

Hang Li

# Machine Learning Methods

Translated by Lu Lin and  
Huanqiang Zeng



清华大学出版社  
TSINGHUA UNIVERSITY PRESS

Springer

# Machine Learning Methods

Hang Li

# Machine Learning Methods



清华大学出版社  
TSINGHUA UNIVERSITY PRESS

Springer

Hang Li  
Bytedance Technology  
Beijing, China

*Translated by*  
Lu Lin  
Huaqiao University  
Quanzhou, Fujian, China

Huanqiang Zeng  
Huaqiao University  
Quanzhou, Fujian, China

ISBN 978-981-99-3916-9

ISBN 978-981-99-3917-6 (eBook)

Jointly published with Tsinghua University Press  
The print edition is not for sale in China (Mainland). Customers from China (Mainland) please order the print book from: Tsinghua University Press.

Translation from the Chinese Simplified language edition: “统计学习方法 (第2版)” by Hang Li, © Tsinghua University Press 2019. Published by Tsinghua University Press. All Rights Reserved.

© Tsinghua University Press 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publishers, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publishers nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publishers remain neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.  
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721,  
Singapore

# Preface

This book is a popular machine learning textbook and reference book in China. The first edition was published in March 2012 under the title *Statistical Learning Methods* focusing on supervised learning, including perceptron,  $k$ -nearest neighbor method, naive Bayes method, decision tree, logistic regression, maximum entropy model, support vector machine, boosting method, EM algorithm, hidden Markov model, and conditional random field. The second edition was published in May 2019, with additional content on unsupervised learning, including the clustering method, singular value decomposition, principal component analysis, latent semantic analysis, probabilistic latent semantic analysis, Markov chain Monte Carlo method, latent Dirichlet assignment, and PageRank Algorithm. By the end of 2021, the two editions had been printed more than 30 times and sold more than 350,000 copies.

Many thanks go to Assistant Professor Lin Lu and Prof. Zeng Huanqiang of Huaqiao University for translating this book into English. It is renamed *Machine Learning Methods* and published by Springer Press. We would be honored if overseas readers could also benefit from this book.

This book can be used as a reference for teaching statistical machine learning and related courses. Its target readers also include undergraduates and postgraduates in artificial intelligence, machine learning, natural language processing, and other related majors. In this book, we are dedicated to introducing machine learning methods systematically and comprehensively. Regarding content selection, it focuses on the most important and commonly used methods. In the narrative style, each chapter describes a single method and is relatively independent and complete. Following a unified framework, the discussion of all methods remains systemic. The reader can read it from beginning to end or focus on some individual chapters. Complicated methods are described in simple terms with necessary proofs and examples, enabling beginners to grasp the basic content and essence of the methods and use them accurately. Relevant theories are briefly described as well. At the end of each chapter, exercises with references and relevant research trends are provided to meet readers' needs for further learning.

Currently, we are also adding content on deep learning and reinforcement learning to the existing version. The reprinted version is hoped to be translated into English in the future.

Finally, I gratefully acknowledge Dr. Chang Lanlan, the editor of Springer Publishing, Assistant Professor Lin Lu, and Prof. Zeng Huanqiang of Huaqiao University for their efforts and support.

Beijing, China

Hang Li

# Contents

<b>1</b>	<b>Introduction to Machine Learning and Supervised Learning</b>	<b>1</b>
1.1	Machine Learning .....	1
1.1.1	Characteristics of Machine Learning .....	1
1.1.2	The Object of Machine Learning .....	2
1.1.3	The Purpose of Machine Learning .....	2
1.1.4	Methods of Machine Learning .....	3
1.1.5	The Study of Machine Learning .....	3
1.1.6	The Importance of Machine Learning .....	4
1.2	Classification of Machine Learning .....	4
1.2.1	The Basic Classification .....	4
1.2.2	Classification by Model Types .....	11
1.2.3	Classification by Algorithm .....	12
1.2.4	Classification by Technique .....	13
1.3	Three Elements of Machine Learning Methods .....	15
1.3.1	Model .....	16
1.3.2	Strategy .....	17
1.3.3	Algorithm .....	20
1.4	Model Evaluation and Model Selection .....	20
1.4.1	Training Error and Test Error .....	20
1.4.2	Over-Fitting and Model Selection .....	21
1.5	Regularization and Cross-Validation .....	24
1.5.1	Regularization .....	24
1.5.2	Cross-Validation .....	25
1.6	Generalization Ability .....	26
1.6.1	Generalization Error .....	26
1.6.2	Generalization Error Bound .....	27
1.7	Generative Approach and Discriminative Model .....	29
1.8	Supervised Learning Application .....	31
1.8.1	Classification .....	31
1.8.2	Tagging .....	33

