 (Hardback)

Library of Congress Cataloging-in-Publication Data

Data classification : algorithms and applications / edited by Charu C. Aggarwal.
pages cm -- (Chapman & Hall/CRC data mining and knowledge discovery series ; 35)
Summary: "This book homes in on three primary aspects of data classification: the core methods for data classification including probabilistic classification, decision trees, rule-based methods, and SVM methods; different problem domains and scenarios such as multimedia data, text data, biological data, categorical data, network data, data streams and uncertain data: and different variations of the classification problem such as ensemble methods, visual methods, transfer learning, semi-supervised methods and active learning. These advanced methods can be used to enhance the quality of the underlying classification results"-- Provided by publisher.
Includes bibliographical references and index.
ISBN 978-1-4665-8674-1 (hardback)
1. File organization (Computer science) 2. Categories (Mathematics) 3. Algorithms. I. Aggarwal, Charu C.

QA76.9.F5.D38 2014
005.74'1--dc23 2013050912

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

To my wife Lata, and my daughter Sayani

Contents

Editor Biography **xxiii**

Contributors **xxv**

Preface **xxvii**

1 An Introduction to Data Classification **1**
Charu C. Aggarwal
1.1 Introduction . . . 2
1.2 Common Techniques in Data Classification . . . 4
1.2.1 Feature Selection Methods . . . 4
1.2.2 Probabilistic Methods . . . 6
1.2.3 Decision Trees . . . 7
1.2.4 Rule-Based Methods . . . 9
1.2.5 Instance-Based Learning . . . 11
1.2.6 SVM Classifiers . . . 11
1.2.7 Neural Networks . . . 14
1.3 Handing Different Data Types . . . 16
1.3.1 Large Scale Data: Big Data and Data Streams . . . 16
1.3.1.1 Data Streams . . . 16
1.3.1.2 The Big Data Framework . . . 17
1.3.2 Text Classification . . . 18
1.3.3 Multimedia Classification . . . 20
1.3.4 Time Series and Sequence Data Classification . . . 20
1.3.5 Network Data Classification . . . 21
1.3.6 Uncertain Data Classification . . . 21
1.4 Variations on Data Classification . . . 22
1.4.1 Rare Class Learning . . . 22
1.4.2 Distance Function Learning . . . 22
1.4.3 Ensemble Learning for Data Classification . . . 23
1.4.4 Enhancing Classification Methods with Additional Data . . . 24
1.4.4.1 Semi-Supervised Learning . . . 24
1.4.4.2 Transfer Learning . . . 26
1.4.5 Incorporating Human Feedback . . . 27
1.4.5.1 Active Learning . . . 28
1.4.5.2 Visual Learning . . . 29
1.4.6 Evaluating Classification Algorithms . . . 30
1.5 Discussion and Conclusions . . . 31

2 Feature Selection for Classification: A Review **37**
Jiliang Tang, Salem Alelyani, and Huan Liu
2.1 Introduction . . . 38
2.1.1 Data Classification . . . 39
2.1.2 Feature Selection . . . 40
2.1.3 Feature Selection for Classification . . . 42
2.2 Algorithms for Flat Features . . . 43
2.2.1 Filter Models . . . 44
2.2.2 Wrapper Models . . . 46
2.2.3 Embedded Models . . . 47
2.3 Algorithms for Structured Features . . . 49
2.3.1 Features with Group Structure . . . 50
2.3.2 Features with Tree Structure . . . 51
2.3.3 Features with Graph Structure . . . 53
2.4 Algorithms for Streaming Features . . . 55
2.4.1 The Grafting Algorithm . . . 56
2.4.2 The Alpha-Investing Algorithm . . . 56
2.4.3 The Online Streaming Feature Selection Algorithm . . . 57
2.5 Discussions and Challenges . . . 57
2.5.1 Scalability . . . 57
2.5.2 Stability . . . 58
2.5.3 Linked Data . . . 58

3 Probabilistic Models for Classification **65**
Hongbo Deng, Yizhou Sun, Yi Chang, and Jiawei Han
3.1 Introduction . . . 66
3.2 Naive Bayes Classification . . . 67
3.2.1 Bayes' Theorem and Preliminary . . . 67
3.2.2 Naive Bayes Classifier . . . 69
3.2.3 Maximum-Likelihood Estimates for Naive Bayes Models . . . 70
3.2.4 Applications . . . 71
3.3 Logistic Regression Classification . . . 72
3.3.1 Logistic Regression . . . 73
3.3.2 Parameters Estimation for Logistic Regression . . . 74
3.3.3 Regularization in Logistic Regression . . . 75
3.3.4 Applications . . . 76
3.4 Probabilistic Graphical Models for Classification . . . 76
3.4.1 Bayesian Networks . . . 76
3.4.1.1 Bayesian Network Construction . . . 77
3.4.1.2 Inference in a Bayesian Network . . . 78
3.4.1.3 Learning Bayesian Networks . . . 78
3.4.2 Hidden Markov Models . . . 78
3.4.2.1 The Inference and Learning Algorithms . . . 79
3.4.3 Markov Random Fields . . . 81
3.4.3.1 Conditional Independence . . . 81
3.4.3.2 Clique Factorization . . . 81
3.4.3.3 The Inference and Learning Algorithms . . . 82
3.4.4 Conditional Random Fields . . . 82
3.4.4.1 The Learning Algorithms . . . 83
3.5 Summary . . . 83

4 Decision Trees: Theory and Algorithms **87**
Victor E. Lee, Lin Liu, and Ruoming Jin
4.1 Introduction . . . 87
4.2 Top-Down Decision Tree Induction . . . 91
4.2.1 Node Splitting . . . 92
4.2.2 Tree Pruning . . . 97
4.3 Case Studies with C4.5 and CART . . . 99
4.3.1 Splitting Criteria . . . 100
4.3.2 Stopping Conditions . . . 100
4.3.3 Pruning Strategy . . . 101
4.3.4 Handling Unknown Values: Induction and Prediction . . . 101
4.3.5 Other Issues: Windowing and Multivariate Criteria . . . 102
4.4 Scalable Decision Tree Construction . . . 103
4.4.1 RainForest-Based Approach . . . 104
4.4.2 SPIES Approach . . . 105
4.4.3 Parallel Decision Tree Construction . . . 107
4.5 Incremental Decision Tree Induction . . . 108
4.5.1 ID3 Family . . . 108
4.5.2 VFDT Family . . . 110
4.5.3 Ensemble Method for Streaming Data . . . 113
4.6 Summary . . . 114

5 Rule-Based Classification **121**
Xiao-Li Li and Bing Liu
5.1 Introduction . . . 121
5.2 Rule Induction . . . 123
5.2.1 Two Algorithms for Rule Induction . . . 123
5.2.1.1 CN2 Induction Algorithm (Ordered Rules) . . . 124
5.2.1.2 RIPPER Algorithm and Its Variations (Ordered Classes) . . . 125
5.2.2 Learn One Rule in Rule Learning . . . 126
5.3 Classification Based on Association Rule Mining . . . 129
5.3.1 Association Rule Mining . . . 130
5.3.1.1 Definitions of Association Rules, Support, and Confidence . . . 131
5.3.1.2 The Introduction of Apriori Algorithm . . . 133
5.3.2 Mining Class Association Rules . . . 136
5.3.3 Classification Based on Associations . . . 139
5.3.3.1 Additional Discussion for CARs Mining . . . 139
5.3.3.2 Building a Classifier Using CARs . . . 140
5.3.4 Other Techniques for Association Rule-Based Classification . . . 142
5.4 Applications . . . 144
5.4.1 Text Categorization . . . 144
5.4.2 Intrusion Detection . . . 147
5.4.3 Using Class Association Rules for Diagnostic Data Mining . . . 148
5.4.4 Gene Expression Data Analysis . . . 149
5.5 Discussion and Conclusion . . . 150

6 Instance-Based Learning: A Survey **157**
Charu C. Aggarwal
6.1 Introduction . . . 157
6.2 Instance-Based Learning Framework . . . 159
6.3 The Nearest Neighbor Classifier . . . 160
6.3.1 Handling Symbolic Attributes . . . 163
6.3.2 Distance-Weighted Nearest Neighbor Methods . . . 163
6.3.3 Local Distance Scaling . . . 164
6.3.4 Attribute-Weighted Nearest Neighbor Methods . . . 164
6.3.5 Locally Adaptive Nearest Neighbor Classifier . . . 167
6.3.6 Combining with Ensemble Methods . . . 169
6.3.7 Multi-Label Learning . . . 169
6.4 Lazy SVM Classification . . . 171
6.5 Locally Weighted Regression . . . 172
6.6 Lazy Naive Bayes . . . 173
6.7 Lazy Decision Trees . . . 173
6.8 Rule-Based Classification . . . 174
6.9 Radial Basis Function Networks: Leveraging Neural Networks for Instance-Based Learning . . . 175
6.10 Lazy Methods for Diagnostic and Visual Classification . . . 176
6.11 Conclusions and Summary . . . 180

7 Support Vector Machines **187**
Po-Wei Wang and Chih-Jen Lin
7.1 Introduction . . . 187
7.2 The Maximum Margin Perspective . . . 188
7.3 The Regularization Perspective . . . 190
7.4 The Support Vector Perspective . . . 191
7.5 Kernel Tricks . . . 194
7.6 Solvers and Algorithms . . . 196
7.7 Multiclass Strategies . . . 198
7.8 Conclusion . . . 201

8 Neural Networks: A Review **205**
Alain Biem
8.1 Introduction . . . 206
8.2 Fundamental Concepts . . . 208
8.2.1 Mathematical Model of a Neuron . . . 208
8.2.2 Types of Units . . . 209
8.2.2.1 McCullough Pitts Binary Threshold Unit . . . 209
8.2.2.2 Linear Unit . . . 210
8.2.2.3 Linear Threshold Unit . . . 211
8.2.2.4 Sigmoidal Unit . . . 211
8.2.2.5 Distance Unit . . . 211
8.2.2.6 Radial Basis Unit . . . 211
8.2.2.7 Polynomial Unit . . . 212
8.2.3 Network Topology . . . 212
8.2.3.1 Layered Network . . . 212
8.2.3.2 Networks with Feedback . . . 212
8.2.3.3 Modular Networks . . . 213
8.2.4 Computation and Knowledge Representation . . . 213

8.2.5 Learning . . . 213
8.2.5.1 Hebbian Rule . . . 213
8.2.5.2 The Delta Rule . . . 214
8.3 Single-Layer Neural Network . . . 214
8.3.1 The Single-Layer Perceptron . . . 214
8.3.1.1 Perceptron Criterion . . . 214
8.3.1.2 Multi-Class Perceptrons . . . 216
8.3.1.3 Perceptron Enhancements . . . 216
8.3.2 Adaline . . . 217
8.3.2.1 Two-Class Adaline . . . 217
8.3.2.2 Multi-Class Adaline . . . 218
8.3.3 Learning Vector Quantization (LVQ) . . . 219
8.3.3.1 LVQ1 Training . . . 219
8.3.3.2 LVQ2 Training . . . 219
8.3.3.3 Application and Limitations . . . 220
8.4 Kernel Neural Network . . . 220
8.4.1 Radial Basis Function Network . . . 220
8.4.2 RBFN Training . . . 222
8.4.2.1 Using Training Samples as Centers . . . 222
8.4.2.2 Random Selection of Centers . . . 222
8.4.2.3 Unsupervised Selection of Centers . . . 222
8.4.2.4 Supervised Estimation of Centers . . . 223
8.4.2.5 Linear Optimization of Weights . . . 223
8.4.2.6 Gradient Descent and Enhancements . . . 223
8.4.3 RBF Applications . . . 223
8.5 Multi-Layer Feedforward Network . . . 224
8.5.1 MLP Architecture for Classification . . . 224
8.5.1.1 Two-Class Problems . . . 225
8.5.1.2 Multi-Class Problems . . . 225
8.5.1.3 Forward Propagation . . . 226
8.5.2 Error Metrics . . . 227
8.5.2.1 Mean Square Error (MSE) . . . 227
8.5.2.2 Cross-Entropy (CE) . . . 227
8.5.2.3 Minimum Classification Error (MCE) . . . 228
8.5.3 Learning by Backpropagation . . . 228
8.5.4 Enhancing Backpropagation . . . 229
8.5.4.1 Backpropagation with Momentum . . . 230
8.5.4.2 Delta-Bar-Delta . . . 231
8.5.4.3 Rprop Algorithm . . . 231
8.5.4.4 Quick-Prop . . . 231
8.5.5 Generalization Issues . . . 232
8.5.6 Model Selection . . . 232
8.6 Deep Neural Networks . . . 232
8.6.1 Use of Prior Knowledge . . . 233
8.6.2 Layer-Wise Greedy Training . . . 234
8.6.2.1 Deep Belief Networks (DBNs) . . . 234
8.6.2.2 Stack Auto-Encoder . . . 235
8.6.3 Limits and Applications . . . 235
8.7 Summary . . . 235

9 A Survey of Stream Classification Algorithms **245**
Charu C. Aggarwal
9.1 Introduction . . . 245
9.2 Generic Stream Classification Algorithms . . . 247
9.2.1 Decision Trees for Data Streams . . . 247
9.2.2 Rule-Based Methods for Data Streams . . . 249
9.2.3 Nearest Neighbor Methods for Data Streams . . . 250
9.2.4 SVM Methods for Data Streams . . . 251
9.2.5 Neural Network Classifiers for Data Streams . . . 252
9.2.6 Ensemble Methods for Data Streams . . . 253
9.3 Rare Class Stream Classification . . . 254
9.3.1 Detecting Rare Classes . . . 255
9.3.2 Detecting Novel Classes . . . 255
9.3.3 Detecting Infrequently Recurring Classes . . . 256
9.4 Discrete Attributes: The Massive Domain Scenario . . . 256
9.5 Other Data Domains . . . 262
9.5.1 Text Streams . . . 262
9.5.2 Graph Streams . . . 264
9.5.3 Uncertain Data Streams . . . 267
9.6 Conclusions and Summary . . . 267

10 Big Data Classification **275**
Hanghang Tong
10.1 Introduction . . . 275
10.2 Scale-Up on a Single Machine . . . 276
10.2.1 Background . . . 276
10.2.2 SVMPerf . . . 276
10.2.3 Pegasos . . . 277
10.2.4 Bundle Methods . . . 279
10.3 Scale-Up by Parallelism . . . 280
10.3.1 Parallel Decision Trees . . . 280
10.3.2 Parallel SVMs . . . 281
10.3.3 MRM-ML . . . 281
10.3.4 SystemML . . . 282
10.4 Conclusion . . . 283

11 Text Classification **287**
Charu C. Aggarwal and ChengXiang Zhai
11.1 Introduction . . . 288
11.2 Feature Selection for Text Classification . . . 290
11.2.1 Gini Index . . . 291
11.2.2 Information Gain . . . 292
11.2.3 Mutual Information . . . 292
11.2.4 χ^2-Statistic . . . 292
11.2.5 Feature Transformation Methods: Unsupervised and Supervised LSI . . . 293
11.2.6 Supervised Clustering for Dimensionality Reduction . . . 294
11.2.7 Linear Discriminant Analysis . . . 294
11.2.8 Generalized Singular Value Decomposition . . . 295
11.2.9 Interaction of Feature Selection with Classification . . . 296
11.3 Decision Tree Classifiers . . . 296
11.4 Rule-Based Classifiers . . . 298

11.5 Probabilistic and Naive Bayes Classifiers . . . 300
11.5.1 Bernoulli Multivariate Model . . . 301
11.5.2 Multinomial Distribution . . . 304
11.5.3 Mixture Modeling for Text Classification . . . 305
11.6 Linear Classifiers . . . 308
11.6.1 SVM Classifiers . . . 308
11.6.2 Regression-Based Classifiers . . . 311
11.6.3 Neural Network Classifiers . . . 312
11.6.4 Some Observations about Linear Classifiers . . . 315
11.7 Proximity-Based Classifiers . . . 315
11.8 Classification of Linked and Web Data . . . 317
11.9 Meta-Algorithms for Text Classification . . . 321
11.9.1 Classifier Ensemble Learning . . . 321
11.9.2 Data Centered Methods: Boosting and Bagging . . . 322
11.9.3 Optimizing Specific Measures of Accuracy . . . 322
11.10 Leveraging Additional Training Data . . . 323
11.10.1 Semi-Supervised Learning . . . 324
11.10.2 Transfer Learning . . . 326
11.10.3 Active Learning . . . 327
11.11 Conclusions and Summary . . . 327

12 Multimedia Classification **337**
Shiyu Chang, Wei Han, Xianming Liu, Ning Xu, Pooya Khorrami, and Thomas S. Huang
12.1 Introduction . . . 338
12.1.1 Overview . . . 338
12.2 Feature Extraction and Data Pre-Processing . . . 339
12.2.1 Text Features . . . 340
12.2.2 Image Features . . . 341
12.2.3 Audio Features . . . 344
12.2.4 Video Features . . . 345
12.3 Audio Visual Fusion . . . 345
12.3.1 Fusion Methods . . . 346
12.3.2 Audio Visual Speech Recognition . . . 346
12.3.2.1 Visual Front End . . . 347
12.3.2.2 Decision Fusion on HMM . . . 348
12.3.3 Other Applications . . . 349
12.4 Ontology-Based Classification and Inference . . . 349
12.4.1 Popular Applied Ontology . . . 350
12.4.2 Ontological Relations . . . 350
12.4.2.1 Definition . . . 351
12.4.2.2 Subclass Relation . . . 351
12.4.2.3 Co-Occurrence Relation . . . 352
12.4.2.4 Combination of the Two Relations . . . 352
12.4.2.5 Inherently Used Ontology . . . 353
12.5 Geographical Classification with Multimedia Data . . . 353
12.5.1 Data Modalities . . . 353
12.5.2 Challenges in Geographical Classification . . . 354
12.5.3 Geo-Classification for Images . . . 355
12.5.3.1 Classifiers . . . 356
12.5.4 Geo-Classification for Web Videos . . . 356

12.6 Conclusion . . . 356

13 Time Series Data Classification **365**
Dimitrios Kotsakos and Dimitrios Gunopulos
13.1 Introduction . . . 365
13.2 Time Series Representation . . . 367
13.3 Distance Measures . . . 367
13.3.1 L_p-Norms . . . 367
13.3.2 Dynamic Time Warping (DTW) . . . 367
13.3.3 Edit Distance . . . 368
13.3.4 Longest Common Subsequence (LCSS) . . . 369
13.4 k-NN . . . 369
13.4.1 Speeding up the k-NN . . . 370
13.5 Support Vector Machines (SVMs) . . . 371
13.6 Classification Trees . . . 372
13.7 Model-Based Classification . . . 374
13.8 Distributed Time Series Classification . . . 375
13.9 Conclusion . . . 375

14 Discrete Sequence Classification **379**
Mohammad Al Hasan
14.1 Introduction . . . 379
14.2 Background . . . 380
14.2.1 Sequence . . . 380
14.2.2 Sequence Classification . . . 381
14.2.3 Frequent Sequential Patterns . . . 381
14.2.4 n-Grams . . . 382
14.3 Sequence Classification Methods . . . 382
14.4 Feature-Based Classification . . . 382
14.4.1 Filtering Method for Sequential Feature Selection . . . 383
14.4.2 Pattern Mining Framework for Mining Sequential Features . . . 385
14.4.3 A Wrapper-Based Method for Mining Sequential Features . . . 386
14.5 Distance-Based Methods . . . 386
14.5.0.1 Alignment-Based Distance . . . 387
14.5.0.2 Keyword-Based Distance . . . 388
14.5.0.3 Kernel-Based Similarity . . . 388
14.5.0.4 Model-Based Similarity . . . 388
14.5.0.5 Time Series Distance Metrics . . . 388
14.6 Model-Based Method . . . 389
14.7 Hybrid Methods . . . 390
14.8 Non-Traditional Sequence Classification . . . 391
14.8.1 Semi-Supervised Sequence Classification . . . 391
14.8.2 Classification of Label Sequences . . . 392
14.8.3 Classification of Sequence of Vector Data . . . 392
14.9 Conclusions . . . 393

15 Collective Classification of Network Data **399**
Ben London and Lise Getoor
15.1 Introduction . . . 399
15.2 Collective Classification Problem Definition . . . 400
15.2.1 Inductive vs. Transductive Learning . . . 401

15.2.2 Active Collective Classification . . . 402
15.3 Iterative Methods . . . 402
15.3.1 Label Propagation . . . 402
15.3.2 Iterative Classification Algorithms . . . 404
15.4 Graph-Based Regularization . . . 405
15.5 Probabilistic Graphical Models . . . 406
15.5.1 Directed Models . . . 406
15.5.2 Undirected Models . . . 408
15.5.3 Approximate Inference in Graphical Models . . . 409
15.5.3.1 Gibbs Sampling . . . 409
15.5.3.2 Loopy Belief Propagation (LBP) . . . 410
15.6 Feature Construction . . . 410
15.6.1 Data Graph . . . 411
15.6.2 Relational Features . . . 412
15.7 Applications of Collective Classification . . . 412
15.8 Conclusion . . . 413

16 Uncertain Data Classification **417**
Reynold Cheng, Yixiang Fang, and Matthias Renz
16.1 Introduction . . . 417
16.2 Preliminaries . . . 419
16.2.1 Data Uncertainty Models . . . 419
16.2.2 Classification Framework . . . 419
16.3 Classification Algorithms . . . 420
16.3.1 Decision Trees . . . 420
16.3.2 Rule-Based Classification . . . 424
16.3.3 Associative Classification . . . 426
16.3.4 Density-Based Classification . . . 429
16.3.5 Nearest Neighbor-Based Classification . . . 432
16.3.6 Support Vector Classification . . . 436
16.3.7 Naive Bayes Classification . . . 438
16.4 Conclusions . . . 441

17 Rare Class Learning **445**
Charu C. Aggarwal
17.1 Introduction . . . 445
17.2 Rare Class Detection . . . 448
17.2.1 Cost Sensitive Learning . . . 449
17.2.1.1 MetaCost: A Relabeling Approach . . . 449
17.2.1.2 Weighting Methods . . . 450
17.2.1.3 Bayes Classifiers . . . 450
17.2.1.4 Proximity-Based Classifiers . . . 451
17.2.1.5 Rule-Based Classifiers . . . 451
17.2.1.6 Decision Trees . . . 451
17.2.1.7 SVM Classifier . . . 452
17.2.2 Adaptive Re-Sampling . . . 452
17.2.2.1 Relation between Weighting and Sampling . . . 453
17.2.2.2 Synthetic Over-Sampling: SMOTE . . . 453
17.2.2.3 One Class Learning with Positive Class . . . 453
17.2.2.4 Ensemble Techniques . . . 454
17.2.3 Boosting Methods . . . 454

17.3 The Semi-Supervised Scenario: Positive and Unlabeled Data 455
17.3.1 Difficult Cases and One-Class Learning 456
17.4 The Semi-Supervised Scenario: Novel Class Detection 456
17.4.1 One Class Novelty Detection 457
17.4.2 Combining Novel Class Detection with Rare Class Detection 458
17.4.3 Online Novelty Detection 458
17.5 Human Supervision 459
17.6 Other Work 461
17.7 Conclusions and Summary 462

18 Distance Metric Learning for Data Classification 469
Fei Wang
18.1 Introduction 469
18.2 The Definition of Distance Metric Learning 470
18.3 Supervised Distance Metric Learning Algorithms 471
18.3.1 Linear Discriminant Analysis (LDA) 472
18.3.2 Margin Maximizing Discriminant Analysis (MMDA) 473
18.3.3 Learning with Side Information (LSI) 474
18.3.4 Relevant Component Analysis (RCA) 474
18.3.5 Information Theoretic Metric Learning (ITML) 475
18.3.6 Neighborhood Component Analysis (NCA) 475
18.3.7 Average Neighborhood Margin Maximization (ANMM) 476
18.3.8 Large Margin Nearest Neighbor Classifier (LMNN) 476
18.4 Advanced Topics 477
18.4.1 Semi-Supervised Metric Learning 477
18.4.1.1 Laplacian Regularized Metric Learning (LRML) 477
18.4.1.2 Constraint Margin Maximization (CMM) 478
18.4.2 Online Learning 478
18.4.2.1 Pseudo-Metric Online Learning Algorithm (POLA) 479
18.4.2.2 Online Information Theoretic Metric Learning (OITML) . . . 480
18.5 Conclusions and Discussions 480

19 Ensemble Learning 483
Yaliang Li, Jing Gao, Qi Li, and Wei Fan
19.1 Introduction 484
19.2 Bayesian Methods 487
19.2.1 Bayes Optimal Classifier 487
19.2.2 Bayesian Model Averaging 488
19.2.3 Bayesian Model Combination 490
19.3 Bagging 491
19.3.1 General Idea 491
19.3.2 Random Forest 493
19.4 Boosting 495
19.4.1 General Boosting Procedure 495
19.4.2 AdaBoost 496
19.5 Stacking 498
19.5.1 General Stacking Procedure 498
19.5.2 Stacking and Cross-Validation 500
19.5.3 Discussions 501
19.6 Recent Advances in Ensemble Learning 502
19.7 Conclusions 503

20 Semi-Supervised Learning **511**

Kaushik Sinha

20.1 Introduction . . . 511
20.1.1 Transductive vs. Inductive Semi-Supervised Learning . . . 514
20.1.2 Semi-Supervised Learning Framework and Assumptions . . . 514
20.2 Generative Models . . . 515
20.2.1 Algorithms . . . 516
20.2.2 Description of a Representative Algorithm . . . 516
20.2.3 Theoretical Justification and Relevant Results . . . 517
20.3 Co-Training . . . 519
20.3.1 Algorithms . . . 520
20.3.2 Description of a Representative Algorithm . . . 520
20.3.3 Theoretical Justification and Relevant Results . . . 520
20.4 Graph-Based Methods . . . 522
20.4.1 Algorithms . . . 522
20.4.1.1 Graph Cut . . . 522
20.4.1.2 Graph Transduction . . . 523
20.4.1.3 Manifold Regularization . . . 524
20.4.1.4 Random Walk . . . 525
20.4.1.5 Large Scale Learning . . . 526
20.4.2 Description of a Representative Algorithm . . . 526
20.4.3 Theoretical Justification and Relevant Results . . . 527
20.5 Semi-Supervised Learning Methods Based on Cluster Assumption . . . 528
20.5.1 Algorithms . . . 528
20.5.2 Description of a Representative Algorithm . . . 529
20.5.3 Theoretical Justification and Relevant Results . . . 529
20.6 Related Areas . . . 531
20.7 Concluding Remarks . . . 531

21 Transfer Learning **537**

Sinno Jialin Pan

21.1 Introduction . . . 538
21.2 Transfer Learning Overview . . . 541
21.2.1 Background . . . 541
21.2.2 Notations and Definitions . . . 541
21.3 Homogenous Transfer Learning . . . 542
21.3.1 Instance-Based Approach . . . 542
21.3.1.1 Case I: No Target Labeled Data . . . 543
21.3.1.2 Case II: A Few Target Labeled Data . . . 544
21.3.2 Feature-Representation-Based Approach . . . 545
21.3.2.1 Encoding Specific Knowledge for Feature Learning . . . 545
21.3.2.2 Learning Features by Minimizing Distance between Distributions . . . 548
21.3.2.3 Learning Features Inspired by Multi-Task Learning . . . 549
21.3.2.4 Learning Features Inspired by Self-Taught Learning . . . 550
21.3.2.5 Other Feature Learning Approaches . . . 550
21.3.3 Model-Parameter-Based Approach . . . 550
21.3.4 Relational-Information-Based Approaches . . . 552
21.4 Heterogeneous Transfer Learning . . . 553
21.4.1 Heterogeneous Feature Spaces . . . 553
21.4.2 Different Label Spaces . . . 554

21.5 Transfer Bounds and Negative Transfer . . . 554
21.6 Other Research Issues . . . 555
21.6.1 Binary Classification vs. Multi-Class Classification . . . 556
21.6.2 Knowledge Transfer from Multiple Source Domains . . . 556
21.6.3 Transfer Learning Meets Active Learning . . . 556
21.7 Applications of Transfer Learning . . . 557
21.7.1 NLP Applications . . . 557
21.7.2 Web-Based Applications . . . 557
21.7.3 Sensor-Based Applications . . . 557
21.7.4 Applications to Computer Vision . . . 557
21.7.5 Applications to Bioinformatics . . . 557
21.7.6 Other Applications . . . 558
21.8 Concluding Remarks . . . 558

22 Active Learning: A Survey **571**

Charu C. Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and Philip S. Yu

22.1 Introduction . . . 572
22.2 Motivation and Comparisons to Other Strategies . . . 574
22.2.1 Comparison with Other Forms of Human Feedback . . . 575
22.2.2 Comparisons with Semi-Supervised and Transfer Learning . . . 576
22.3 Querying Strategies . . . 576
22.3.1 Heterogeneity-Based Models . . . 577
22.3.1.1 Uncertainty Sampling . . . 577
22.3.1.2 Query-by-Committee . . . 578
22.3.1.3 Expected Model Change . . . 578
22.3.2 Performance-Based Models . . . 579
22.3.2.1 Expected Error Reduction . . . 579
22.3.2.2 Expected Variance Reduction . . . 580
22.3.3 Representativeness-Based Models . . . 580
22.3.4 Hybrid Models . . . 580
22.4 Active Learning with Theoretical Guarantees . . . 581
22.4.1 A Simple Example . . . 581
22.4.2 Existing Works . . . 582
22.4.3 Preliminaries . . . 582
22.4.4 Importance Weighted Active Learning . . . 582
22.4.4.1 Algorithm . . . 583
22.4.4.2 Consistency . . . 583
22.4.4.3 Label Complexity . . . 584
22.5 Dependency-Oriented Data Types for Active Learning . . . 585
22.5.1 Active Learning in Sequences . . . 585
22.5.2 Active Learning in Graphs . . . 585
22.5.2.1 Classification of Many Small Graphs . . . 586
22.5.2.2 Node Classification in a Single Large Graph . . . 587
22.6 Advanced Methods . . . 589
22.6.1 Active Learning of Features . . . 589
22.6.2 Active Learning of Kernels . . . 590
22.6.3 Active Learning of Classes . . . 591
22.6.4 Streaming Active Learning . . . 591
22.6.5 Multi-Instance Active Learning . . . 592
22.6.6 Multi-Label Active Learning . . . 593
22.6.7 Multi-Task Active Learning . . . 593

22.6.8 Multi-View Active Learning . . . 594
22.6.9 Multi-Oracle Active Learning . . . 594
22.6.10 Multi-Objective Active Learning . . . 595
22.6.11 Variable Labeling Costs . . . 596
22.6.12 Active Transfer Learning . . . 596
22.6.13 Active Reinforcement Learning . . . 597
22.7 Conclusions . . . 597

23 Visual Classification **607**
Giorgio Maria Di Nunzio
23.1 Introduction . . . 608
23.1.1 Requirements for Visual Classification . . . 609
23.1.2 Visualization Metaphors . . . 610
23.1.2.1 2D and 3D Spaces . . . 610
23.1.2.2 More Complex Metaphors . . . 610
23.1.3 Challenges in Visual Classification . . . 611
23.1.4 Related Works . . . 611
23.2 Approaches . . . 612
23.2.1 Nomograms . . . 612
23.2.1.1 Naïve Bayes Nomogram . . . 613
23.2.2 Parallel Coordinates . . . 613
23.2.2.1 Edge Cluttering . . . 614
23.2.3 Radial Visualizations . . . 614
23.2.3.1 Star Coordinates . . . 615
23.2.4 Scatter Plots . . . 616
23.2.4.1 Clustering . . . 617
23.2.4.2 Naïve Bayes Classification . . . 617
23.2.5 Topological Maps . . . 619
23.2.5.1 Self-Organizing Maps . . . 619
23.2.5.2 Generative Topographic Mapping . . . 619
23.2.6 Trees . . . 620
23.2.6.1 Decision Trees . . . 621
23.2.6.2 Treemap . . . 622
23.2.6.3 Hyperbolic Tree . . . 623
23.2.6.4 Phylogenetic Trees . . . 623
23.3 Systems . . . 623
23.3.1 EnsembleMatrix and ManiMatrix . . . 623
23.3.2 Systematic Mapping . . . 624
23.3.3 iVisClassifier . . . 624
23.3.4 ParallelTopics . . . 625
23.3.5 VisBricks . . . 625
23.3.6 WHIDE . . . 625
23.3.7 Text Document Retrieval . . . 625
23.4 Summary and Conclusions . . . 626

24 Evaluation of Classification Methods **633**
Nele Verbiest, Karel Vermeulen, and Ankur Teredesai
24.1 Introduction . . . 633
24.2 Validation Schemes . . . 634
24.3 Evaluation Measures . . . 636
24.3.1 Accuracy Related Measures . . . 636

24.3.1.1 Discrete Classifiers 636
24.3.1.2 Probabilistic Classifiers 638
24.3.2 Additional Measures 642
24.4 Comparing Classifiers 643
24.4.1 Parametric Statistical Comparisons 644
24.4.1.1 Pairwise Comparisons 644
24.4.1.2 Multiple Comparisons 644
24.4.2 Non-Parametric Statistical Comparisons 646
24.4.2.1 Pairwise Comparisons 646
24.4.2.2 Multiple Comparisons 647
24.4.2.3 Permutation Tests 651
24.5 Concluding Remarks 652

25 Educational and Software Resources for Data Classification **657**
Charu C. Aggarwal
25.1 Introduction 657
25.2 Educational Resources 658
25.2.1 Books on Data Classification 658
25.2.2 Popular Survey Papers on Data Classification 658
25.3 Software for Data Classification 659
25.3.1 Data Benchmarks for Software and Research 660
25.4 Summary 661

Index **667**

Editor Biography

Charu C. Aggarwal is a Research Scientist at the IBM T. J. Watson Research Center in Yorktown Heights, New York. He completed his B.S. from IIT Kanpur in 1993 and his Ph.D. from Massachusetts Institute of Technology in 1996. His research interest during his Ph.D. years was in combinatorial optimization (network flow algorithms), and his thesis advisor was Professor James B. Orlin. He has since worked in the field of performance analysis, databases, and data mining. He has published over 200 papers in refereed conferences and journals, and has applied for or been granted over 80 patents. He is author or editor of ten books. Because of the commercial value of the aforementioned patents, he has received several invention achievement awards and has thrice been designated a *Master Inventor* at IBM. He is a recipient of an *IBM Corporate Award* (2003) for his work on bio-terrorist threat detection in data streams, a recipient of the *IBM Outstanding Innovation Award* (2008) for his scientific contributions to privacy technology, a recipient of the *IBM Outstanding Technical Achievement Award* (2009) for his work on data streams, and a recipient of an *IBM Research Division Award* (2008) for his contributions to System S. He also received the *EDBT 2014 Test of Time Award* for his work on condensation-based privacy-preserving data mining.