1.8.3	Regression .....	34
References .....		37
<b>2</b>	<b>Perceptron .....</b>	<b>39</b>
2.1	The Perceptron Model .....	39
2.2	Perceptron Learning Strategy .....	41
2.2.1	Linear Separability of the Dataset .....	41
2.2.2	Perceptron Learning Strategy .....	41
2.3	Perceptron Learning Algorithm .....	43
2.3.1	The Primal Form of the Perceptron Learning Algorithm .....	43
2.3.2	Convergence of the Algorithm .....	46
2.3.3	The Dual Form of the Perceptron Learning Algorithm .....	49
References .....		52
<b>3</b>	<b>K-Nearest Neighbor .....</b>	<b>55</b>
3.1	The <i>K</i> -Nearest Neighbor Algorithm .....	55
3.2	The <i>K</i> -Nearest Neighbor Model .....	56
3.2.1	Model .....	56
3.2.2	Distance Metrics .....	57
3.2.3	The Selection of <i>k</i> Value .....	58
3.2.4	Classification Decision Rule .....	59
3.3	Implementation of <i>K</i> -Nearest Neighbor: The <i>kd</i> -Tree .....	60
3.3.1	Constructing the <i>kd</i> -Tree .....	60
3.3.2	Searching for <i>kd</i> -Tree .....	62
References .....		65
<b>4</b>	<b>The Naïve Bayes Method .....</b>	<b>67</b>
4.1	The Learning and Classification of Naïve Bayes .....	67
4.1.1	Basic Methods .....	67
4.1.2	Implications of Posterior Probability Maximization .....	69
4.2	Parameter Estimation of the Naïve Bayes Method .....	70
4.2.1	Maximum Likelihood Estimation .....	70
4.2.2	Learning and Classification Algorithms .....	70
4.2.3	Bayesian Estimation .....	72
References .....		75
<b>5</b>	<b>Decision Tree .....</b>	<b>77</b>
5.1	Decision Tree Model and Learning .....	77
5.1.1	Decision Tree Model .....	77
5.1.2	Decision Tree and If-Then Rules .....	78
5.1.3	Decision Tree and Conditional Probability Distributions .....	79
5.1.4	Decision Tree Learning .....	79
5.2	Feature Selection .....	81

5.2.1	The Feature Selection Problem .....	81
5.2.2	Information Gain .....	83
5.2.3	Information Gain Ratio .....	87
5.3	Generation of Decision Tree .....	87
5.3.1	ID3 Algorithm .....	87
5.3.2	C4.5 Generation Algorithm .....	89
5.4	Pruning of Decision Tree .....	90
5.5	CART Algorithm .....	92
5.5.1	CART Generation .....	93
5.5.2	CART Pruning .....	98
References	.....	102
<b>6</b>	<b>Logistic Regression and Maximum Entropy Model</b> .....	103
6.1	Logistic Regression Model .....	103
6.1.1	Logistic Distribution .....	103
6.1.2	Binomial Logistic Regression Model .....	104
6.1.3	Model Parameter Estimation .....	106
6.1.4	Multi-nomial Logistic Regression .....	107
6.2	Maximum Entropy Model .....	107
6.2.1	Maximum Entropy Principle .....	107
6.2.2	Definition of Maximum Entropy Model .....	110
6.2.3	Learning of the Maximum Entropy Model .....	111
6.2.4	Maximum Likelihood Estimation .....	116
6.3	Optimization Algorithm of Model Learning .....	117
6.3.1	Improved Iterative Scaling .....	118
6.3.2	Quasi-Newton Method .....	122
References	.....	125
<b>7</b>	<b>Support Vector Machine</b> .....	127
7.1	Linear Support Vector Machine in the Linearly Separable Case and Hard Margin Maximization .....	128
7.1.1	Linear Support Vector Machine in the Linearly Separable Case .....	128
7.1.2	Function Margin and Geometric Margin .....	129
7.1.3	Maximum Margin .....	132
7.1.4	Dual Algorithm of Learning .....	137
7.2	Linear Support Vector Machine and Soft Margin Maximization .....	144
7.2.1	Linear Support Vector Machine .....	144
7.2.2	Dual Learning Algorithm .....	145
7.2.3	Support Vector .....	149
7.2.4	Hinge Loss Function .....	150
7.3	Non-Linear Support Vector Machine and Kernel Functions .....	153
7.3.1	Kernel Trick .....	153
7.3.2	Positive Definite Kernel .....	156
7.3.3	Commonly Used Kernel Functions .....	162

7.3.4	Nonlinear Support Vector Classifier .....	164
7.4	Sequential Minimal Optimization Algorithm .....	165
7.4.1	The Method of Solving Two-Variable Quadratic Programming .....	166
7.4.2	Selection Methods of Variables .....	170
7.4.3	SMO Algorithm .....	172
	References .....	177
<b>8</b>	<b>Boosting .....</b>	<b>179</b>
8.1	AdaBoost Algorithm .....	179
8.1.1	The Basic Idea of Boosting .....	179
8.1.2	AdaBoost Algorithm .....	180
8.1.3	AdaBoost Example .....	183
8.2	Training Error Analysis of AdaBoost Algorithm .....	185
8.3	Explanation of AdaBoost Algorithm .....	187
8.3.1	Forward Stepwise Algorithm .....	187
8.3.2	Forward Stepwise Algorithm and AdaBoost .....	189
8.4	Boosting Tree .....	191
8.4.1	Boosting Tree Model .....	191
8.4.2	Boosting Tree Algorithm .....	191
8.4.3	Gradient Boosting .....	196
	References .....	199
<b>9</b>	<b>EM Algorithm and Its Extensions .....</b>	<b>201</b>
9.1	Introduction of the EM Algorithm .....	201
9.1.1	EM Algorithm .....	201
9.1.2	Derivation of the EM Algorithm .....	205
9.1.3	Application of the EM Algorithm in Unsupervised Learning .....	208
9.2	The Convergence of the EM Algorithm .....	208
9.3	Application of the EM Algorithm in the Learning of the Gaussian Mixture Model .....	210
9.3.1	Gaussian Mixture Model .....	211
9.3.2	The EM Algorithm for Parameter Estimation of the Gaussian Mixture Model .....	211
9.4	Extensions of the EM Algorithm .....	214
9.4.1	The Maximization-Maximization Algorithm of $F$ -Function .....	214
9.4.2	GEM Algorithm .....	217
9.5	Summary .....	218
9.6	Further Reading .....	219
9.7	Exercises .....	220
	References .....	220