He served as an associate editor of the *IEEE Transactions on Knowledge and Data Engineering* from 2004 to 2008. He is an associate editor of the *ACM Transactions on Knowledge Discovery and Data Mining*, an action editor of the *Data Mining and Knowledge Discovery Journal*, editor-in-chief of the *ACM SIGKDD Explorations*, and an associate editor of the *Knowledge and Information Systems Journal*. He serves on the advisory board of the *Lecture Notes on Social Networks*, a publication by Springer. He serves as the vice-president of the *SIAM Activity Group on Data Mining*, which is responsible for all data mining activities organized by SIAM, including their main data mining conference. He is a fellow of the IEEE and the ACM, for "*contributions to knowledge discovery and data mining algorithms*."

Contributors

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, New York

Salem Alelyani
Arizona State University
Tempe, Arizona

Mohammad Al Hasan
Indiana University - Purdue University
Indianapolis, Indiana

Alain Biem
IBM T. J. Watson Research Center
Yorktown Heights, New York

Shiyu Chang
University of Illinois at Urbana-Champaign
Urbana, Illinois

Yi Chang
Yahoo! Labs
Sunnyvale, California

Reynold Cheng
The University of Hong Kong
Hong Kong

Hongbo Deng
Yahoo! Research
Sunnyvale, California

Giorgio Maria Di Nunzio
University of Padua
Padova, Italy

Wei Fan
Huawei Noah's Ark Lab
Hong Kong

Yixiang Fang
The University of Hong Kong
Hong Kong

Jing Gao
State University of New York at Buffalo
Buffalo, New York

Quanquan Gu
University of Illinois at Urbana-Champaign
Urbana, Illinois

Dimitrios Gunopulos
University of Athens
Athens, Greece

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

Wei Han
University of Illinois at Urbana-Champaign
Urbana, Illinois

Thomas S. Huang
University of Illinois at Urbana-Champaign
Urbana, Illinois

Ruoming Jin
Kent State University
Kent, Ohio

Xiangnan Kong
University of Illinois at Chicago
Chicago, Illinois

Dimitrios Kotsakos
University of Athens
Athens, Greece

Victor E. Lee
John Carroll University
University Heights, Ohio

Qi Li
State University of New York at Buffalo
Buffalo, New York

Xiao-Li Li
Institute for Infocomm Research
Singapore

Yaliang Li
State University of New York at Buffalo
Buffalo, New York

Bing Liu
University of Illinois at Chicago
Chicago, Illinois

Huan Liu
Arizona State University
Tempe, Arizona

Lin Liu
Kent State University
Kent, Ohio

Xianming Liu
University of Illinois at Urbana-Champaign
Urbana, Illinois

Ben London
University of Maryland
College Park, Maryland

Sinno Jialin Pan
Institute for Infocomm Research
Sinpapore

Pooya Khorrami
University of Illinois at Urbana-Champaign
Urbana, Illinois

Chih-Jen Lin
National Taiwan University
Taipei, Taiwan

Matthias Renz
University of Munich
Munich, Germany

Kaushik Sinha
Wichita State University
Wichita, Kansas

Yizhou Sun
Northeastern University
Boston, Massachusetts

Jiliang Tang
Arizona State University
Tempe, Arizona

Ankur Teredesai
University of Washington
Tacoma, Washington

Hanghang Tong
City University of New York
New York, New York

Nele Verbiest
Ghent University
Belgium

Karel Vermeulen
Ghent University
Belgium

Fei Wang
IBM T. J. Watson Research Center
Yorktown Heights, New York

Po-Wei Wang
National Taiwan University
Taipei, Taiwan

Ning Xu
University of Illinois at Urbana-Champaign
Urbana, Illinois

Philip S. Yu
University of Illinois at Chicago
Chicago, Illinois

ChengXiang Zhai
University of Illinois at Urbana-Champaign
Urbana, Illinois

Preface

The problem of classification is perhaps one of the most widely studied in the data mining and machine learning communities. This problem has been studied by researchers from several disciplines over several decades. Applications of classification include a wide variety of problem domains such as text, multimedia, social networks, and biological data. Furthermore, the problem may be encountered in a number of different scenarios such as streaming or uncertain data. Classification is a rather diverse topic, and the underlying algorithms depend greatly on the data domain and problem scenario.

Therefore, this book will focus on three primary aspects of data classification. The first set of chapters will focus on the core methods for data classification. These include methods such as probabilistic classification, decision trees, rule-based methods, instance-based techniques, SVM methods, and neural networks. The second set of chapters will focus on different problem domains and scenarios such as multimedia data, text data, time-series data, network data, data streams, and uncertain data. The third set of chapters will focus on different variations of the classification problem such as ensemble methods, visual methods, transfer learning, semi-supervised methods, and active learning. These are advanced methods, which can be used to enhance the quality of the underlying classification results.

The classification problem has been addressed by a number of different communities such as pattern recognition, databases, data mining, and machine learning. In some cases, the work by the different communities tends to be fragmented, and has not been addressed in a unified way. This book will make a conscious effort to address the work of the different communities in a unified way. The book will start off with an overview of the basic methods in data classification, and then discuss progressively more refined and complex methods for data classification. Special attention will also be paid to more recent problem domains such as graphs and social networks.

The chapters in the book will be divided into three types:

- **Method Chapters:** These chapters discuss the *key techniques* that are commonly used for classification, such as probabilistic methods, decision trees, rule-based methods, instance-based methods, SVM techniques, and neural networks.

- **Domain Chapters:** These chapters discuss the specific methods used for different *domains* of data such as text data, multimedia data, time-series data, discrete sequence data, network data, and uncertain data. Many of these chapters can also be considered application chapters, because they explore the specific characteristics of the problem in a particular domain. Dedicated chapters are also devoted to large data sets and data streams, because of the recent importance of the big data paradigm.

- **Variations and Insights:** These chapters discuss the *key variations* on the classification process such as classification ensembles, rare-class learning, distance function learning, active learning, and visual learning. Many variations such as transfer learning and semi-supervised learning use side-information in order to enhance the classification results. A separate chapter is also devoted to evaluation aspects of classifiers.

This book is designed to be comprehensive in its coverage of the entire area of classification, and it is hoped that it will serve as a knowledgeable compendium to students and researchers.

Chapter 1

An Introduction to Data Classification

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY
charu@us.ibm.com

1.1 Introduction 2
1.2 Common Techniques in Data Classification 4
1.2.1 Feature Selection Methods 4
1.2.2 Probabilistic Methods 6
1.2.3 Decision Trees 7
1.2.4 Rule-Based Methods 9
1.2.5 Instance-Based Learning 11
1.2.6 SVM Classifiers 11
1.2.7 Neural Networks 14
1.3 Handing Different Data Types 16
1.3.1 Large Scale Data: Big Data and Data Streams 16
1.3.1.1 Data Streams 16
1.3.1.2 The Big Data Framework 17
1.3.2 Text Classification 18
1.3.3 Multimedia Classification 20
1.3.4 Time Series and Sequence Data Classification 20
1.3.5 Network Data Classification 21
1.3.6 Uncertain Data Classification 21
1.4 Variations on Data Classification 22
1.4.1 Rare Class Learning 22
1.4.2 Distance Function Learning 22
1.4.3 Ensemble Learning for Data Classification 23
1.4.4 Enhancing Classification Methods with Additional Data 24
1.4.4.1 Semi-Supervised Learning 24
1.4.4.2 Transfer Learning 26
1.4.5 Incorporating Human Feedback 27
1.4.5.1 Active Learning 28
1.4.5.2 Visual Learning 29
1.4.6 Evaluating Classification Algorithms 30
1.5 Discussion and Conclusions 31
Bibliography 31

1.1 Introduction

The problem of data classification has numerous applications in a wide variety of mining applications. This is because the problem attempts to learn the relationship between a set of *feature variables* and a *target variable* of interest. Since many practical problems can be expressed as associations between feature and target variables, this provides a broad range of applicability of this model. The problem of classification may be stated as follows:

Given a set of training data points along with associated training labels, determine the class label for an unlabeled test instance.

Numerous variations of this problem can be defined over different settings. Excellent overviews on data classification may be found in [39, 50, 63, 85]. Classification algorithms typically contain two phases:

- *Training Phase:* In this phase, a model is constructed from the training instances.
- *Testing Phase:* In this phase, the model is used to assign a label to an unlabeled test instance.

In some cases, such as lazy learning, the training phase is omitted entirely, and the classification is performed directly from the relationship of the training instances to the test instance. Instance-based methods such as the nearest neighbor classifiers are examples of such a scenario. Even in such cases, a pre-processing phase such as a nearest neighbor index construction may be performed in order to ensure efficiency during the testing phase.

The output of a classification algorithm may be presented for a test instance in one of two ways:

1. *Discrete Label:* In this case, a label is returned for the test instance.
2. *Numerical Score:* In this case, a numerical score is returned for each class label and test instance combination. Note that the numerical score can be converted to a discrete label for a test instance, by picking the class with the highest score for that test instance. The advantage of a numerical score is that it now becomes possible to compare the relative propensity of different test instances to belong to a particular class of importance, and rank them if needed. Such methods are used often in rare class detection problems, where the original class distribution is highly imbalanced, and the discovery of some classes is more valuable than others.

The classification problem thus segments the unseen test instances into groups, as defined by the class label. While the segmentation of examples into groups is also done by clustering, there is a key difference between the two problems. In the case of clustering, the segmentation is done using similarities between the feature variables, with no prior understanding of the structure of the groups. In the case of classification, the segmentation is done on the basis of a *training data* set, which encodes knowledge about the structure of the groups in the form of a *target variable*. Thus, while the segmentations of the data are usually related to notions of similarity, as in clustering, significant deviations from the similarity-based segmentation may be achieved in practical settings. As a result, the classification problem is referred to as *supervised learning*, just as clustering is referred to as *unsupervised learning*. The supervision process often provides significant application-specific utility, because the class labels may represent important properties of interest.

Some common application domains in which the classification problem arises, are as follows:

- **Customer Target Marketing:** Since the classification problem relates feature variables to target classes, this method is extremely popular for the problem of customer target marketing.

In such cases, feature variables describing the customer may be used to predict their buying interests on the basis of previous training examples. The target variable may encode the buying interest of the customer.

- **Medical Disease Diagnosis:** In recent years, the use of data mining methods in medical technology has gained increasing traction. The features may be extracted from the medical records, and the class labels correspond to whether or not a patient may pick up a disease in the future. In these cases, it is desirable to make disease predictions with the use of such information.
- **Supervised Event Detection:** In many temporal scenarios, class labels may be associated with time stamps corresponding to unusual events. For example, an intrusion activity may be represented as a class label. In such cases, time-series classification methods can be very useful.
- **Multimedia Data Analysis:** It is often desirable to perform classification of large volumes of multimedia data such as photos, videos, audio or other more complex multimedia data. Multimedia data analysis can often be challenging, because of the complexity of the underlying feature space and the semantic gap between the feature values and corresponding inferences.
- **Biological Data Analysis:** Biological data is often represented as discrete sequences, in which it is desirable to predict the properties of particular sequences. In some cases, the biological data is also expressed in the form of networks. Therefore, classification methods can be applied in a variety of different ways in this scenario.
- **Document Categorization and Filtering:** Many applications, such as newswire services, require the classification of large numbers of documents in real time. This application is referred to as document categorization, and is an important area of research in its own right.
- **Social Network Analysis:** Many forms of social network analysis, such as collective classification, associate labels with the underlying nodes. These are then used in order to predict the labels of other nodes. Such applications are very useful for predicting useful properties of actors in a social network.

The diversity of problems that can be addressed by classification algorithms is significant, and covers many domains. It is impossible to exhaustively discuss all such applications in either a single chapter or book. Therefore, this book will organize the area of classification into key topics of interest. The work in the data classification area typically falls into a number of broad categories;

- **Technique-centered:** The problem of data classification can be solved using numerous classes of techniques such as decision trees, rule-based methods, neural networks, SVM methods, nearest neighbor methods, and probabilistic methods. This book will cover the most popular classification methods in the literature comprehensively.
- **Data-Type Centered:** Many different data types are created by different applications. Some examples of different data types include text, multimedia, uncertain data, time series, discrete sequence, and network data. Each of these different data types requires the design of different techniques, each of which can be quite different.
- **Variations on Classification Analysis:** Numerous variations on the standard classification problem exist, which deal with more challenging scenarios such as rare class learning, transfer learning, semi-supervised learning, or active learning. Alternatively, different variations of classification, such as ensemble analysis, can be used in order to improve the effectiveness of classification algorithms. These issues are of course closely related to issues of model evaluation. All these issues will be discussed extensively in this book.

This chapter will discuss each of these issues in detail, and will also discuss how the organization of the book relates to these different areas of data classification. The chapter is organized as follows. The next section discusses the common techniques that are used for data classification. Section 1.3 explores the use of different data types in the classification process. Section 1.4 discusses the different variations of data classification. Section 1.5 discusses the conclusions and summary.

1.2 Common Techniques in Data Classification

In this section, the different methods that are commonly used for data classification will be discussed. These methods will also be associated with the different chapters in this book. It should be pointed out that these methods represent the most *common* techniques used for data classification, and it is difficult to comprehensively discuss all the methods in a single book. The most common methods used in data classification are decision trees, rule-based methods, probabilistic methods, SVM methods, instance-based methods, and neural networks. Each of these methods will be discussed briefly in this chapter, and all of them will be covered comprehensively in the different chapters of this book.

1.2.1 Feature Selection Methods

The first phase of virtually all classification algorithms is that of feature selection. In most data mining scenarios, a wide variety of features are collected by individuals who are often not domain experts. Clearly, the irrelevant features may often result in poor modeling, since they are not well related to the class label. In fact, such features will typically worsen the classification accuracy because of overfitting, when the training data set is small and such features are allowed to be a part of the training model. For example, consider a medical example where the features from the blood work of different patients are used to predict a particular disease. Clearly, a feature such as the *Cholesterol* level is predictive of heart disease, whereas a feature[1] such as *PSA* level is not predictive of heart disease. However, if a small training data set is used, the *PSA* level may have freak correlations with heart disease because of random variations. While the impact of a single variable may be small, the cumulative effect of many irrelevant features can be significant. This will result in a training model, that *generalizes* poorly to unseen test instances. Therefore, it is critical to use the correct features during the training process.

There are two broad kinds of feature selection methods:

1. *Filter Models:* In these cases, a crisp criterion on a single feature, or a subset of features, is used to evaluate their suitability for classification. This method is independent of the specific algorithm being used.

2. *Wrapper Models:* In these cases, the feature selection process is embedded into a classification algorithm, in order to make the feature selection process sensitive to the classification algorithm. This approach recognizes the fact that different algorithms may work better with different features.

In order to perform feature selection with filter models, a number of different measures are used in order to quantify the relevance of a feature to the classification process. Typically, these measures compute the imbalance of the feature values over different ranges of the attribute, which may either be discrete or numerical. Some examples are as follows:

[1] This feature is used to measure prostate cancer in men.

- *Gini Index:* Let $p_1 \ldots p_k$ be the fraction of classes that correspond to a particular *value* of the discrete attribute. Then, the gini-index of that value of the discrete attribute is given by:

$$G = 1 - \sum_{i=1}^{k} p_i^2 \tag{1.1}$$

  The value of G ranges between 0 and $1 - 1/k$. Smaller values are more indicative of class imbalance. This indicates that the feature value is more discriminative for classification. The overall gini-index for the attribute can be measured by weighted averaging over different values of the discrete attribute, or by using the maximum gini-index over any of the different discrete values. Different strategies may be more desirable for different scenarios, though the weighted average is more commonly used.

- *Entropy:* The entropy of a particular value of the discrete attribute is measured as follows:

$$E = -\sum_{i=1}^{k} p_i \cdot \log(p_i) \tag{1.2}$$

  The same notations are used above, as for the case of the gini-index. The value of the entropy lies between 0 and $\log(k)$, with smaller values being more indicative of class skew.

- *Fisher's Index:* The Fisher's index measures the ratio of the between class scatter to the within class scatter. Therefore, if p_j is the fraction of training examples belonging to class j, μ_j is the mean of a particular feature for class j, μ is the global mean for that feature, and σ_j is the standard deviation of that feature for class j, then the Fisher score F can be computed as follows:

$$F = \frac{\sum_{j=1}^{k} p_j \cdot (\mu_j - \mu)^2}{\sum_{j=1}^{k} p_j \cdot \sigma_j^2} \tag{1.3}$$

A wide variety of other measures such as the χ^2-statistic and mutual information are also available in order to quantify the discriminative power of attributes. An approach known as the Fisher's discriminant [61] is also used in order to *combine* the different features into directions in the data that are highly relevant to classification. Such methods are of course feature *transformation* methods, which are also closely related to feature selection methods, just as unsupervised dimensionality reduction methods are related to unsupervised feature selection methods.

The Fisher's discriminant will be explained below for the two-class problem. Let $\overline{\mu_0}$ and $\overline{\mu_1}$ be the d-dimensional row vectors representing the means of the records in the two classes, and let Σ_0 and Σ_1 be the corresponding $d \times d$ covariance matrices, in which the (i, j)th entry represents the covariance between dimensions i and j for that class. Then, the equivalent Fisher score $FS(\overline{V})$ for a d-dimensional row vector $\overline{V}$ may be written as follows:

$$FS(\overline{V}) = \frac{(\overline{V} \cdot (\overline{\mu_0} - \overline{\mu_1}))^2}{\overline{V}(p_0 \cdot \Sigma_0 + p_1 \cdot \Sigma_1)\overline{V}^T} \tag{1.4}$$

This is a generalization of the *axis-parallel* score in Equation 1.3, to an arbitrary direction $\overline{V}$. The goal is to determine a direction $\overline{V}$, which maximizes the Fisher score. It can be shown that the optimal direction $\overline{V^*}$ may be determined by solving a generalized eigenvalue problem, and is given by the following expression:

$$\overline{V^*} = (p_0 \cdot \Sigma_0 + p_1 \cdot \Sigma_1)^{-1}(\overline{\mu_0} - \overline{\mu_1})^T \tag{1.5}$$

If desired, successively orthogonal directions may be determined by iteratively projecting the data onto the residual subspace, after determining the optimal directions one by one.

More generally, it should be pointed out that many features are often closely correlated with one another, and the *additional* utility of an attribute, once a certain set of features have already been selected, is different from its standalone utility. In order to address this issue, the *Minimum Redundancy Maximum Relevance* approach was proposed in [69], in which features are incrementally selected on the basis of their incremental gain on adding them to the feature set. Note that this method is also a filter model, since the evaluation is on a subset of features, and a crisp criterion is used to evaluate the subset.

In wrapper models, the feature selection phase is embedded into an iterative approach with a classification algorithm. In each iteration, the classification algorithm evaluates a particular set of features. This set of features is then augmented using a particular (e.g., greedy) strategy, and tested to see of the quality of the classification improves. Since the classification algorithm is used for evaluation, this approach will generally create a feature set, which is sensitive to the classification algorithm. This approach has been found to be useful in practice, because of the wide diversity of models on data classification. For example, an SVM would tend to prefer features in which the two classes separate out using a linear model, whereas a nearest neighbor classifier would prefer features in which the different classes are clustered into spherical regions. A good survey on feature selection methods may be found in [59]. Feature selection methods are discussed in detail in Chapter 2.

1.2.2 Probabilistic Methods

Probabilistic methods are the most fundamental among all data classification methods. Probabilistic classification algorithms use statistical inference to find the best class for a given example. In addition to simply assigning the best class like other classification algorithms, probabilistic classification algorithms will output a corresponding *posterior* probability of the test instance being a member of each of the possible classes. The posterior probability is defined as the probability after observing the specific characteristics of the test instance. On the other hand, the *prior* probability is simply the fraction of training records belonging to each particular class, with no knowledge of the test instance. After obtaining the posterior probabilities, we use decision theory to determine class membership for each new instance. Basically, there are two ways in which we can estimate the posterior probabilities.

In the first case, the posterior probability of a particular class is estimated by determining the class-conditional probability and the prior class separately and then applying Bayes' theorem to find the parameters. The most well known among these is the Bayes classifier, which is known as a generative model. For ease in discussion, we will assume discrete feature values, though the approach can easily be applied to numerical attributes with the use of discretization methods. Consider a test instance with d different features, which have values $X = \langle x_1 \ldots x_d \rangle$ respectively. Its is desirable to determine the posterior probability that the class $Y(T)$ of the test instance T is i. In other words, we wish to determine the posterior probability $P(Y(T) = i | x_1 \ldots x_d)$. Then, the Bayes rule can be used in order to derive the following:

$$P(Y(T) = i | x_1 \ldots x_d) = P(Y(T) = i) \cdot \frac{P(x_1 \ldots x_d | Y(T) = i)}{P(x_1 \ldots x_d)} \tag{1.6}$$

Since the denominator is constant across all classes, and one only needs to determine the class with the maximum posterior probability, one can approximate the aforementioned expression as follows:

$$P(Y(T) = i | x_1 \ldots x_d) \propto P(Y(T) = i) \cdot P(x_1 \ldots x_d | Y(T) = i) \tag{1.7}$$

The key here is that the expression on the right can be evaluated more easily in a data-driven way, as long as the *naive Bayes assumption* is used for simplification. Specifically, in Equation1.7, the expression $P(Y(T) = i | x_1 \ldots x_d)$ can be expressed as the product of the feature-wise conditional

probabilities.

$$P(x_1 \ldots x_d | Y(T) = i) = \prod_{j=1}^{d} P(x_j | Y(T) = i) \tag{1.8}$$

This is referred to as *conditional independence*, and therefore the Bayes method is referred to as "naive." This simplification is crucial, because these individual probabilities can be estimated from the training data in a more robust way. The naive Bayes theorem is crucial in providing the ability to perform the product-wise simplification. The term $P(x_j|Y(T) = i)$ is computed as the fraction of the records in the portion of the training data corresponding to the ith class, which contains feature value x_j for the jth attribute. If desired, Laplacian smoothing can be used in cases when enough data is not available to estimate these values robustly. This is quite often the case, when a small amount of training data may contain few or no training records containing a particular feature value. The Bayes rule has been used quite successfully in the context of a wide variety of applications, and is particularly popular in the context of text classification. In spite of the naive independence assumption, the Bayes model seems to be quite effective in practice. A detailed discussion of the naive assumption in the context of the effectiveness of the Bayes classifier may be found in [38].

Another probabilistic approach is to directly model the posterior probability, by learning a discriminative function that maps an input feature vector directly onto a class label. This approach is often referred to as a discriminative model. Logistic regression is a popular discriminative classifier, and its goal is to directly estimate the posterior probability $P(Y(T) = i|X)$ from the training data. Formally, the logistic regression model is defined as

$$P(Y(T) = i|X) = \frac{1}{1 + e^{-\theta^T X}}, \tag{1.9}$$

where θ is the vector of parameters to be estimated. In general, maximum likelihood is used to determine the parameters of the logistic regression. To handle overfitting problems in logistic regression, regularization is introduced to penalize the log likelihood function for large values of θ. The logistic regression model has been extensively used in numerous disciplines, including the Web, and the medical and social science fields.

A variety of other probabilistic models are known in the literature, such as probabilistic graphical models, and conditional random fields. An overview of probabilistic methods for data classification are found in [20, 64]. Probabilistic methods for data classification are discussed in Chapter 3.

1.2.3 Decision Trees

Decision trees create a hierarchical partitioning of the data, which relates the different partitions at the leaf level to the different classes. The hierarchical partitioning at each level is created with the use of a *split criterion*. The split criterion may either use a condition (or predicate) on a single attribute, or it may contain a condition on multiple attributes. The former is referred to as a univariate split, whereas the latter is referred to as a multivariate split. The overall approach is to try to recursively split the training data so as to maximize the discrimination among the different classes over different nodes. The discrimination among the different classes is maximized, when the level of skew among the different classes in a given node is maximized. A measure such as the gini-index or entropy is used in order to quantify this skew. For example, if $p_1 \ldots p_k$ is the fraction of the records belonging to the k different classes in a node N, then the gini-index $G(N)$ of the node N is defined as follows:

$$G(N) = 1 - \sum_{i=1}^{k} p_i^2 \tag{1.10}$$

The value of $G(N)$ lies between 0 and $1 - 1/k$. The smaller the value of $G(N)$, the greater the skew. In the cases where the classes are evenly balanced, the value is $1 - 1/k$. An alternative measure is

TABLE 1.1: Training Data Snapshot Relating Cardiovascular Risk Based on Previous Events to Different Blood Parameters

Patient Name	CRP Level	Cholestrol	High Risk? (Class Label)
Mary	3.2	170	Y
Joe	0.9	273	N
Jack	2.5	213	Y
Jane	1.7	229	N
Tom	1.1	160	N
Peter	1.9	205	N
Elizabeth	8.1	160	Y
Lata	1.3	171	N
Daniela	4.5	133	Y
Eric	11.4	122	N
Michael	1.8	280	Y

the entropy $E(N)$:

$$E(N) = -\sum_{i=1}^{k} p_i \cdot \log(p_i) \tag{1.11}$$

The value of the entropy lies[2] between 0 and $\log(k)$. The value is $\log(k)$, when the records are perfectly balanced among the different classes. This corresponds to the scenario with maximum entropy. The smaller the entropy, the greater the skew in the data. Thus, the gini-index and entropy provide an effective way to evaluate the quality of a node in terms of its level of discrimination between the different classes.

While constructing the training model, the split is performed, so as to minimize the weighted sum of the gini-index or entropy of the two nodes. This step is performed recursively, until a termination criterion is satisfied. The most obvious termination criterion is one where all data records in the node belong to the same class. More generally, the termination criterion requires either a minimum level of skew or purity, or a minimum number of records in the node in order to avoid overfitting. One problem in decision tree construction is that there is no way to *predict* the best time to stop decision tree growth, in order to prevent overfitting. Therefore, in many variations, the decision tree is pruned in order to remove nodes that may correspond to overfitting. There are different ways of pruning the decision tree. One way of pruning is to use a minimum description length principle in deciding when to prune a node from the tree. Another approach is to hold out a small portion of the training data during the decision tree growth phase. It is then tested to see whether replacing a subtree with a single node improves the classification accuracy on the hold out set. If this is the case, then the pruning is performed. In the testing phase, a test instance is assigned to an appropriate path in the decision tree, based on the evaluation of the split criteria in a hierarchical decision process. The class label of the corresponding leaf node is reported as the relevant one.

Figure 1.1 provides an example of how the decision tree is constructed. Here, we have illustrated a case where the two measures (features) of the blood parameters of patients are used in order to assess the level of cardiovascular risk in the patient. The two measures are the *C-Reactive Protein (CRP)* level and *Cholesterol* level, which are well known parameters related to cardiovascular risk. It is assumed that a training data set is available, which is already labeled into high risk and low risk patients, based on previous cardiovascular events such as myocardial infarctions or strokes. At the same time, it is assumed that the feature values of the blood parameters for these patients are available. A snapshot of this data is illustrated in Table 1.1. It is evident from the training data that

[2]The value of the expression at $p_i = 0$ needs to be evaluated at the limit.

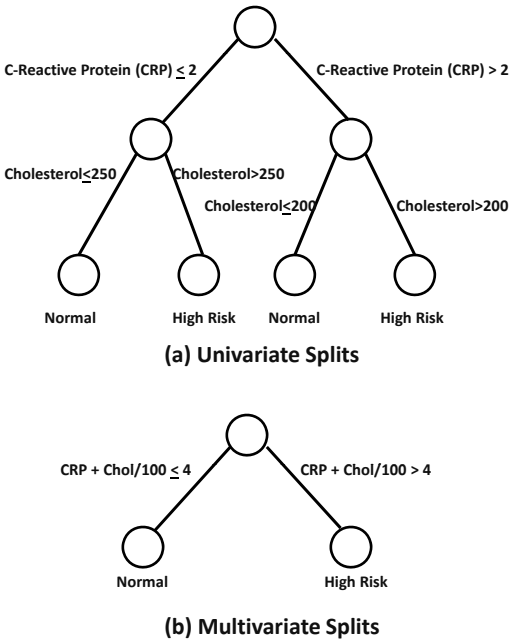


FIGURE 1.1: Illustration of univariate and multivariate splits for decision tree construction.

higher CRP and Cholesterol levels correspond to greater risk, though it is possible to reach more definitive conclusions by combining the two.

An example of a decision tree that constructs the classification model on the basis of the two features is illustrated in Figure 1.1(a). This decision tree uses univariate splits, by first partitioning on the CRP level, and then using a split criterion on the Cholesterol level. Note that the Cholesterol split criteria in the two CRP branches of the tree are different. In principle, different features can be used to split different nodes at the same level of the tree. It is also sometimes possible to use conditions on multiple attributes in order to create more powerful splits at a particular level of the tree. An example is illustrated in Figure 1.1(b), where a linear combination of the two attributes provides a much more powerful split than a single attribute. The split condition is as follows:

$$\texttt{CRP + Cholestrol}/100 \leq 4$$

Note that a single condition such as this is able to partition the training data very well into the two classes (with a few exceptions). Therefore, the split is more powerful in discriminating between the two classes in a smaller number of levels of the decision tree. Where possible, it is desirable to construct more compact decision trees in order to obtain the most accurate results. Such splits are referred to as multivariate splits. Some of the earliest methods for decision tree construction include C4.5 [72], ID3 [73], and CART [22]. A detailed discussion of decision trees may be found in [22, 65, 72, 73]. Decision trees are discussed in Chapter 4.

1.2.4 Rule-Based Methods

Rule-based methods are closely related to decision trees, except that they do not create a strict hierarchical partitioning of the training data. Rather, overlaps are allowed in order to create greater robustness for the training model. Any path in a decision tree may be interpreted as a rule, which

assigns a test instance to a particular label. For example, for the case of the decision tree illustrated in Figure 1.1(a), the rightmost path corresponds to the following rule:

$$\texttt{CRP} > 2 \ \texttt{\&} \ \texttt{Cholestrol} > 200 \Rightarrow \texttt{HighRisk}$$

It is possible to create a set of disjoint rules from the different paths in the decision tree. In fact, a number of methods such as *C4.5*, create related models for both decision tree construction and rule construction. The corresponding rule-based classifier is referred to as *C4.5Rules*.

Rule-based classifiers can be viewed as more general models than decision tree models. While decision trees require the induced rule sets to be *non-overlapping*, this is not the case for rule-based classifiers. For example, consider the following rule:

$$\texttt{CRP} > 3 \Rightarrow \texttt{HighRisk}$$

Clearly, this rule overlaps with the previous rule, and is also quite relevant to the prediction of a given test instance. In rule-based methods, a set of rules is mined from the training data in the first phase (or training phase). During the testing phase, it is determined which rules are relevant to the test instance and the final result is based on a combination of the class values predicted by the different rules.

In many cases, it may be possible to create rules that possibly conflict with one another on the right hand side for a particular test instance. Therefore, it is important to design methods that can effectively determine a resolution to these conflicts. The method of resolution depends upon whether the rule sets are ordered or unordered. If the rule sets are ordered, then the top matching rules can be used to make the prediction. If the rule sets are unordered, then the rules can be used to vote on the test instance. Numerous methods such as *Classification based on Associations (CBA)* [58], *CN2* [31], and *RIPPER* [26] have been proposed in the literature, which use a variety of rule induction methods, based on different ways of mining and prioritizing the rules.

Methods such as *CN2* and *RIPPER* use the *sequential covering paradigm*, where rules with high accuracy and coverage are sequentially mined from the training data. The idea is that a rule is grown corresponding to specific target class, and then all training instances matching (or *covering*) the antecedent of that rule are removed. This approach is applied repeatedly, until only training instances of a particular class remain in the data. This constitutes the *default class*, which is selected for a test instance, when no rule is fired. The process of mining a rule for the training data is referred to as *rule growth*. The growth of a rule involves the successive addition of conjuncts to the left-hand side of the rule, after the selection of a particular consequent class. This can be viewed as growing a single "best" path in a decision tree, by adding conditions (split criteria) to the left-hand side of the rule. After the rule growth phase, a rule-pruning phase is used, which is analogous to decision tree construction. In this sense, the rule-growth of rule-based classifiers share a number of conceptual similarities with decision tree classifiers. These rules are ranked in the same order as they are mined from the training data. For a given test instance, the class variable in the consequent of the first matching rule is reported. If no matching rule is found, then the default class is reported as the relevant one.

Methods such as *CBA* [58] use the traditional association rule framework, in which rules are determined with the use of specific support and confidence measures. Therefore, these methods are referred to as associative classifiers. It is also relatively easy to prioritize these rules with the use of these parameters. The final classification can be performed by either using the majority vote from the matching rules, or by picking the top ranked rule(s) for classification. Typically, the confidence of the rule is used to prioritize them, and the support is used to prune for statistical significance. A single catch-all rule is also created for test instances that are not covered by any rule. Typically, this catch-all rule might correspond to the majority class among training instances not covered by any rule. Rule-based methods tend to be more robust than decision trees, because they are not

restricted to a strict hierarchical partitioning of the data. This is most evident from the relative performance of these methods in some sparse high dimensional domains such as text. For example, while many rule-based methods such as *RIPPER* are frequently used for the text domain, decision trees are used rarely for text. Another advantage of these methods is that they are relatively easy to generalize to different data types such as sequences, XML or graph data [14, 93]. In such cases, the left-hand side of the rule needs to be defined in a way that is specific for that data domain. For example, for a sequence classification problem [14], the left-hand side of the rule corresponds to a sequence of symbols. For a graph-classification problem, the left-hand side of the rule corresponds to a frequent structure [93]. Therefore, while rule-based methods are related to decision trees, they have significantly greater expressive power. Rule-based methods are discussed in detail in Chapter 5.

1.2.5 Instance-Based Learning

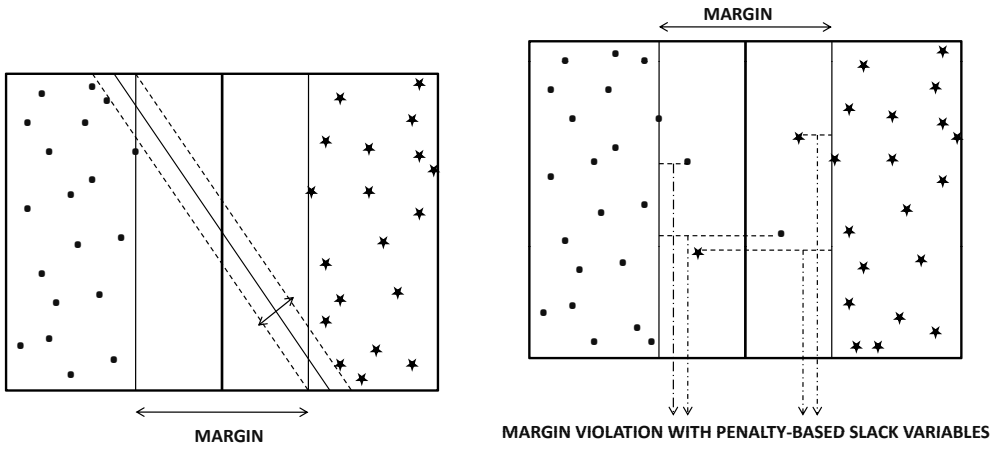
In instance-based learning, the first phase of constructing the training model is often dispensed with. The test instance is directly related to the training instances in order to create a classification model. Such methods are referred to as *lazy learning methods*, because they wait for knowledge of the test instance in order to create a locally optimized model, which is specific to the test instance. The advantage of such methods is that they can be directly tailored to the particular test instance, and can avoid the information loss associated with the incompleteness of any training model. An overview of instance-based methods may be found in [15, 16, 89].