<b>10 Hidden Markov Model</b>	221
10.1 The Basic Concept of Hidden Markov Model	221
10.1.1 Definition of Hidden Markov Model	221
10.1.2 The Generation Process of the Observation Sequence	225
10.1.3 Three Basic Problems of the Hidden Markov Model	225
10.2 Probability Calculation Algorithms	226
10.2.1 Direct Calculation Method	226
10.2.2 Forward Algorithm	227
10.2.3 Backward Algorithm	230
10.2.4 Calculation of Some Probabilities and Expected Values	231
10.3 Learning Algorithms	233
10.3.1 Supervised Learning Methods	233
10.3.2 Baum-Welch Algorithm	234
10.3.3 Baum-Welch Model Parameter Estimation Formula	237
10.4 Prediction Algorithm	238
10.4.1 Approximation Algorithm	238
10.4.2 Viterbi Algorithm	239
References	245
<b>11 Conditional Random Field</b>	247
11.1 Probabilistic Undirected Graphical Model	247
11.1.1 Model Definition	247
11.1.2 Factorization of Probabilistic Undirected Graphical Model	250
11.2 The Definition and Forms of Conditional Random Field	251
11.2.1 The Definition of Conditional Random Field	251
11.2.2 The Parameterized Form of the Conditional Random Field	253
11.2.3 The Simplified Form of Conditional Random Field	254
11.2.4 The Matrix Form of the Conditional Random Field	256
11.3 The Probability Computation Problem of Conditional Random Field	258
11.3.1 Forward–Backward Algorithm	258
11.3.2 Probability Computation	259
11.3.3 The Computation of Expected Value	259
11.4 Learning Algorithms of Conditional Random Field	260
11.4.1 Improved Iterative Scaling	261
11.4.2 Quasi-Newton Method	264
11.5 The Prediction Algorithm of Conditional Random Field	266
References	271

<b>12</b>	<b>Summary of Supervised Learning Methods</b>	273
12.1	Application	273
12.2	Models	277
12.3	Learning Strategies	278
12.4	Learning Algorithms	280
<b>13</b>	<b>Introduction to Unsupervised Learning</b>	281
13.1	The Fundamentals of Unsupervised Learning	281
13.2	Basic Issues	282
13.2.1	Clustering	282
13.2.2	Dimensionality Reduction	283
13.2.3	Probability Model Estimation	284
13.3	Three Elements of Machine Learning	285
13.4	Unsupervised Learning Methods	286
13.4.1	Clustering	286
13.4.2	Dimensionality Reduction	287
13.4.3	Topic Modeling	288
13.4.4	Graph Analytics	289
	References	292
<b>14</b>	<b>Clustering</b>	293
14.1	Basic Concepts of Clustering	294
14.1.1	Similarity or Distance	294
14.1.2	Class or Cluster	297
14.1.3	Distance Between Classes	299
14.2	Hierarchical Clustering	300
14.3	k-means Clustering	302
14.3.1	Model	302
14.3.2	Strategy	303
14.3.3	Algorithm	304
14.3.4	Algorithm Characteristics	306
	References	309
<b>15</b>	<b>Singular Value Decomposition</b>	311
15.1	Introduction	311
15.2	Definition and Properties of Singular Value Decomposition	311
15.2.1	Definition and Theorem	311
15.2.2	Compact Singular Value Decomposition and Truncated Singular Value Decomposition	317
15.2.3	Geometry Interpretation	319
15.2.4	Main Properties	321
15.3	Computation of Singular Value Decomposition	323
15.4	Singular Value Decomposition and Matrix Approximation	327
15.4.1	Frobenius Norm	327
15.4.2	Optimal Approximation of the Matrix	328

15.4.3	The Outer Product Expansion of Matrix .....	331
References .....		336
<b>16</b>	<b>Principal Component Analysis .....</b>	<b>337</b>
16.1	Overall Principal Component Analysis .....	337
16.1.1	Basic Ideas .....	337
16.1.2	Definition and Derivation .....	340
16.1.3	Main Properties .....	342
16.1.4	The Number of Principal Components .....	347
16.1.5	The Overall Principal Components of Normalized Variables .....	351
16.2	Sample Principal Component Analysis .....	352
16.2.1	The Definition and Properties of the Sample Principal Components .....	352
16.2.2	Eigenvalue Decomposition Algorithm of Aorrelation Matrix .....	355
16.2.3	Singular Value Decomposition Algorithm for Data Matrix .....	358
References .....		364
<b>17</b>	<b>Latent Semantic Analysis .....</b>	<b>365</b>
17.1	Word Vector Space and Topic Vector Space .....	366
17.1.1	Word Vector Space .....	366
17.1.2	Topic Vector Space .....	368
17.2	Latent Semantic Analysis Algorithm .....	372
17.2.1	Matrix Singular Value Decomposition Algorithm .....	372
17.2.2	Examples .....	374
17.3	Non-negative Matrix Factorization Algorithm .....	377
17.3.1	Non-negative Matrix Factorization .....	378
17.3.2	Latent Semantic Analysis Model .....	379
17.3.3	Formalization of Non-negative Matrix Factorization .....	379
17.3.4	Algorithm .....	380
References .....		385
<b>18</b>	<b>Probabilistic Latent Semantic Analysis .....</b>	<b>387</b>
18.1	Probabilistic Latent Semantic Analysis Model .....	387
18.1.1	Basic Ideas .....	387
18.1.2	Generative Model .....	388
18.1.3	Co-occurrence Model .....	390
18.1.4	Model Properties .....	391
18.2	Algorithms for Probabilistic Latent Semantic Analysis .....	394
References .....		398