An example of a very simple instance-based method is the nearest neighbor classifier. In the nearest neighbor classifier, the top k nearest neighbors in the training data are found to the given test instance. The class label with the largest presence among the k nearest neighbors is reported as the relevant class label. If desired, the approach can be made faster with the use of nearest neighbor index construction. Many variations of the basic instance-based learning algorithm are possible, wherein aggregates of the training instances may be used for classification. For example, small clusters can be created from the instances of each class, and the centroid of each cluster may be used as a new instance. Such an approach is much more efficient and also more robust because of the reduction of noise associated with the clustering phase which aggregates the noisy records into more robust aggregates. Other variations of instance-based learning use different variations on the distance function used for classification. For example, methods that are based on the Mahalanobis distance or Fisher's discriminant may be used for more accurate results. The problem of distance function design is intimately related to the problem of instance-based learning. Therefore, separate chapters have been devoted in this book to these topics.

A particular form of instance-based learning, is one where the nearest neighbor classifier is not explicitly used. This is because the distribution of the class labels may not match with the notion of proximity defined by a particular distance function. Rather, a locally optimized classifier is constructed using the examples in the neighborhood of a test instance. Thus, the neighborhood is used only to define the neighborhood in which the classification model is constructed in a lazy way. Local classifiers are generally more accurate, because of the simplification of the class distribution within the locality of the test instance. This approach is more generally referred to as *lazy learning*. This is a more general notion of instance-based learning than traditional nearest neighbor classifiers. Methods for instance-based classification are discussed in Chapter 6. Methods for distance-function learning are discussed in Chapter 18.

1.2.6 SVM Classifiers

SVM methods use linear conditions in order to separate out the classes from one another. The idea is to use a linear condition that separates the two classes from each other as well as possible. Consider the medical example discussed earlier, where the risk of cardiovascular disease is related to diagnostic features from patients.



(a) Hard separation (b) Soft separation

FIGURE 1.2: Hard and soft support vector machines.

$$\texttt{CRP + Cholestrol}/100 \leq 4$$

In such a case, the split condition in the multivariate case may also be used as stand-alone condition for classification. This, a SVM classifier, may be considered a single level decision tree with a very carefully chosen multivariate split condition. Clearly, since the effectiveness of the approach depends only on a single separating hyperplane, it is critical to define this separation carefully.

Support vector machines are generally defined for binary classification problems. Therefore, the class variable y_i for the ith training instance $\overline{X_i}$ is assumed to be drawn from $\{-1,+1\}$. The most important criterion, which is commonly used for SVM classification, is that of the *maximum margin hyperplane*. In order to understand this point, consider the case of linearly separable data illustrated in Figure 1.2(a). Two possible separating hyperplanes, with their corresponding *support vectors* and *margins* have been illustrated in the figure. It is evident that one of the separating hyperplanes has a much larger margin than the other, and is therefore more desirable because of its greater generality for unseen test examples. Therefore, one of the important criteria for support vector machines is to achieve maximum margin separation of the hyperplanes.

In general, it is assumed for d dimensional data that the separating hyperplane is of the form $\overline{W} \cdot \overline{X} + b = 0$. Here $\overline{W}$ is a d-dimensional vector representing the coefficients of the hyperplane of separation, and b is a constant. Without loss of generality, it may be assumed (because of appropriate coefficient scaling) that the two symmetric support vectors have the form $\overline{W} \cdot \overline{X} + b = 1$ and $\overline{W} \cdot \overline{X} + b = -1$. The coefficients $\overline{W}$ and b need to be learned from the training data $\mathcal{D}$ in order to maximize the margin of separation between these two parallel hyperplanes. It can shown from elementary linear algebra that the distance between these two hyperplanes is $2/||\overline{W}||$. Maximizing this objective function is equivalent to minimizing $||\overline{W}||^2/2$. The problem constraints are defined by the fact that the training data points for each class are on one side of the support vector. Therefore, these constraints are as follows:

$$\overline{W} \cdot \overline{X_i} + b \geq +1 \quad \forall i : y_i = +1 \tag{1.12}$$

$$\overline{W} \cdot \overline{X_i} + b \leq -1 \quad \forall i : y_i = -1 \tag{1.13}$$

This is a constrained convex quadratic optimization problem, which can be solved using Lagrangian methods. In practice, an off-the-shelf optimization solver may be used to achieve the same goal.

In practice, the data may not be linearly separable. In such cases, soft-margin methods may be used. A slack $\xi_i \geq 0$ is introduced for training instance, and a training instance is allowed to violate the support vector constraint, for a penalty, which is dependent on the slack. This situation is illustrated in Figure 1.2(b). Therefore, the new set of constraints are now as follows:

$$\overline{W} \cdot \overline{X_i} + b \geq +1 - \xi_i \quad \forall i : y_i = +1 \tag{1.14}$$

$$\overline{W} \cdot \overline{X_i} + b \leq -1 + \xi_i \quad \forall i : y_i = -1 \tag{1.15}$$

$$\xi_i \geq 0 \tag{1.16}$$

Note that additional non-negativity constraints also need to be imposed in the slack variables. The objective function is now $||\overline{W}||^2/2 + C \cdot \sum_{i=1}^{n} \xi_i$. The constant C regulates the importance of the margin and the slack requirements. In other words, small values of C make the approach closer to soft-margin SVM, whereas large values of C make the approach more of the hard-margin SVM. It is also possible to solve this problem using off-the-shelf optimization solvers.

It is also possible to use transformations on the feature variables in order to design non-linear SVM methods. In practice, non-linear SVM methods are learned using kernel methods. The key idea here is that SVM formulations can be solved using only pairwise dot products (similarity values) between objects. In other words, the optimal decision about the class label of a test instance, from the solution to the quadratic optimization problem in this section, can be expressed in terms of the following:

1. Pairwise dot products of different training instances.
2. Pairwise dot product of the test instance and different training instances.

The reader is advised to refer to [84] for the specific details of the solution to the optimization formulation. The dot product between a pair of instances can be viewed as notion of similarity among them. Therefore, the aforementioned observations imply that it is possible to perform SVM classification, with pairwise similarity information between training data pairs and training-test data pairs. The actual feature values are not required.

This opens the door for using transformations, which are represented by their similarity values. These similarities can be viewed as kernel functions $K(\overline{X}, \overline{Y})$, which measure similarities between the points $\overline{X}$ and $\overline{Y}$. Conceptually, the kernel function may be viewed as dot product between the pair of points in a newly transformed space (denoted by mapping function $\Phi(\cdot)$). However, this transformation does not need to be explicitly computed, as long as the kernel function (dot product) $K(\overline{X}, \overline{Y})$ is already available:

$$K(\overline{X}, \overline{Y}) = \Phi(\overline{X}) \cdot \Phi(\overline{Y}) \tag{1.17}$$

Therefore, all computations can be performed in the original space using the dot products implied by the kernel function. Some interesting examples of kernel functions include the Gaussian radial basis function, polynomial kernel, and hyperbolic tangent, which are listed below in the same order.

$$K(\overline{X_i}, \overline{X_j}) = e^{-||\overline{X_i} - \overline{X_j}||^2/2\sigma^2} \tag{1.18}$$

$$K(\overline{X_i}, \overline{X_j}) = (\overline{X_i} \cdot \overline{X_j} + 1)^h \tag{1.19}$$

$$K(\overline{X_i}, \overline{X_j}) = \tanh(\kappa \overline{X_i} \cdot \overline{X_j} - \delta) \tag{1.20}$$

These different functions result in different kinds of nonlinear decision boundaries in the original space, but they correspond to a linear separator in the transformed space. The performance of a classifier can be sensitive to the choice of the kernel used for the transformation. One advantage of kernel methods is that they can also be extended to arbitrary data types, as long as appropriate pairwise similarities can be defined.

The major downside of SVM methods is that they are slow. However, they are very popular and tend to have high accuracy in many practical domains such as text. An introduction to SVM methods may be found in [30, 46, 75, 76, 85]. Kernel methods for support vector machines are discussed in [75]. SVM methods are discussed in detail in Chapter 7.

1.2.7 Neural Networks

Neural networks attempt to simulate biological systems, corresponding to the human brain. In the human brain, neurons are connected to one another via points, which are referred to as *synapses*. In biological systems, learning is performed by changing the strength of the synaptic connections, in response to impulses.

This biological analogy is retained in an artificial neural network. The basic computation unit in an artificial neural network is a *neuron* or *unit*. These units can be arranged in different kinds of architectures by connections between them. The most basic architecture of the neural network is a perceptron, which contains a set of input nodes and an output node. The output unit receives a set of inputs from the input units. There are d different input units, which is exactly equal to the dimensionality of the underlying data. The data is assumed to be numerical. Categorical data may need to be transformed to binary representations, and therefore the number of inputs may be larger. The output node is associated with a set of weights $\overline{W}$, which are used in order to compute a function $f(\cdot)$ of its inputs. Each component of the weight vector is associated with a connection from the input unit to the output unit. The weights can be viewed as the analogue of the synaptic strengths in biological systems. In the case of a perceptron architecture, the input nodes do not perform any computations. They simply transmit the input attribute forward. Computations are performed only at the output nodes in the basic perceptron architecture. The output node uses its weight vector along with the input attribute values in order to compute a function of the inputs. A typical function, which is computed at the output nodes, is the signed linear function:

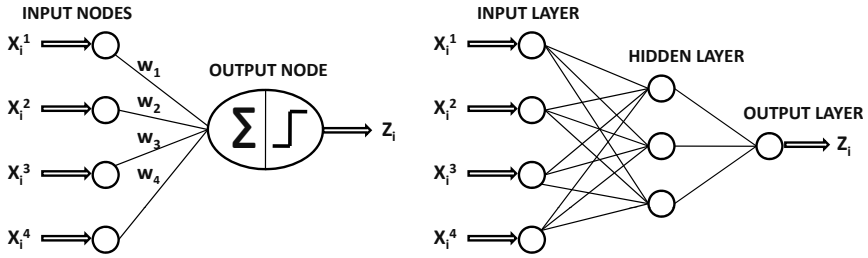
$$z_i = \text{sign}\{\overline{W} \cdot \overline{X_i} + b\} \tag{1.21}$$

The output is a predicted value of the binary class variable, which is assumed to be drawn from $\{-1, +1\}$. The notation b denotes the bias. Thus, for a vector $\overline{X_i}$ drawn from a dimensionality of d, the weight vector $\overline{W}$ should also contain d elements. Now consider a binary classification problem, in which all labels are drawn from $\{+1, -1\}$. We assume that the class label of $\overline{X_i}$ is denoted by y_i. In that case, the sign of the predicted function z_i yields the class label. An example of the perceptron architecture is illustrated in Figure 1.3(a). Thus, the goal of the approach is to *learn* the set of weights $\overline{W}$ with the use of the training data, so as to minimize the least squares error $(y_i - z_i)^2$. The idea is that we start off with random weights and gradually update them, when a mistake is made by applying the current function on the training example. The magnitude of the update is regulated by a learning rate λ. This update is similar to the updates in gradient descent, which are made for least-squares optimization. In the case of neural networks, the update function is as follows.

$$\overline{W}^{t+1} = \overline{W}^t + \lambda(y_i - z_i)\overline{X_i} \tag{1.22}$$

Here, $\overline{W}^t$ is the value of the weight vector in the tth iteration. It is not difficult to show that the incremental update vector is related to the negative gradient of $(y_i - z_i)^2$ with respect to $\overline{W}$. It is also easy to see that updates are made to the weights, only when mistakes are made in classification. When the outputs are correct, the incremental change to the weights is zero.

The similarity to support vector machines is quite striking, in the sense that a linear function is also learned in this case, and the sign of the linear function predicts the class label. In fact, the perceptron model and support vector machines are closely related, in that both are linear function approximators. In the case of support vector machines, this is achieved with the use of maximum margin optimization. In the case of neural networks, this is achieved with the use of an incremental



(a) Perceptron (b) Multilayer

FIGURE 1.3: Single and multilayer neural networks.

learning algorithm, which is approximately equivalent to least squares error optimization of the prediction.

The constant λ regulates the learning rate. The choice of learning rate is sometimes important, because learning rates that are too small will result in very slow training. On the other hand, if the learning rates are too fast, this will result in oscillation between suboptimal solutions. In practice, the learning rates are fast initially, and then allowed to gradually slow down over time. The idea here is that initially large steps are likely to be helpful, but are then reduced in size to prevent oscillation between suboptimal solutions. For example, after t iterations, the learning rate may be chosen to be proportional to $1/t$.

The aforementioned discussion was based on the simple perceptron architecture, which can model only linear relationships. In practice, the neural network is arranged in three layers, referred to as the *input layer*, *hidden layer*, and the *output layer*. The input layer only transmits the inputs forward, and therefore, there are really only two layers to the neural network, which can perform computations. Within the hidden layer, there can be any number of layers of neurons. In such cases, there can be an arbitrary number of layers in the neural network. In practice, there is only one hidden layer, which leads to a 2-layer network. An example of a multilayer network is illustrated in Figure 1.3(b). The perceptron can be viewed as a very special kind of neural network, which contains only a single layer of neurons (corresponding to the output node). Multilayer neural networks allow the approximation of nonlinear functions, and complex decision boundaries, by an appropriate choice of the network topology, and non-linear functions at the nodes. In these cases, a logistic or sigmoid function known as a *squashing function* is also applied to the inputs of neurons in order to model non-linear characteristics. It is possible to use different non-linear functions at different nodes. Such general architectures are very powerful in approximating arbitrary functions in a neural network, given enough training data and training time. This is the reason that neural networks are sometimes referred to as *universal function approximators*.

In the case of single layer perceptron algorthms, the training process is easy to perform by using a gradient descent approach. The major challenge in training multilayer networks is that it is no longer known for intermediate (hidden layer) nodes, what their "expected" output should be. This is only known for the final output node. Therefore, some kind of "error feedback" is required, in order to determine the changes in the weights at the intermediate nodes. The training process proceeds in two phases, one of which is in the forward direction, and the other is in the backward direction.

1. *Forward Phase:* In the forward phase, the activation function is repeatedly applied to propagate the inputs from the neural network in the forward direction. Since the final output is supposed to match the class label, the final output at the output layer provides an error value, depending on the training label value. This error is then used to update the weights of the output layer, and propagate the weight updates backwards in the next phase.

2. *Backpropagation Phase:* In the backward phase, the errors are propagated backwards through the neural network layers. This leads to the updating of the weights in the neurons of the different layers. The gradients at the previous layers are learned as a function of the errors and weights in the layer ahead of it. The learning rate λ plays an important role in regulating the rate of learning.

In practice, any arbitrary function can be approximated well by a neural network. The price of this generality is that neural networks are often quite slow in practice. They are also sensitive to noise, and can sometimes overfit the training data.

The previous discussion assumed only binary labels. It is possible to create a k-label neural network, by either using a multiclass "one-versus-all" meta-algorithm, or by creating a neural network architecture in which the number of output nodes is equal to the number of class labels. Each output represents prediction to a particular label value. A number of implementations of neural network methods have been studied in [35,57,66,77,88], and many of these implementations are designed in the context of text data. It should be pointed out that both neural networks and SVM classifiers use a linear model that is quite similar. The main difference between the two is in how the optimal linear hyperplane is determined. Rather than using a direct optimization methodology, neural networks use a *mistake-driven* approach to data classification [35]. Neural networks are described in detail in [19,51]. This topic is addressed in detail in Chapter 8.

1.3 Handing Different Data Types

Different data types require the use of different techniques for data classification. This is because the choice of data type often qualifies the kind of problem that is solved by the classification approach. In this section, we will discuss the different data types commonly studied in classification problems, which may require a certain level of special handling.

1.3.1 Large Scale Data: Big Data and Data Streams

With the increasing ability to collect different types of large scale data, the problems of scale have become a challenge to the classification process. Clearly, larger data sets allow the creation of more accurate and sophisticated models. However, this is not necessarily helpful, if one is computationally constrained by problems of scale. Data streams and big data analysis have different challenges. In the former case, real time processing creates challenges, whereas in the latter case, the problem is created by the fact that computation and data access over extremely large amounts of data is inefficient. It is often difficult to compute summary statistics from large volumes, because the access needs to be done in a distributed way, and it is too expensive to shuffle large amounts of data around. Each of these challenges will be discussed in this subsection.

1.3.1.1 Data Streams

The ability to continuously collect and process large volumes of data has lead to the popularity of data streams [4]. In the streaming scenario, two primary problems arise in the construction of training models.

- *One-pass Constraint:* Since data streams have very large volume, all processing algorithms need to perform their computations in a single pass over the data. This is a significant challenge, because it excludes the use of many iterative algorithms that work robustly over static data sets. Therefore, it is crucial to design the training models in an efficient way.

- *Concept Drift:* The data streams are typically created by a generating process, which may change over time. This results in *concept drift*, which corresponds to changes in the underlying stream patterns over time. The presence of concept drift can be detrimental to classification algorithms, because models become stale over time. Therefore, it is crucial to adjust the model in an incremental way, so that it achieves high accuracy over current test instances.

- *Massive Domain Constraint:* The streaming scenario often contains discrete attributes that take on millions of possible values. This is because streaming items are often associated with discrete identifiers. Examples could be email addresses in an email addresses, IP addresses in a network packet stream, and URLs in a click stream extracted from proxy Web logs. The massive domain problem is ubiquitous in streaming applications. In fact, many synopsis data structures, such as the count-min sketch [33], and the Flajolet-Martin data structure [41], have been designed with this issue in mind. While this issue has not been addressed very extensively in the stream *mining* literature (beyond basic synopsis methods for counting), recent work has made a number of advances in this direction [9].

Conventional classification algorithms need to be appropriately modified in order to address the aforementioned challenges. The special scenarios, such as those in which the domain of the stream data is large, or the classes are rare, pose special challenges. Most of the well known techniques for streaming classification use space-efficient data structures for easily updatable models [13, 86]. Furthermore, these methods are explicitly designed to handle concept drift by making the models temporally adaptive, or by using different models over different regions of the data stream. Special scenarios or data types need dedicated methods in the streaming scenario. For example, the massive-domain scenario can be addressed [9] by incorporating the count-min data structure [33] as a synopsis structure within the training model. A specially difficult case is that of rare class learning, in which rare class instances may be mixed with occurrences of completely new classes. This problem can be considered a hybrid between classification and outlier detection. Nevertheless it is the most common case in the streaming domain, in applications such as intrusion detection. In these cases, some kinds of rare classes (intrusions) may already be known, whereas other rare classes may correspond to previously unseen threats. A book on data streams, containing extensive discussions on key topics in the area, may be found in [4]. The different variations of the streaming classification problem are addressed in detail in Chapter 9.

1.3.1.2 The Big Data Framework

While streaming algorithms work under the assumption that the data is too large to be stored explicitly, the big data framework leverages advances in storage technology in order to actually store the data and process it. However, as the subsequent discussion will show, even if the data can be explicitly stored, it is often not easy to process and extract insights from it.

In the simplest case, the data is stored on disk on a single machine, and it is desirable to scale up the approach with disk-efficient algorithms. While many methods such as the nearest neighbor classifier and associative classifiers can be scaled up with more efficient subroutines, other methods such as decision trees and SVMs require dedicated methods for scaling up. Some examples of scalable decision tree methods include *SLIQ* [48], *BOAT* [42], and *RainForest* [43]. Some early parallel implementations of decision trees include the *SPRINT* method [82]. Typically, scalable decision tree methods can be performed in one of two ways. Methods such as *RainForest* increase scalability by storing attribute-wise summaries of the training data. These summaries are sufficient for performing single-attribute splits efficiently. Methods such as *BOAT* use a combination of bootstrapped samples, in order to yield a decision tree, which is very close to the accuracy that one would have obtained by using the complete data.

An example of a scalable SVM method is *SVMLight* [53]. This approach focusses on the fact that the quadratic optimization problem in SVM is computationally intensive. The idea is to always

optimize only a small working set of variables while keeping the others fixed. This working set is selected by using a steepest descent criterion. This optimizes the advantage gained from using a particular subset of attributes. Another strategy used is to discard training examples, which do not have any impact on the margin of the classifiers. Training examples that are away from the decision boundary, and on its "correct" side, have no impact on the margin of the classifier, even if they are removed. Other methods such as *SVMPerf* [54] reformulate the SVM optimization to reduce the number of slack variables, and increase the number of constraints. A cutting plane approach, which works with a small subset of constraints at a time, is used in order to solve the resulting optimization problem effectively.

Further challenges arise for extremely large data sets. This is because an increasing size of the data implies that a distributed file system must be used in order to store it, and distributed processing techniques are required in order to ensure sufficient scalability. The challenge here is that if large segments of the data are available on different machines, it is often too expensive to shuffle the data across different machines in order to extract integrated insights from it. Thus, as in all distributed infrastructures, it is desirable to exchange intermediate insights, so as to minimize communication costs. For an application programmer, this can sometimes create challenges in terms of keeping track of where different parts of the data are stored, and the precise ordering of communications in order to minimize the costs.

In this context, Google's *MapReduce* framework [37] provides an effective method for analysis of large amounts of data, especially when the nature of the computations involve linearly computable statistical functions over the elements of the data streams. One desirable aspect of this framework is that it abstracts out the precise details of where different parts of the data are stored to the application programmer. As stated in [37]: "*The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.*" Many classification algorithms such as k-means are naturally linear in terms of their scalability with the size of the data. A primer on the *MapReduce* framework implementation on *Apache Hadoop* may be found in [87]. The key idea here is to use a *Map* function in order to distribute the work across the different machines, and then provide an automated way to shuffle out much smaller data in (key,value) pairs containing intermediate results. The *Reduce* function is then applied to the aggregated results from the *Map* step in order to obtain the final results.

Google's original *MapReduce* framework was designed for analyzing large amounts of Web logs, and more specifically deriving linearly computable statistics from the logs. It has been shown [44] that a declarative framework is particularly useful in many *MapReduce* applications, and that many existing classification algorithms can be generalized to the *MapReduce* framework. A proper choice of the algorithm to adapt to the *MapReduce* framework is crucial, since the framework is particularly effective for linear computations. It should be pointed out that the major attraction of the *MapReduce* framework is its ability to provide application programmers with a cleaner abstraction, which is independent of very specific run-time details of the distributed system. It should not, however, be assumed that such a system is somehow inherently superior to existing methods for distributed parallelization from an *effectiveness* or *flexibility* perspective, especially if an application programmer is willing to design such details from scratch. A detailed discussion of classification algorithms for big data is provided in Chapter 10.

1.3.2 Text Classification

One of the most common data types used in the context of classification is that of text data. Text data is ubiquitous, especially because of its popularity, both on the Web and in social networks. While a text document can be treated as a string of words, it is more commonly used as a bag-of-words, in which the ordering information between words is not used. This representation of

text is much closer to multidimensional data. However, the standard methods for multidimensional classification often need to be modified for text.

The main challenge with text classification is that the data is extremely high dimensional and sparse. A typical text lexicon may be of a size of a hundred thousand words, but a document may typically contain far fewer words. Thus, most of the attribute values are zero, and the frequencies are relatively small. Many common words may be very noisy and not very discriminative for the classification process. Therefore, the problems of feature selection and representation are particularly important in text classification.

Not all classification methods are equally popular for text data. For example, rule-based methods, the Bayes method, and SVM classifiers tend to be more popular than other classifiers. Some rule-based classifiers such as *RIPPER* [26] were originally designed for text classification. Neural methods and instance-based methods are also sometimes used. A popular instance-based method used for text classification is Rocchio's method [56, 74]. Instance-based methods are also sometimes used with centroid-based classification, where frequency-truncated centroids of class-specific clusters are used, instead of the original documents for the k-nearest neighbor approach. This generally provides better accuracy, because the centroid of a small closely related set of documents is often a more stable representation of that data locality than any single document. This is especially true because of the sparse nature of text data, in which two related documents may often have only a small number of words in common.

Many classifiers such as decision trees, which are popularly used in other data domains, are not quite as popular for text data. The reason for this is that decision trees use a strict hierarchical partitioning of the data. Therefore, the features at the higher levels of the tree are implicitly given greater importance than other features. In a text collection containing hundreds of thousands of features (words), a single word usually tells us very little about the class label. Furthermore, a decision tree will typically partition the data space with a very small number of splits. This is a problem, when this value is orders of magnitude less than the underlying data dimensionality. Of course, decision trees in text are not very balanced either, because of the fact that a given word is contained only in a small subset of the documents. Consider the case where a split corresponds to presence or absence of a word. Because of the imbalanced nature of the tree, most paths from the root to leaves will correspond to word-absence decisions, and a very small number (less than 5 to 10) word-presence decisions. Clearly, this will lead to poor classification, especially in cases where word-absence does not convey much information, and a modest number of word presence decisions are required. Univariate decision trees do not work very well for very high dimensional data sets, because of disproportionate importance to some features, and a corresponding inability to effectively leverage all the available features. It is possible to improve the effectiveness of decision trees for text classification by using multivariate splits, though this can be rather expensive.

The standard classification methods, which are used for the text domain, also need to be suitably modified. This is because of the high dimensional and sparse nature of the text domain. For example, text has a dedicated model, known as the multinomial Bayes model, which is different from the standard Bernoulli model [12]. The Bernoulli model treats the presence and absence of a word in a text document in a symmetric way. However, in a given text document, only a small fraction of the lexicon size is present in it. The absence of a word is usually far less informative than the presence of a word. The symmetric treatment of word presence and word absence can sometimes be detrimental to the effectiveness of a Bayes classifier in the text domain. In order to achieve this goal, the multinomial Bayes model is used, which uses the frequency of word presence in a document, but ignores non-occurrence.

In the context of SVM classifiers, scalability is important, because such classifiers scale poorly both with number of training documents and data dimensionality (lexicon size). Furthermore, the sparsity of text (i.e., few non-zero feature values) should be used to improve the training efficiency. This is because the training model in an SVM classifier is constructed using a constrained quadratic optimization problem, which has as many constraints as the number of data points. This is rather

large, and it directly results in an increased size of the corresponding Lagrangian relaxation. In the case of kernel SVM, the space-requirements for the kernel matrix could also scale quadratically with the number of data points. A few methods such as *SVMLight* [53] address this issue by carefully breaking down the problem into smaller subproblems, and optimizing only a few variables at a time. Other methods such as *SVMPerf* [54] also leverage the sparsity of the text domain. The *SVMPerf* method scales as $O(n \cdot s)$, where s is proportional to the average number of non-zero feature values per training document.

Text classification often needs to be performed in scenarios, where it is accompanied by linked data. The links between documents are typically inherited from domains such as the Web and social networks. In such cases, the links contain useful information, which should be leveraged in the classification process. A number of techniques have recently been designed to utilize such side information in the classification process. Detailed surveys on text classification may be found in [12, 78]. The problem of text classification is discussed in detail in Chapter 11 of this book.

1.3.3 Multimedia Classification

With the increasing popularity of social media sites, multimedia data has also become increasingly popular. In particular sites such as *Flickr* or *Youtube* allow users to upload their photos or videos at these sites. In such cases, it is desirable to perform classification of either portions or all of either a photograph or a video. In these cases, rich meta-data may also be available, which can facilitate more effective data classification. The issue of data representation is a particularly important one for multimedia data, because poor representations have a large semantic gap, which creates challenges for the classification process. The combination of text with multimedia data in order to create more effective classification models has been discussed in [8]. Many methods such as semi-supervised learning and transfer learning can also be used in order to improve the effectiveness of the data classification process. Multimedia data poses unique challenges, both in terms of data representation, and information fusion. Methods for multimedia data classification are discussed in [60]. A detailed discussion of methods for multimedia data classification is provided in Chapter 12.

1.3.4 Time Series and Sequence Data Classification

Both of these data types are temporal data types in which the attributes are of two types. The first type is the contextual attribute (time), and the second attribute, which corresponds to the time series value, is the behavioral attribute. The main difference between time series and sequence data is that time series data is continuous, whereas sequence data is discrete. Nevertheless, this difference is quite significant, because it changes the nature of the commonly used models in the two scenarios.

Time series data is popular in many applications such as sensor networks, and medical informatics, in which it is desirable to use large volumes of streaming time series data in order to perform the classification. Two kinds of classification are possible with time-series data:

- *Classifying specific time-instants:* These correspond to specific events that can be inferred at particular instants of the data stream. In these cases, the labels are associated with instants in time, and the behavior of one or more time series are used in order to classify these instants. For example, the detection of significant events in real-time applications can be an important application in this scenario.

- *Classifying part or whole series:* In these cases, the class labels are associated with portions or all of the series, and these are used for classification. For example, an ECG time-series will show characteristic shapes for specific diagnostic criteria for diseases.

Both of these scenarios are equally important from the perspective of analytical inferences in a wide variety of scenarios. Furthermore, these scenarios are also relevant to the case of sequence data. Sequence data arises frequently in biological, Web log mining, and system analysis applications. The discrete nature of the underlying data necessitates the use of methods that are quite different from the case of continuous time series data. For example, in the case of discrete sequences, the nature of the distance functions and modeling methodologies are quite different than those in time-series data.

A brief survey of time-series and sequence classification methods may be found in [91]. A detailed discussion on time-series data classification is provided in Chapter 13, and that of sequence data classification methods is provided in Chapter 14. While the two areas are clearly connected, there are significant differences between these two topics, so as to merit separate topical treatment.

1.3.5 Network Data Classification

Network data is quite popular in Web and social networks applications in which a variety of different scenarios for node classification arise. In most of these scenarios, the class labels are associated with nodes in the underlying network. In many cases, the labels are known only for a subset of the nodes. It is desired to use the known subset of labels in order to make predictions about nodes for which the labels are unknown. This problem is also referred to as *collective classification*. In this problem, the key assumption is that of *homophily*. This implies that edges imply similarity relationships between nodes. It is assumed that the labels vary smoothly over neighboring nodes. A variety of methods such as Bayes methods and spectral methods have been generalized to the problem of collective classification. In cases where content information is available at the nodes, the effectiveness of classification can be improved even further. A detailed survey on collective classification methods may be found in [6].

A different form of graph classification is one in which many small graphs exist, and labels are associated with individual graphs. Such cases arise commonly in the case of chemical and biological data, and are discussed in detail in [7]. The focus of the chapter in this book is on very large graphs and social networks because of their recent popularity. A detailed discussion of network classification methods is provided in Chapter 15 of this book.

1.3.6 Uncertain Data Classification

Many forms of data collection are uncertain in nature. For example, data collected with the use of sensors is often uncertain. Furthermore, in cases when data perturbation techniques are used, the data becomes uncertain. In some cases, statistical methods are used in order to infer parts of the data. An example is the case of link inference in network data. Uncertainty can play an important role in the classification of uncertain data. For example, if an attribute is known to be uncertain, its contribution to the training model can be de-emphasized, with respect to an attribute that has deterministic attributes.

The problem of uncertain data classification was first studied in [5]. In these methods, the uncertainty in the attributes is used as a first-class variable in order to improve the effectiveness of the classification process. This is because the relative importance of different features depends not only on their correlation with the class variable, but also the uncertainty inherent in them. Clearly, when the values of an attribute are more uncertain, it is less desirable to use them for the classification process. This is achieved in [5] with the use of a density-based transform that accounts for the varying level of uncertainty of attributes. Subsequently, many other methods have been proposed to account for the uncertainty in the attributes during the classification process. A detailed description of uncertain data classification methods is provided in Chapter 16.

1.4 Variations on Data Classification

Many natural variations of the data classification problem correspond to either small variations of the standard classification problem or are enhancements of classification with the use of additional data. The key variations of the classification problem are those of rare-class learning and distance function learning. Enhancements of the data classification problem make use of meta-algorithms, more data in methods such as transfer learning and co-training, active learning, and human intervention in visual learning. In addition, the topic of model evaluation is an important one in the context of data classification. This is because the issue of model evaluation is important for the design of effective classification meta-algorithms. In the following section, we will discuss the different variations of the classification problem.

1.4.1 Rare Class Learning

Rare class learning is an important variation of the classification problem, and is closely related to outlier analysis [1]. In fact, it can be considered a supervised variation of the outlier detection problem. In rare class learning, the distribution of the classes is highly imbalanced in the data, and it is typically more important to correctly determine the positive class. For example, consider the case where it is desirable to classify patients into malignant and normal categories. In such cases, the majority of patients may be normal, though it is typically much more costly to misclassify a truly malignant patient (false negative). Thus, false negatives are more *costly* than false positives. The problem is closely related to cost-sensitive learning, since the misclassification of different classes has different classes. The major difference with the standard classification problem is that the objective function of the problem needs to be modified with costs. This provides several avenues that can be used in order to effectively solve this problem:

- *Example Weighting:* In this case, the examples are weighted differently, depending upon their cost of misclassification. This leads to minor changes in most classification algorithms, which are relatively simple to implement. For example, in an SVM classifier, the objective function needs to be appropriately weighted with costs, whereas in a decision tree, the quantification of the split criterion needs to weight the examples with costs. In a nearest neighbor classifier, the k nearest neighbors are appropriately weighted while determining the class with the largest presence.
- *Example Re-sampling:* In this case, the examples are appropriately re-sampled, so that rare classes are over-sampled, whereas the normal classes are under-sampled. A standard classifier is applied to the re-sampled data without any modification. From a technical perspective, this approach is equivalent to example weighting. However, from a computational perspective, such an approach has the advantage that the newly re-sampled data has much smaller size. This is because most of the examples in the data correspond to the normal class, which is drastically under-sampled, whereas the rare class is typically only mildly over-sampled.

Many variations of the rare class detection problem are possible, in which either examples of a single class are available, or the normal class is contaminated with rare class examples. A survey of algorithms for rare class learning may be found in [25]. This topic is discussed in detail in Chapter 17.

1.4.2 Distance Function Learning

Distance function learning is an important problem that is closely related to data classification. In this problem it is desirable to relate pairs of data instances to a distance value with the use of ei-

ther supervised or unsupervised methods [3]. For example, consider the case of an image collection, in which the similarity is defined on the basis of a user-centered semantic criterion. In such a case, the use of standard distance functions such as the Euclidian metric may not reflect the semantic similarities between two images well, because they are based on human perception, and may even vary from collection to collection. Thus, the best way to address this issue is to explicitly incorporate human feedback into the learning process. Typically, this feedback is incorporated either in terms of pairs of images with explicit distance values, or in terms of rankings of different images to a given target image. Such an approach can be used for a variety of different data domains. This is the training data that is used for learning purposes. A detailed survey of distance function learning methods is provided in [92]. The topic of distance function learning is discussed in detail in Chapter 18.