<b>19</b>	<b>Markov Chain Monte Carlo Method</b>	401
19.1	Monte Carlo Method	401
19.1.1	Random Sampling	402
19.1.2	Mathematical Expectation Estimate	403
19.1.3	Integral Computation	404
19.2	Markov Chain	406
19.2.1	Basic Definition	406
19.2.2	Discrete-Time Markov Chain	407
19.2.3	Continuous-Time Markov Chain	413
19.2.4	Properties of Markov Chain	414
19.3	Markov Chain Monte Carlo Method	419
19.3.1	Basic Ideas	419
19.3.2	Basic Steps	421
19.3.3	Markov Chain Monte Carlo Method and Machine Learning	421
19.4	Metropolis–Hastings Algorithm	422
19.4.1	Fundamental Concepts	423
19.4.2	Metropolis–Hastings Algorithm	426
19.4.3	The Single-Component Metropolis–Hastings Algorithm	427
19.5	Gibbs Sampling	428
19.5.1	Basic Principles	429
19.5.2	Gibbs Sampling Algorithm	430
19.5.3	Sampling Computation	432
	References	437
<b>20</b>	<b>Latent Dirichlet Allocation</b>	439
20.1	Dirichlet Distribution	440
20.1.1	Definition of Distribution	440
20.1.2	Conjugate Prior	443
20.2	Latent Dirichlet Allocation Model	445
20.2.1	Basic Ideas	445
20.2.2	Model Definition	446
20.2.3	Probability Graphical Model	448
20.2.4	The Changeability of Random Variable Sequences	449
20.2.5	Probability Formula	450
20.3	Gibbs Sampling Algorithm for LDA	451
20.3.1	Basic Ideas	452
20.3.2	Major Parts of Algorithm	453
20.3.3	Algorithm Post-processing	455
20.3.4	Algorithm	456
20.4	Variational EM Algorithm for LDA	458
20.4.1	Variational Reasoning	458
20.4.2	Variational EM Algorithm	460
20.4.3	Algorithm Derivation	461

20.4.4	Algorithm Summary .....	468
References .....		471
<b>21</b>	<b>The PageRank Algorithm .....</b>	<b>473</b>
21.1	The Definition of PageRank .....	473
21.1.1	Basic Ideas .....	473
21.1.2	The Directed Graph and Random Walk Model .....	475
21.1.3	The Basic Definition of PageRank .....	477
21.1.4	General Definition of PageRank .....	480
21.2	Computation of PageRank .....	482
21.2.1	Iterative Algorithm .....	482
21.2.2	Power Method .....	484
21.2.3	Algebraic Algorithms .....	489
References .....		491
<b>22</b>	<b>A Summary of Unsupervised Learning Methods .....</b>	<b>493</b>
22.1	The Relationships and Characteristics of Unsupervised Learning Methods .....	493
22.1.1	The Relationships Between Various Methods .....	493
22.1.2	Unsupervised Learning Methods .....	494
22.1.3	Basic Machine Learning Methods .....	495
22.2	The Relationships and Characteristics of Topic Models .....	496
References .....		497
<b>Appendix A: Gradient Descent .....</b>		<b>499</b>
<b>Appendix B: Newton Method and Quasi-Newton Method .....</b>		<b>501</b>
<b>Appendix C: Language Duality .....</b>		<b>509</b>
<b>Appendix D: Basic Subspaces of Matrix .....</b>		<b>513</b>
<b>Appendix E: The Definition of KL Divergence and the Properties of Dirichlet Distribution .....</b>		<b>517</b>
<b>Color Diagrams .....</b>		<b>521</b>
<b>Index .....</b>		<b>525</b>

# Chapter 1

## Introduction to Machine Learning and Supervised Learning

This chapter gives an overview of supervised learning methods. Supervised learning is the machine learning task of inferring a model from labeled training data. It is an important area in machine learning or machine learning.

In this chapter, some basic concepts of machine learning and supervised learning are introduced, which gives readers general knowledge of machine learning and supervised learning.

Section 1.1 of this chapter describes the definition, research objects and machine learning methods. Section 1.2 covers the classification of machine learning. In general, machine learning can be classified into supervised learning, unsupervised learning and reinforcement learning. Section 1.3 introduces the three elements of machine learning methods: models, strategies and algorithms; Sects. 1.4–1.7 introduce several essential concepts of supervised learning successively, including model evaluation and model selection, regularization and cross-validation, generalization ability of learning, generative model and discriminative model; Finally, Sect. 1.8 discusses the application of supervised learning, including classification, tagging and regression.

### 1.1 Machine Learning

#### 1.1.1 *Characteristics of Machine Learning*

Machine learning is a discipline in which computers build data-based probabilistic statistical models and use them to predict and analyze data. Machine learning is also known as statistical machine learning. Herbert A. Simon once defined “learning” as “a system’s ability to improve its performance by executing a process”. According to this view, statistical machine learning is machine learning in which computer systems improve system performance by utilizing data and statistical methods.