1.4.3 Ensemble Learning for Data Classification

A meta-algorithm is a classification method that re-uses one or more currently existing classification algorithm by applying either multiple models for robustness, or combining the results of the same algorithm with different parts of the data. The general goal of the algorithm is to obtain more robust results by combining the results from multiple training models either sequentially or independently. The overall error of a classification model depends upon the bias and variance, in addition to the intrinsic noise present in the data. The bias of a classifier depends upon the fact that the decision boundary of a particular model may not correspond to the true decision boundary. For example, the training data may not have a linear decision boundary, but an SVM classifier will assume a linear decision boundary. The variance is based on the random variations in the particular training data set. Smaller training data sets will have larger variance. Different forms of ensemble analysis attempt to reduce this bias and variance. The reader is referred to [84] for an excellent discussion on bias and variance.

Meta-algorithms are used commonly in many data mining problems such as clustering and outlier analysis [1, 2] in order to obtain more accurate results from different data mining problems. The area of classification is the richest one from the perspective of meta-algorithms, because of its crisp evaluation criteria and relative ease in combining the results of different algorithms. Some examples of popular meta-algorithms are as follows:

- *Boosting:* Boosting [40] is a common technique used in classification. The idea is to focus on successively difficult portions of the data set in order to create models that can classify the data points in these portions more accurately, and then use the ensemble scores over all the components. A hold-out approach is used in order to determine the incorrectly classified instances for each portion of the data set. Thus, the idea is to sequentially determine better classifiers for more difficult portions of the data, and then combine the results in order to obtain a meta-classifier, which works well on all parts of the data.

- *Bagging:* Bagging [24] is an approach that works with random data samples, and combines the results from the models constructed using different samples. The training examples for each classifier are selected by sampling with replacement. These are referred to as *bootstrap* samples. This approach has often been shown to provide superior results in certain scenarios, though this is not always the case. This approach is not effective for reducing the bias, but can reduce the variance, because of the specific random aspects of the training data.

- *Random Forests:* Random forests [23] are a method that use sets of decision trees on either splits with randomly generated vectors, or random subsets of the training data, and compute the score as a function of these different components. Typically, the random vectors are generated from a fixed probability distribution. Therefore, random forests can be created by either random *split* selection, or random *input* selection. Random forests are closely related

to bagging, and in fact bagging with decision trees can be considered a special case of random forests, in terms of how the sample is selected (bootstrapping). In the case of random forests, it is also possible to create the trees in a lazy way, which is tailored to the particular test instance at hand.

- *Model Averaging and Combination:* This is one of the most common models used in ensemble analysis. In fact, the random forest method discussed above is a special case of this idea. In the context of the classification problem, many Bayesian methods [34] exist for the model combination process. The use of different models ensures that the error caused by the bias of a particular classifier does not dominate the classification results.

- *Stacking:* Methods such as stacking [90] also combine different models in a variety of ways, such as using a second-level classifier in order to perform the combination. The output of different first-level classifiers is used to create a new feature representation for the second level classifier. These first level classifiers may be chosen in a variety of ways, such as using different bagged classifiers, or by using different training models. In order to avoid overfitting, the training data needs to be divided into two subsets for the first and second level classifiers.

- *Bucket of Models:* In this approach [94] a "hold-out" portion of the data set is used in order to decide the most appropriate model. The most appropriate model is one in which the highest accuracy is achieved in the held out data set. In essence, this approach can be viewed as a competition or bake-off contest between the different models.

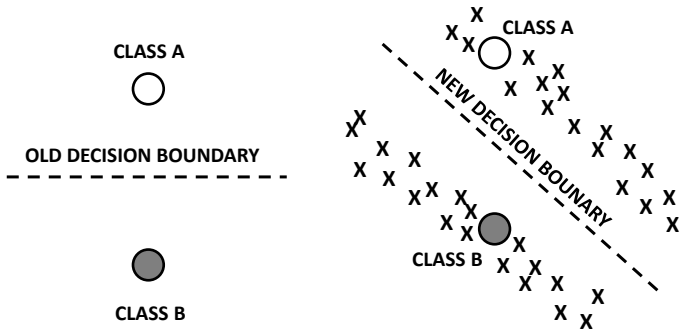
The area of meta-algorithms in classification is very rich, and different variations may work better in different scenarios. An overview of different meta-algorithms in classification is provided in Chapter 19.

1.4.4 Enhancing Classification Methods with Additional Data

In this class of methods, *additional labeled or unlabeled data* is used to enhance classification. Both these methods are used when there is a direct paucity of the underlying training data. In the case of transfer learning, additional training (labeled) data from a different domain or problem is used to supervise the classification process. On the other hand, in the case of semi-supervised learning, unlabeled data is used to enhance the classification process. These methods are briefly described in this section.

1.4.4.1 Semi-Supervised Learning

Semi-supervised learning methods improve the effectiveness of learning methods with the use of *unlabeled* data, when only a small amount of labeled data is available. The main difference between semi-supervised learning and transfer learning methods is that *unlabeled* data with the same features is used in the former, whereas external labeled data (possibly from a different source) is used in the latter. A key question arises as to why unlabeled data should improve the effectiveness of classification in any way, when it does not provide any additional labeling knowledge. The reason for this is that unlabeled data provides a good idea of the manifolds in which the data is embedded, as well as the density structure of the data in terms of the clusters and sparse regions. The key assumption is that the classification labels exhibit a smooth variation over different parts of the manifold structure of the underlying data. This manifold structure can be used to determine feature correlations, and joint feature distributions, which are very helpful for classification. The semi-supervised setting is also sometimes referred to as the *transductive* setting, when the test instances must be specified together with the training instances. Some problem settings such as collective classification of network data are naturally transductive.



(a) only labeled examples (b) labeled and unlabeled examples

FIGURE 1.4: Impact of unsupervised examples on classification process.

The motivation of semi-supervised learning is that knowledge of the dense regions in the space and correlated regions of the space are helpful for classification. Consider the two-class example illustrated in Figure 1.4(a), in which only a single training example is available for each class. In such a case, the decision boundary between the two classes is the straight line perpendicular to the one joining the two classes. However, suppose that some additional unsupervised examples are available, as illustrated in Figure 1.4(b). These unsupervised examples are denoted by 'x'. In such a case, the decision boundary changes from Figure 1.4(a). The major assumption here is that the classes vary *less* in dense regions of the training data, because of the *smoothness* assumption. As a result, even though the added examples do not have labels, they contribute significantly to improvements in classification accuracy.

In this example, the *correlations* between feature values were estimated with unlabeled training data. This has an intuitive interpretation in the context of text data, where *joint* feature distributions can be estimated with unlabeled data. For example, consider a scenario, where training data is available about predicting whether a document is the "*politics*" category. It may be possible that the word "*Obama*" (or some of the less common words) may not occur in any of the (small number of) training documents. However, the word "*Obama*" may often co-occur with many features of the "*politics*" category in the unlabeled instances. Thus, the unlabeled instances can be used to learn the relevance of these less common features to the classification process, especially when the amount of available training data is small.

Similarly, when the data is clustered, each cluster in the data is likely to predominantly contain data records of one class or the other. The identification of these clusters only requires unsupervised data rather than labeled data. Once the clusters have been identified from unlabeled data, only a small number of labeled examples are required in order to determine confidently which label corresponds to which cluster. Therefore, when a test example is classified, its clustering structure provides critical information for its classification process, even when a smaller number of labeled examples are available. It has been argued in [67] that the accuracy of the approach may increase exponentially with the number of labeled examples, as long as the assumption of smoothness in label structure variation holds true. Of course, in real life, this may not be true. Nevertheless, it has been shown repeatedly in many domains that the addition of unlabeled data provides significant advantages for the classification process. An argument for the effectiveness of semi-supervised learning that uses the spectral clustering structure of the data may be found in [18]. In some domains such as graph data, semi-supervised learning is the only way in which classification may be performed. This is because a given node may have very few neighbors of a specific class.

Semi-supervised methods are implemented in a wide variety of ways. Some of these methods directly try to label the unlabeled data in order to increase the size of the training set. The idea is

to incrementally add the most confidently predicted label to the training data. This is referred to as *self training*. Such methods have the downside that they run the risk of overfitting. For example, when an unlabeled example is added to the training data with a specific label, the label might be incorrect because of the specific characteristics of the feature space, or the classifier. This might result in further propagation of the errors. The results can be quite severe in many scenarios.

Therefore, semi-supervised methods need to be carefully designed in order to avoid overfitting. An example of such a method is *co-training* [21], which partitions the attribute set into two subsets, on which classifier models are independently constructed. The top label predictions of one classifier are used to augment the training data of the other, and vice-versa. Specifically, the steps of co-training are as follows:

1. Divide the feature space into two disjoint subsets f_1 and f_2.
2. Train two independent classifier models $\mathcal{M}_1$ and $\mathcal{M}_2$, which use the disjoint feature sets f_1 and f_2, respectively.
3. Add the unlabeled instance with the most confidently predicted label from $\mathcal{M}_1$ to the training data for $\mathcal{M}_2$ and vice-versa.
4. Repeat all the above steps.

Since the two classifiers are independently constructed on different feature sets, such an approach avoids overfitting. The partitioning of the feature set into f_1 and f_2 can be performed in a variety of ways. While it is possible to perform random partitioning of features, it is generally advisable to leverage redundancy in the feature set to construct f_1 and f_2. Specifically, each feature set f_i should be picked so that the features in f_j (for $j \neq i$) are redundant with respect to it. Therefore, each feature set represents a different view of the data, which is sufficient for classification. This ensures that the "confident" labels assigned to the other classifier are of high quality. At the same time, overfitting is avoided to at least some degree, because of the disjoint nature of the feature set used by the two classifiers. Typically, an erroneously assigned class label will be more easily detected by the disjoint feature set of the other classifier, which was not used to assign the erroneous label. For a test instance, each of the classifiers is used to make a prediction, and the combination score from the two classifiers may be used. For example, if the naive Bayes method is used as the base classifier, then the product of the two classifier scores may be used.

The aforementioned methods are generic meta-algorithms for semi-supervised leaning. It is also possible to design variations of existing classification algorithms such as the EM-method, or transductive SVM classifiers. EM-based methods [67] are very popular for text data. These methods attempt to model the joint probability distributions of the features and the labels with the use of partially supervised clustering methods. This allows the estimation of the conditional probabilities in the Bayes classifier to be treated as missing data, for which the EM-algorithm is very effective. This approach shows a connection between the partially supervised clustering and partially supervised classification problems. The results show that partially supervised classification is most effective, when the clusters in the data correspond to the different classes. In transductive SVMs, the labels of the unlabeled examples are also treated as integer decision variables. The SVM formulation is modified in order to determine the maximum margin SVM, with the best possible label assignment of unlabeled examples. Surveys on semi-supervised methods may be found in [29, 96]. Semi-supervised methods are discussed in Chapter 20.

1.4.4.2 Transfer Learning

As in the case of semi-supervised learning, transfer learning methods are used when there is a direct paucity of the underlying training data. However, the difference from semi-supervised learning is that, instead of using unlabeled data, labeled data from a different domain is used to enhance

the learning process. For example, consider the case of learning the class label of Chinese documents, where enough training data is not available about the documents. However, similar English documents may be available that contain training labels. In such cases, the knowledge in training data for the English documents can be *transferred* to the Chinese document scenario for more effective classification. Typically, this process requires some kind of "bridge" in order to relate the Chinese documents to the English documents. An example of such a "bridge" could be pairs of similar Chinese and English documents though many other models are possible. In many cases, a small amount of auxiliary training data in the form of labeled Chinese training documents may also be available in order to further enhance the effectiveness of the transfer process. This general principle can also be applied to cross-category or cross-domain scenarios where knowledge from one classification category is used to enhance the learning of another category [71], or the knowledge from one data domain (e.g., text) is used to enhance the learning of another data domain (e.g., images) [36, 70, 71, 95]. Broadly speaking, transfer learning methods fall into one of the following four categories:

1. *Instance-Based Transfer:* In this case, the feature spaces of the two domains are highly overlapping; even the class labels may be the same. Therefore, it is possible to transfer knowledge from one domain to the other by simply re-weighting the features.

2. *Feature-Based Transfer:* In this case, there may be some overlaps among the features, but a significant portion of the feature space may be different. Often, the goal is to perform a transformation of each feature set into a new low dimensional space, which can be shared across related tasks.

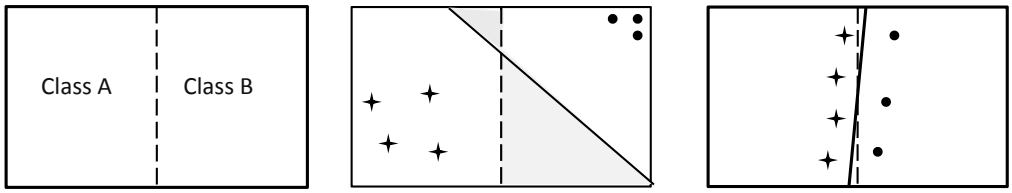
3. *Parameter-Based Transfer:* In this case, the motivation is that a good training model has typically learned a lot of structure. Therefore, if two tasks are related, then the structure can be transferred to learn the target task.

4. *Relational-Transfer Learning:* The idea here is that if two domains are related, they may share some similarity relations among objects. These similarity relations can be used for transfer learning across domains.

The major challenge in such transfer learning methods is that *negative* transfer can be caused in some cases when the side information used is very noisy or irrelevant to the learning process. Therefore, it is critical to use the transfer learning process in a careful and judicious way in order to truly improve the quality of the results. A survey on transfer learning methods may be found in [68], and a detailed discussion on this topic may be found in Chapter 21.

1.4.5 Incorporating Human Feedback

A different way of enhancing the classification process is to use some form of human supervision in order to improve the effectiveness of the classification process. Two forms of human feedback are quite popular, and they correspond to active learning and visual learning, respectively. These forms of feedback are different in that the former is typically focussed on *label acquisition* with human feedback, so as to enhance the training data. The latter is focussed on either visually creating a training model, or by visually performing the classification in a diagnostic way. Nevertheless, both forms of incorporating human feedback work with the assumption that the active input of a user can provide better knowledge for the classification process. It should be pointed out that the feedback in active learning may not always come from a user. Rather a generic concept of an *oracle* (such as *Amazon Mechanical Turk*) may be available for the feedback.



(a) Class Separation

(b) Random Sample with SVM Classifier

(c) Active Sample with SVM Classifier

FIGURE 1.5: Motivation of active learning.

1.4.5.1 Active Learning

Most classification algorithms assume that the learner is a passive recipient of the data set, which is then used to create the training model. Thus, the data collection phase is cleanly separated out from modeling, and is generally not addressed in the context of model construction. However, data collection is costly, and is often the (cost) bottleneck for many classification algorithms. In active learning, the goal is to collect more labels *during the learning process* in order to improve the effectiveness of the classification process at a low cost. Therefore, the learning process and data collection process are tightly integrated with one another and enhance each other. Typically, the classification is performed in an interactive way with the learner providing well-chosen examples to the user, for which the user may then provide labels.

For example, consider the two-class example of Figure 1.5. Here, we have a very simple division of the data into two classes, which is shown by a vertical dotted line, as illustrated in Figure 1.5(a). The two classes here are labeled by A and B. Consider the case where it is possible to query only seven examples for the two different classes. In this case, it is quite possible that the small number of allowed samples may result in a training data which is unrepresentative of the true separation between the two classes. Consider the case when an SVM classifier is used in order to construct a model. In Figure 1.5(b), we have shown a total of seven samples randomly chosen from the underlying data. Because of the inherent noisiness in the process of picking a small number of samples, an SVM classifier will be unable to accurately divide the data space. This is shown in Figure 1.5(b), where a portion of the data space is incorrectly classified, because of the error of modeling the SVM classifier. In Figure 1.5(c), we have shown an example of a well chosen set of seven instances along the decision boundary of the two classes. In this case, the SVM classifier is able to accurately model the decision regions between the two classes. This is because of the careful choice of the instances chosen by the active learning process. An important point to note is that it is particularly useful to sample instances that can clearly demarcate the decision boundary between the two classes.

In general, the examples are typically chosen for which the learner has the greatest level of uncertainty based on the current training knowledge and labels. This choice evidently provides the greatest additional information to the learner in cases where the greatest uncertainty exists about the current label. As in the case of semi-supervised learning, the assumption is that unlabeled data is copious, but acquiring labels for it is expensive. Therefore, by using the help of the learner in choosing the appropriate examples to label, it is possible to greatly reduce the effort involved in the classification process. Active learning algorithms often use support vector machines, because the latter are particularly good at determining the boundaries between the different classes. Examples that lie on these boundaries are good candidates to query the user, because the greatest level of uncertainty exists for these examples. Numerous criteria exist for training example choice in active

learning algorithms, most of which try to either reduce the uncertainty in classification or reduce the error associated with the classification process. Some examples of criteria that are commonly used in order to query the learner are as follows:

- *Uncertainty Sampling:* In this case, the learner queries the user for labels of examples, for which the greatest level of uncertainty exists about its correct output [45].
- *Query by Committee (QBC):* In this case, the learner queries the user for labels of examples in which a committee of classifiers have the greatest disagreement. Clearly, this is another indirect way to ensure that examples with the greatest uncertainty are queries [81].
- *Greatest Model Change:* In this case, the learner queries the user for labels of examples, which cause the greatest level of change from the current model. The goal here is to learn new knowledge that is not currently incorporated in the model [27].
- *Greatest Error Reduction:* In this case, the learner queries the user for labels of examples, which causes the greatest reduction of error in the current example [28].
- *Greatest Variance Reduction:* In this case, the learner queries the user for examples, which result in greatest reduction in output variance [28]. This is actually similar to the previous case, since the variance is a component of the total error.
- *Representativeness:* In this case, the learner queries the user for labels that are most *representative* of the underlying data. Typically, this approach combines one of the aforementioned criteria (such as uncertainty sampling or QBC) with a representativeness model such as a density-based method in order to perform the classification [80].