The main characteristics of machine learning are: (1) Machine learning is based on computer and network. (2) Machine learning is a data-driven discipline that uses data as its object of study. (3) The purpose of machine learning is to predict and analyze data. (4) Machine learning is method-centered, with machine learning methods building and applying models to prediction and analysis. (5) Machine learning is an interdisciplinary subject of probability, statistics, information, computation, optimization, computer science, etc. It has gradually formed its theoretical system and methodology along with its development.

### ***1.1.2 The Object of Machine Learning***

The object of machine learning research is data. The study starts from data, covers feature extraction, abstraction of data models and knowledge discovery of data, and returns to data analysis and prediction. As the object of machine learning, the data involved are diverse, including various digital, textual, image, video, and audio data, as well as their combinations on computers and the Internet.

The basic assumption of machine learning about data is that similar data have certain statistical regularity, which is the premise of machine learning. Here, similar data refers to the data with some common properties, such as English articles, Internet web pages, data in databases, etc. Because of their statistical regularity, they can be handled with probability and statistics. For example, random variables can describe the data feature, while probability distribution can depict the statistical regularity. In machine learning, data, which variables or groups of variables represent, can be further categorized according to the variable types (continuous and discrete variables) that denote them. This book focuses on the discussion of discrete variables and concerns data analysis and prediction with models constructed based on data. Issues like data observation and collection are not covered in this book.

### ***1.1.3 The Purpose of Machine Learning***

Machine learning is for predicting and analyzing data, especially the prediction and analysis of unknown new data. The prediction of data can make the computer more intelligent or boost the computer's performance; the analysis of data enables people to acquire new knowledge and make discoveries. The data prediction and analysis are achieved by constructing probability and statistical models. The general goal of machine learning is to analyze the types of models to be learned and ways to learn so that the model can accurately predict and analyze the data. It is also important to consider the possibility of maximizing learning efficiency as much as possible.

### 1.1.4 *Methods of Machine Learning*

Machine learning methods construct probability statistic models based upon data to predict and analyze the data. Machine learning consists of supervised learning, unsupervised learning, reinforcement learning, etc.

In this book, Chaps. 1 and 2 deal with supervised and unsupervised learning, the two primary machine learning approaches, respectively.

Machine learning methods can be summarized as follows: it starts from a given set of limited training data for learning and assumes that the data are independently and identically distributed; moreover, it is assumed that the model to be learned belongs to a set of functions, which is called hypothesis space; an optimal model is selected from a hypothesis space based on a specific evaluation criterion to make an optimal prediction of available training data and unknown test data; the selection of the optimal model is implemented by the algorithm. In this way, the machine learning method includes the hypothesis space of the model, the criteria of model selection and the algorithm of model learning, or model, strategy and algorithm for short. The factors mentioned above are addressed as the three elements of machine learning.

The steps to implement the machine learning method are as follows:

- (1) Acquire a limited set of training data;
- (2) Determine the hypothesis space that contains all possible models, that is, the set of learning models;
- (3) Determine the criteria for model selection, that is, the learning strategy;
- (4) Implement the algorithm for solving the optimal model, that is, the algorithm for learning;
- (5) Choose the optimal model by learning methods;
- (6) Use the learned optimal model to predict or analyze new data.

This chapter introduces supervised learning methods, mainly including classification, tagging and regression methods. These methods are being broadly applied in some domains, such as natural language processing, information retrieval, text data mining, etc.

### 1.1.5 *The Study of Machine Learning*

Machine learning research generally falls into three aspects: machine learning methods, machine learning theory and machine learning application, which have different focuses. The research of machine learning methods aims at developing new learning methods. The study of machine learning theory explores the effectiveness and efficiency of machine learning methods and the fundamental theoretical issues of machine learning, yet the research of machine learning applications is concerned with applying machine learning methods to practical problems for problem-solving.

### ***1.1.6 The Importance of Machine Learning***

The last two decades have witnessed a remarkable development of machine learning in theory and application. Many significant breakthroughs have been marked by the successful application of machine learning in artificial intelligence, pattern recognition, data mining, natural language processing, speech processing, computational vision, information retrieval, biological information, and many other computer application fields. It has become the core technology in these areas. Furthermore, it is convinced that machine learning will play an increasingly crucial role in future scientific development and technology applications.

The importance of machine learning in science and technology is mainly reflected in the following aspects:

- (1) Machine learning is an effective way of massive data processing. We live in an era of information explosion, where the processing and utilization of massive data is an inevitable demand. Data, in reality, is not only large in scale but also often uncertain. Machine learning tends to be the most powerful tool for processing this type of data.
- (2) Machine learning is an effective means of computer intelligentization. Intelligentization is the inevitable trend of computer development as well as the primary goal of computer technology research and development. In recent decades, research in artificial intelligence and other fields has proved that the application of machine learning in imitating human intelligence is the most effective means despite certain limitations.
- (3) Machine learning is a crucial component of the development of computer science. It can be considered that computer science is composed of three dimensions: system, computation, and information. Machine learning mainly belongs to the information dimension and plays a central role.

## **1.2 Classification of Machine Learning**

Machine learning is a wide-ranging, diverse, and widely-used field, and there is no (at least not yet) single unified body of theory covering everything. The following describes the classification of machine learning methods from different perspectives.

### ***1.2.1 The Basic Classification***

Machine learning is generally categorized into three major groups: supervised learning, unsupervised learning and reinforcement learning. Sometimes it also includes semi-supervised learning and active learning.

- (1) **Supervised learning**

Supervised learning is a machine learning method in which the prediction models are acquired from labeled data. The labeled data represent the relationship between input and output, and the prediction model generates the corresponding output for a given input. The essence of supervised learning is to learn the statistical law of input–output mapping.