These different kinds of models may work well in different kinds of scenarios. Another form of active learning queries the data *vertically*. In other words, instead of examples, it is learned which *attributes* to collect, so as to minimize the error at a given cost level [62]. A survey on active learning methods may be found in [79]. The topic of active learning is discussed in detail in Chapter 22.

1.4.5.2 Visual Learning

The goal of visual learning is typically related to, but different from, active learning. While active learning collects examples from the user, visual learning takes the help of the user in the classification process in either creating the training model or using the model for classification of a particular test instance. This help can be received by learner in two ways:

- *Visual feedback in construction of training models:* In this case, the feedback of the user may be utilized in constructing the best training model. Since the user may often have important domain knowledge, this visual feedback may often result in more effective models. For example, while constructing a decision tree classifier, a user may provide important feedback about the split points at various levels of the tree. At the same time, a visual representation of the current decision tree may be provided to the user in order to facilitate more intuitive choices. An example of a decision tree that is constructed with the use of visual methods is discussed in [17].
- *Diagnostic classification of individual test instances:* In this case, the feedback is provided by the user during classification of test instances, rather than during the process of construction of the model. The goal of this method is different, in that it enables a better understanding of the *causality* of a test instance belonging to a particular class. An example of a visual method for diagnostic classification, which uses exploratory and visual analysis of test instances, is provided in [11]. Such a method is not suitable for classifying large numbers of test instances in batch. It is typically suitable for understanding the classification behavior of a small number of carefully selected test instances.

A general discussion on visual data mining methods is found in [10, 47, 49, 55, 83]. A detailed discussion of methods for visual classification is provided in Chapter 23.

1.4.6 Evaluating Classification Algorithms

An important issue in data classification is that of *evaluation* of classification algorithms. How do we know how well a classification algorithm is performing? There are two primary issues that arise in the evaluation process:

- *Methodology used for evaluation:* Classification algorithms require a training phase and a testing phase, in which the test examples are cleanly separated from the training data. However, in order to *evaluate* an algorithm, some of the labeled examples must be removed from the training data, and the model is constructed on these examples. The problem here is that the removal of labeled examples implicitly underestimates the power of the classifier, as it relates to the set of labels already available. Therefore, how should this removal from the labeled examples be performed so as to not impact the learner accuracy too much?

  Various strategies are possible, such as *hold out*, *bootstrapping*, and *cross-validation*, of which the first is the simplest to implement, and the last provides the greatest accuracy of implementation. In the hold-out approach, a fixed percentage of the training examples are "held out," and not used in the training. These examples are then used for evaluation. Since only a subset of the training data is used, the evaluation tends to be pessimistic with the approach. Some variations use stratified sampling, in which each class is sampled independently in proportion. This ensures that random variations of class frequency between training and test examples are removed.

  In bootstrapping, sampling with replacement is used for creating the training examples. The most typical scenario is that n examples are sampled with replacement, as a result of which the fraction of examples not sampled is equal to $(1-1/n)^n \approx 1/e$, where e is the basis of the natural logarithm. The class accuracy is then evaluated as a weighted combination of the accuracy a_1 on the unsampled (test) examples, and the accuracy a_2 on the full labeled data. The full accuracy A is given by:

$$A = (1-1/e) \cdot a_1 + (1/e) \cdot a_2 \tag{1.23}$$

  This procedure is repeated over multiple bootstrap samples and the final accuracy is reported. Note that the component a_2 tends to be highly optimistic, as a result of which the bootstrapping approach produces highly optimistic estimates. It is most appropriate for smaller data sets.

  In cross-validation, the training data is divided into a set of k disjoint subsets. One of the k subsets is used for testing, whereas the other $(k-1)$ subsets are used for training. This process is repeated by using each of the k subsets as the test set, and the error is averaged over all possibilities. This has the advantage that all examples in the labeled data have an opportunity to be treated as test examples. Furthermore, when k is large, the training data size approaches the full labeled data. Therefore, such an approach approximates the accuracy of the model using the entire labeled data well. A special case is "leave-one-out" cross-validation, where k is chosen to be equal to the number of training examples, and therefore each test segment contains exactly one example. This is, however, expensive to implement.

- *Quantification of accuracy:* This issue deals with the problem of quantifying the error of a classification algorithm. At first sight, it would seem that it is most beneficial to use a measure such as the absolute classification accuracy, which directly computes the fraction of examples that are correctly classified. However, this may not always be appropriate in

all cases. For example, some algorithms may have much lower variance across different data sets, and may therefore be more desirable. In this context, an important issue that arises is that of the statistical significance of the results, when a particular classifier performs better than another on a data set. Another issue is that the output of a classification algorithm may either be presented as a discrete label for the test instance, or a numerical score, which represents the propensity of the test instance to belong to a specific class. For the case where it is presented as a discrete label, the accuracy is the most appropriate score.

In some cases, the output is presented as a numerical score, especially when the class is rare. In such cases, the Precision-Recall or ROC curves may need to be used for the purposes of classification evaluation. This is particularly important in imbalanced and rare-class scenarios. Even when the output is presented as a binary label, the evaluation methodology is different for the rare class scenario. In the rare class scenario, the misclassification of the rare class is typically much more costly than that of the normal class. In such cases, cost sensitive variations of evaluation models may need to be used for greater robustness. For example, the cost sensitive accuracy weights the rare class and normal class examples differently in the evaluation.

An excellent review of evaluation of classification algorithms may be found in [52]. A discussion of evaluation of classification algorithms is provided in Chapter 24.

1.5 Discussion and Conclusions

The problem of data classification has been widely studied in the data mining and machine learning literature. A wide variety of methods are available for data classification, such as decision trees, nearest neighbor methods, rule-based methods, neural networks, or SVM classifiers. Different classifiers may work more effectively with different kinds of data sets and application scenarios.

The data classification problem is relevant in the context of a variety of data types, such as text, multimedia, network data, time-series and sequence data. A new form of data is probabilistic data, in which the underlying data is uncertain and may require a different type of processing in order to use the uncertainty as a first-class variable. Different kinds of data may have different kinds of representations and contextual dependencies. This requires the design of methods that are well tailored to the different data types.

The classification problem has numerous variations that allow the use of either additional training data, or human intervention in order to improve the underlying results. In many cases, meta-algorithms may be used to significantly improve the quality of the underlying results.

The issue of scalability is an important one in the context of data classification. This is because data sets continue to increase in size, as data collection technologies have improved over time. Many data sets are collected continuously, and this has lead to large volumes of data streams. Even in cases where very large volumes of data are collected, big data technologies need to be designed for the classification process. This area of research is still in its infancy, and is rapidly evolving over time.

Bibliography

[1] C. Aggarwal. *Outlier Analysis*, Springer, 2013.

[2] C. Aggarwal and C. Reddy. *Data Clustering: Algorithms and Applications*, CRC Press, 2013.

[3] C. Aggarwal. Towards Systematic Design of Distance Functions in Data Mining Applications, *ACM KDD Conference*, 2003.

[4] C. Aggarwal. *Data Streams: Models and Algorithms*, Springer, 2007.

[5] C. Aggarwal. On Density-based Transforms for Uncertain Data Mining, *ICDE Conference*, 2007.

[6] C. Aggarwal. *Social Network Data Analytics*, Springer, Chapter 5, 2011.

[7] C. Aggarwal and H. Wang. *Managing and Mining Graph Data*, Springer, 2010.

[8] C. Aggarwal and C. Zhai. *Mining Text Data*, Chapter 11, Springer, 2012.

[9] C. Aggarwal and P. Yu. On Classification of High Cardinality Data Streams. *SDM Conference*, 2010.

[10] C. Aggarwal. Towards Effective and Interpretable Data Mining by Visual Interaction, *ACM SIGKDD Explorations*, 2002.

[11] C. Aggarwal. Toward exploratory test-instance-centered diagnosis in high-dimensional classification, *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1001–1015, 2007.

[12] C. Aggarwal and C. Zhai. A survey of text classification algorithms, *Mining Text Data*, Springer, 2012.

[13] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for classification of evolving data streams. In *IEEE TKDE Journal*, 2006.

[14] C. Aggarwal. On Effective Classification of Strings with Wavelets, *ACM KDD Conference*, 2002.

[15] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms, *Machine Learning*, 6(1):37–66, 1991.

[16] D. Aha. Lazy learning: Special issue editorial. *Artificial Intelligence Review*, 11:7–10, 1997.

[17] M. Ankerst, M. Ester, and H.-P. Kriegel. Towards an Effective Cooperation of the User and the Computer for Classification, *ACM KDD Conference*, 2000.

[18] M. Belkin and P. Niyogi. Semi-supervised learning on Riemannian manifolds, *Machine Learning*, 56:209–239, 2004.

[19] C. Bishop. *Neural Networks for Pattern Recognition*, Oxford University Press, 1996.

[20] C. Bishop. *Pattern Recognition and Machine Learning*, Springer, 2007.

[21] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 92–100, 1998.

[22] L. Breiman. *Classification and regression trees*. CRC Press, 1993.

[23] L. Breiman. Random forests. *Journal Machine Learning Archive*, 45(1):5–32, 2001.

[24] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[25] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: Special Issue on Learning from Imbalanced Data Sets, *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[26] W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization, *ACM Transactions on Information Systems*, 17(2):141–173, 1999.

[27] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning, *Machine Learning*, 5(2):201–221, 1994.

[28] D. Cohn, Z. Ghahramani and M. Jordan. Active learning with statistical models, *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[29] O. Chapelle, B. Scholkopf, and A. Zien. *Semi-supervised learning. Vol. 2*, *Cambridge*: MIT Press, 2006.

[30] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, 2000.

[31] P. Clark and T. Niblett. The CN2 Induction algorithm, *Machine Learning*, 3(4):261–283, 1989.

[32] B. Clarke. Bayes model averaging and stacking when model approximation error cannot be ignored, *Journal of Machine Learning Research*, pages 683–712, 2003.

[33] G. Cormode and S. Muthukrishnan, An improved data-stream summary: The count-min sketch and its applications, *Journal of Algorithms*, 55(1), (2005), pp. 58–75.

[34] P. Domingos. Bayesian Averaging of Classifiers and the Overfitting Problem. *ICML Conference*, 2000.

[35] I. Dagan, Y. Karov, and D. Roth. Mistake-driven Learning in Text Categorization, *Proceedings of EMNLP*, 1997.

[36] W. Dai, Y. Chen, G.-R. Xue, Q. Yang, and Y. Yu. Translated learning: Transfer learning across different feature spaces. *Proceedings of Advances in Neural Information Processing Systems*, 2008.

[37] J. Dean and S. Ghemawat. MapReduce: A flexible data processing tool, *Communication of the ACM*, 53:72–77, 2010.

[38] P. Domingos and M. J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997.

[39] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, Wiley, 2001.

[40] Y. Freund, R. Schapire. A decision-theoretic generalization of online learning and application to boosting, *Lecture Notes in Computer Science*, 904:23–37, 1995.

[41] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[42] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. BOAT: Optimistic Decision Tree Construction, *ACM SIGMOD Conference*, 1999.

[43] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest—a framework for fast decision tree construction of large datasets, *VLDB Conference*, pages 416–427, 1998.

[44] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, T. Yuanyuan, and S. Vaithyanathan. SystemML: Declarative Machine Learning with MapReduce, *ICDE Conference*, 2011.

[45] D. Lewis and J. Catlett. Heterogeneous Uncertainty Sampling for Supervised Learning, *ICML Conference*, 1994.

[46] L. Hamel. *Knowledge Discovery with Support Vector Machines*, Wiley, 2009.

[47] C. Hansen and C. Johnson. *Visualization Handbook*, Academic Press, 2004.

[48] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A Fast Scalable Classifier for Data Mining, *EDBT Conference*, 1996.

[49] M. C. F. de Oliveira and H. Levkowitz. Visual Data Mining: A Survey, *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394. 2003.

[50] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2013.

[51] S. Haykin. *Neural Networks and Learning Machines*, Prentice Hall, 2008.

[52] N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective*, Cambridge University Press, 2011.

[53] T. Joachims. Making Large scale SVMs practical, *Advances in Kernel Methods, Support Vector Learning*, pages 169–184, Cambridge: *MIT Press*, 1998.

[54] T. Joachims. Training Linear SVMs in Linear Time, *KDD*, pages 217–226, 2006.

[55] D. Keim. Information and visual data mining, *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[56] W. Lam and C. Y. Ho. Using a Generalized Instance Set for Automatic Text Categorization. *ACM SIGIR Conference*, 1998.

[57] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[58] B. Liu, W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining, *ACM KDD Conference*, 1998.

[59] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*, Springer, 1998.

[60] R. Mayer. *Multimedia Learning*, Cambridge University Press, 2009.

[61] G. J. McLachlan. *Discriminant analysis and statistical pattern recognition*, Wiley-Interscience, Vol. 544, 2004.

[62] P. Melville, M. Saar-Tsechansky, F. Provost, and R. Mooney. An Expected Utility Approach to Active Feature-Value Acquisition. *IEEE ICDM Conference*, 2005.

[63] T. Mitchell. *Machine Learning*, McGraw Hill, 1997.

[64] K. Murphy. *Machine Learning: A Probabilistic Perspective*, MIT Press, 2012.

[65] S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey, *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[66] H. T. Ng, W. Goh and K. Low. Feature Selection, Perceptron Learning, and a Usability Case Study for Text Categorization. *ACM SIGIR Conference*, 1997.

[67] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM, *Machine Learning*, 39(2–3):103–134, 2000.

[68] S. J. Pan and Q. Yang. A survey on transfer learning, *IEEE Transactons on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[69] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[70] G. Qi, C. Aggarwal, and T. Huang. Towards Semantic Knowledge Propagation from Text Corpus to Web Images, *WWW Conference*, 2011.

[71] G. Qi, C. Aggarwal, Y. Rui, Q. Tian, S. Chang, and T. Huang. Towards Cross-Category Knowledge Propagation for Learning Visual Concepts, *CVPR Conference*, 2011.

[72] J. Quinlan. *C4.5: Programs for Machine Learning*, Morgan-Kaufmann Publishers, 1993.

[73] J. R. Quinlan. Induction of decision trees, *Machine Learning*, 1(1):81–106, 1986.

[74] J. Rocchio. Relevance feedback information retrieval. *The Smart Retrieval System - Experiments in Automatic Document Processing*, G. Salton, Ed. Englewood Cliffs, Prentice Hall, NJ: pages 313–323, 1971.

[75] B. Scholkopf and A. J. Smola. Learning with Kernels: *Support Vector Machines, Regularization, Optimization, and Beyond*, Cambridge University Press, 2001.

[76] I. Steinwart and A. Christmann. *Support Vector Machines*, Springer, 2008.

[77] H. Schutze, D. Hull, and J. Pedersen. A Comparison of Classifiers and Document Representations for the Routing Problem. *ACM SIGIR Conference*, 1995.

[78] F. Sebastiani. Machine learning in automated text categorization, *ACM Computing Surveys*, 34(1):1–47, 2002.

[79] B. Settles. *Active Learning*, Morgan and Claypool, 2012.

[80] B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1069–1078, 2008.

[81] H. Seung, M. Opper, and H. Sompolinsky. Query by Committee. Fifth Annual Workshop on Computational Learning Theory, 1992.

[82] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classfier for data mining, *VLDB Conference*, pages 544–555, 1996.

[83] T. Soukop and I. Davidson. *Visual Data Mining: Techniques and Tools for Data Visualization*, Wiley, 2002.

[84] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson, 2005.

[85] V. Vapnik. *The Nature of Statistical Learning Theory*, Springer, New York, 1995.

[86] H. Wang, W. Fan, P. Yu, and J. Han. Mining Concept-Drifting Data Streams with Ensemble Classifiers, *KDD Conference*, 2003.

[87] T. White. *Hadoop: The Definitive Guide*. Yahoo! Press, 2011.

[88] E. Wiener, J. O. Pedersen, and A. S. Weigend. A neural network approach to topic spotting. *SDAIR*, pages 317–332, 1995.

[89] D. Wettschereck, D. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, *Artificial Intelligence Review*, 11(1–5):273–314, 1997.

[90] D. Wolpert. Stacked generalization, *Neural Networks*, 5(2):241–259, 1992.

[91] Z. Xing and J. Pei, and E. Keogh. A brief survey on sequence classification. *SIGKDD Explorations*, 12(1):40–48, 2010.

[92] L. Yang. Distance Metric Learning: A Comprehensive Survey, 2006. `http://www.cs.cmu.edu/~liuy/frame_survey_v2.pdf`

[93] M. J. Zaki and C. Aggarwal. XRules: A Structural Classifier for XML Data, *ACM KDD Conference*, 2003.

[94] B. Zenko. Is combining classifiers better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[95] Y. Zhu, S. J. Pan, Y. Chen, G.-R. Xue, Q. Yang, and Y. Yu. Heterogeneous Transfer Learning for Image Classification. *Special Track on AI and the Web, associated with The Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[96] X. Zhu and A. Goldberg. *Introduction to Semi-Supervised Learning*, Morgan and Claypool, 2009.