#### (A) Input space, feature space and output space

In supervised learning, the set of all possible input and output values is called input space and output space, respectively. The input and output space can be a set of finite elements or the entire Euclidean space. Input and output spaces can be the same or different, but the output space is usually much smaller than the input space.

Each specific input is an instance, which is usually represented by a feature vector. At this time, the space where all feature vectors exist is referred to as feature space. Each dimension of the feature space corresponds to a feature. Sometimes the input space and the feature space are assumed to be the same space and are not distinguished; sometimes, the input space and the feature space are assumed to be different spaces, and the instance is mapped from the input space to the feature space. The model is defined in the feature space.

In supervised learning, input and output are the values of random variables defined in input (feature) space and output space. Input and output variables are represented in capital letters. Conventionally, the input variable is written as  $X$ , and the output variable is written as  $Y$ . The values of the input and output variables are expressed in lowercase letters; that is to say, the value of the input variable is written as  $x$ , and the value of the output variable is written as  $y$ . Variables can be scalar or vector, both of which are represented by the same type of letters. Unless otherwise stated, the vectors in this book are all column vectors. The feature vector of the input instance  $x$  is denoted as:

$$x = (x^{(1)}, x^{(2)}, \dots, x^{(i)}, \dots, x^{(n)})^T$$

$x^{(i)}$  represents the  $i$ -th feature of  $x$ . Note that  $x^{(i)}$  is different from  $x_i$ . In this book,  $x_i$  is generally used to represent the  $i$ -th variable among multiple input variables, namely

$$x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)})^T$$

Supervised learning is about learning a model from a set of training data and making predictions on test data. The training data is composed of input (or feature vector) and output pairs. The training set is usually expressed as:

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

The test data also consists of input and output pairs. The input and output pairs are also called samples or sample points.

The input variable  $X$  and the output variable  $Y$  have different types, which can be continuous or discrete. People provide different names for prediction tasks according to the different types of input and output variables. The prediction problem where the input and output are both continuous variables is called a regression problem. The prediction problem where the output variables are a finite number of discrete variables is called a classification problem. The prediction problem where both the input variable and the output variable are a sequence of variables is named the tagging problem.

#### (B) Joint probability distribution

Supervised learning assumes that the input and output random variables  $X$  and  $Y$  follow the joint probability distribution  $P(X, Y)$ , which represents the distribution function or distribution density function. Note that in the learning process, it is assumed that this joint probability distribution does exist. However, the specific definition of the joint probability distribution is unknown to the learning system. The training and test data are regarded as independently and identically distributed according to the joint probability distribution  $P(X, Y)$ . Machine learning assumes that the data has specific statistical laws. The joint probability distribution of  $X$  and  $Y$  is the basic assumption of supervised learning about the data.

#### (C) Hypothetical space

The purpose of supervised learning is to learn a mapping from input to output, which the model represents. In other words, the learning aims to find the best model. The model belongs to the set of mappings from the input space to the output space, and this set is the hypothesis space. The determination of the hypothesis space implies the determination of the scope of learning.

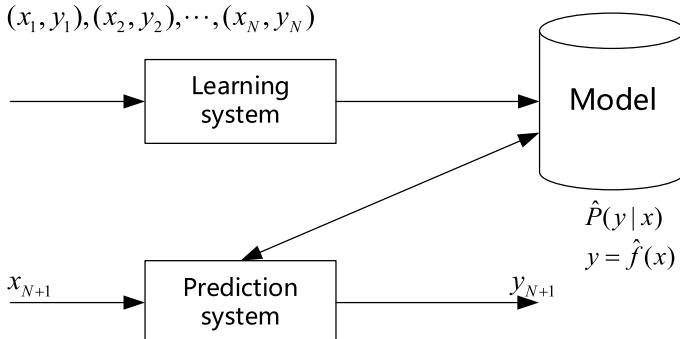
The supervised learning model can be a probabilistic model or a non-probabilistic model, represented by the conditional probability distribution  $P(Y|X)$  or decision function  $Y = f(X)$ , depending on the specific learning method. When making corresponding output predictions for specific inputs, it's written as  $P(y|x)$  or  $y = f(x)$ .

#### (D) Formalization of the problem

Supervised learning utilizes the training dataset to learn a model and then applies the model to make predictions on the test set. Since the labeled training dataset is needed in this process, and the labeled training dataset is often given manually, it is called supervised learning. There are two processes in supervised learning, learning and prediction, completed by the learning and prediction systems. These processes are described in Fig. 1.1.

First, a training dataset is given

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$



**Fig. 1.1** Supervised learning

In this set, the  $T, (x_i, y_i), i = 1, 2, \dots, N$ , is denoted as sample or sample point.  $x_i \in \chi \subseteq R^n$  is the input observation value, also called input or instance, and  $y_i \in y$  is the output observation value, also known as output.

Supervised learning is divided into two processes: learning and prediction, completed by the learning system and the prediction system. In the learning process, the learning system uses a given training dataset to obtain a model through learning (or training), which is expressed as a conditional probability distribution  $\hat{P}(Y|X)$  or a decision function  $\hat{Y} = \hat{f}(X)$ . They both describe the mapping relationship between input and output random variables. In the prediction process, the prediction system gives the corresponding output  $y_{N+1}$  from the model  $y_{N+1} = \arg \max_y \hat{P}(y|x_{N+1})$  or  $y_{N+1} = \hat{f}(x_{N+1})$  for the input  $x_{N+1}$  in the given test set.

In supervised learning, it is hypothesized that training data and test data are independently and identically distributed according to the joint probability distribution  $P(X, Y)$ .

The learning system (namely the learning algorithm) tries to learn the model through the information brought by the samples  $(x_i, y_i)$  in the training dataset. Specifically, for input  $x_i$ , a specific model  $y = f(x)$  can generate an output  $f(x_i)$ , and the corresponding output in the training dataset is  $y_i$ . The difference between the training sample output  $y_i$  and the model output  $f(x_i)$  should be small enough to guarantee the excellent prediction ability of the model. The learning system selects the best model through continuous attempts to make a good enough prediction on the training dataset, along with the best spreading of the prediction of the unknown test dataset.

## (2) Unsupervised learning

Unsupervised learning refers to the machine learning problem of learning predictive models from unlabeled data. Unlabeled data is naturally obtained data, and the predictive model represents the data's category, conversion, or probability. The essence of unsupervised learning is to learn statistical laws or potential structures in the data.

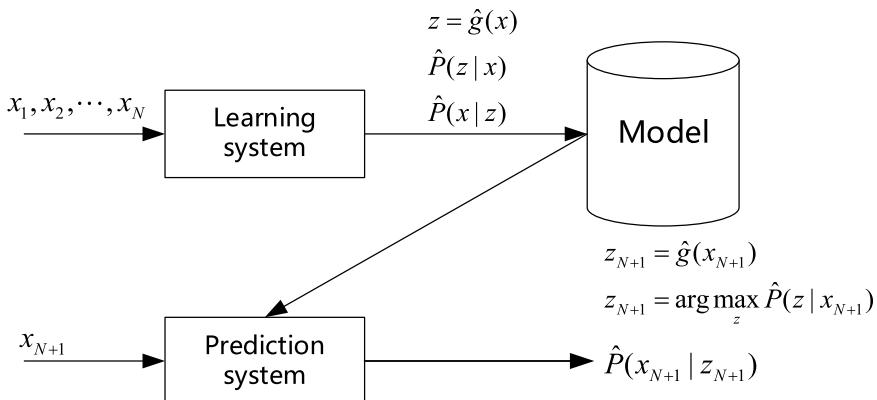
The set that all possible values of the input and the output of the model are respectively called input space and output space. The input and output space can be a collection of finite elements or Euclidean space. Each input is an instance represented by a feature vector. Each output is the result of the analysis of the input, represented by categories, conversion or probability of the input. The model can realize clustering, dimensionality reduction or probability estimation of data.

Suppose  $\chi$  is the input space and  $Z$  is the implicit structure space. The model to be learned can be expressed as a function  $z = g(x)$ , a conditional probability distribution  $P(z|x)$ , or a conditional probability distribution  $P(x|z)$ , where  $x \in \chi$  is the input and  $z \in Z$  is the output. The set containing all possible models is called the hypothesis space. Unsupervised learning aims to select the optimal model under the given evaluation standard from the hypothesis space.

Unsupervised learning usually learns or trains the model using a large amount of unlabeled data, and each sample is an instance. The training data is expressed as  $U = \{x_1, x_2, \dots, x_N\}$ , where  $x_i, i = 1, 2, \dots, N$  is the sample.

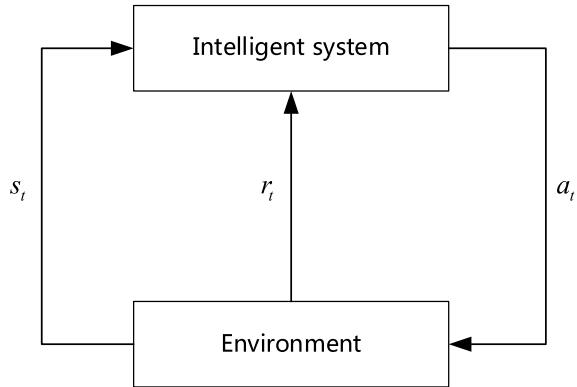
Unsupervised learning can be used to analyze existing data and predict future data. The analysis uses the learned model, i.e., function  $z = \hat{g}(x)$ , conditional probability distribution  $\hat{P}(z|x)$ , or conditional probability distribution  $\hat{P}(x|z)$ . While predicting, the process is similar to supervised learning. It is completed by the learning system and the prediction system, as shown in Fig. 1.2. In the learning process, the learning system learns from the training dataset and obtains an optimal model, expressed as the function  $z = \hat{g}(x)$ , conditional probability distribution  $\hat{P}(z|x)$  or conditional probability distribution  $\hat{P}(x|z)$ . In the prediction process, for the given input  $x_{N+1}$ , the prediction system generates the corresponding output  $z_{N+1}$  by the model  $z_{N+1} = \hat{g}(x_{N+1})$  or  $z_{N+1} = \arg \max_z \hat{P}(z|x_{N+1})$  to perform clustering or dimensionality reduction, and it's also possible to get the input probability  $\hat{P}(x_{N+1}|z_{N+1})$  from the model  $\hat{P}(x|z)$ , then perform probability estimation.

### (3) Reinforcement learning



**Fig. 1.2** Unsupervised learning

**Fig. 1.3** Interaction between intelligent system and environment



Reinforcement learning is a machine learning problem in which the intelligent system learns the optimal behavior strategy through continuous interaction with the environment. Assuming that the interaction between the intelligent system and environment is based on the Markov decision process, the intelligent system can observe the data sequence obtained from the interaction with the environment. The essence of reinforcement learning is to learn the optimal sequential decision.

The interaction between the intelligent system and the environment is shown in Fig. 1.3. At each step  $t$ , the intelligent system observes a state  $s_t$  and a reward  $r_t$  from the environment and takes an action  $a_t$ . According to the action selected by the intelligent system, the environment decides the state  $s_{t+1}$  and reward  $r_{t+1}$  in the next step  $t + 1$ . The strategy to be learned is expressed as an action taken in a given state. The goal of an intelligent system is not to maximize short-term but long-term cumulative rewards. In the process of reinforcement learning, the system constantly makes trial and error to achieve the purpose of learning the optimal strategy.

The Markov decision process of reinforcement learning is a random process on the state, reward, and action sequence. It consists of the five-tuple  $(S, A, P, r, \gamma)$ .

- $S$  is a set of finite state
- $A$  is a set of finite action
- $P$  is the transition probability function:

$$P(s'|s, a) = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- $r$  is the reward function:  $r(s, a) = E(r_{t+1} | s_t = s, a_t = a)$
- $\gamma$  is the discount factor:  $\gamma \in [0, 1]$

The Markov decision process has Markov property, and the next state only depends on the previous state and action, which is represented by the state transition probability function  $P(s'|s, a)$ . The next reward depends on the previous state and action, represented by the reward function  $r(s, a)$ .

Strategy  $\pi$  is defined as a function of action in a given state  $a = f(s)$  or a conditional probability distribution  $P(a|s)$ . Given a strategy  $\pi$ , the behavior of the intelligent system interacting with the environment is determined (either deterministically or stochastically).

The value function or state value function is defined as the mathematical expectation of the long-term cumulative reward of the strategy  $\pi$  starting from a certain state  $s$ :

$$v_\pi(s) = E_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s] \quad (1.1)$$

The action value function is defined as the mathematical expectation of the long-term cumulative reward of strategy  $\pi$  starting from a certain state  $s$  and an action  $a$ :

$$q_\pi(s, a) = E_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s, a_t = a] \quad (1.2)$$

The goal of reinforcement learning is to select the strategy  $\pi^*$  with the largest value function among all possible strategies. Actual learning often starts from specific strategies and continuously optimizes existing strategies. Here,  $\gamma$  indicates that future rewards will decay.

Reinforcement learning methods include policy-based and value-based methods, both of which belong to model-free methods. Besides model-free methods, there are model-based methods.

Model-based methods try to directly learn the model of the Markov decision process, including the transition probability function  $P(s'|s, a)$  and the reward function  $r(s, a)$ . In this case, the feedback of the environment can be predicted through the model, and the strategy  $\pi^*$  with the largest value function can be obtained.

Model-free and strategy-based methods do not directly learn the model but try to learn the optimal strategy  $\pi^*$ , expressed as a function  $a = f^*(s)$  or a conditional probability distribution  $P^*(a|s)$ , which can also achieve optimal environmental decisions. The learning usually starts with a specific strategy and proceeds by searching for better strategies.

Model-free and value-based methods do not directly learn the model as well. Instead, they attempt to solve the optimal value function, especially the optimal action value function  $q^*(s, a)$ . In this case, the optimal strategy can be learned indirectly, and the corresponding action in a given state according to the strategy can be made. Learning usually starts with a specific value function and proceeds by searching for a better value function.

#### (4) Semi-supervised learning and active learning

Semi-supervised learning refers to a machine learning problem using tagged and untagged data to learn a predictive model. There is a small amount of tagged data and a large amount of untagged data in general because the construction of tagged data tends to require labor with high cost, while the collection of untagged data doesn't need much cost. Semi-supervised learning aims to use the information in

untagged data to assist in tagging data and perform supervised learning, achieving better learning results at a lower cost.

Active learning is a machine learning problem in which the machine gives instances to teachers actively and constantly for tagging, and then uses the tagged data to learn the predictive model. General supervised learning uses the given tagged data, which tends to be obtained randomly, so that it can be regarded as “passive learning”. The goal of active learning is to find the most helpful instances of learning to teachers for tagging, achieving better learning results at a small tagging cost.

Semi-supervised learning and active learning are more similar to supervised learning.

### 1.2.2 *Classification by Model Types*

Machine learning or machine learning methods can be classified according to the model type.

#### (1) **The probabilistic model and the non-probabilistic model**

Machine learning models can be divided into probabilistic and non-probabilistic models (or deterministic models). In supervised learning, the probabilistic model takes the form of the conditional probability distribution  $P(y|x)$ , and the non-probabilistic model takes the functional form  $y = f(x)$ , where  $x$  is the input and  $y$  is the output. In unsupervised learning, the probabilistic model takes the form of conditional probability distribution  $P(z|x)$  or  $P(x|z)$ , and the non-probabilistic model takes functional form  $z = g(x)$ , where  $x$  is the input and  $z$  is the output.

The Decision Tree, Naive Bayes, Hidden Markov Model (HMM), Conditional Random Field (CRF), Probabilistic Latent Semantic Analysis (PLSA), Latent Dirichlet Assignment (LDA), and Gaussian Mixture Model (GMM) introduced in this book are all probabilistic models. Perceptron, Support Vector Machine (SVM), k-Nearest Neighbors (KNN), AdaBoost, k-means, Latent Semantic Analysis (LSA), and Neural Networks are non-probabilistic models. Logistic regression can be regarded as both probabilistic model and non-probabilistic model.

The conditional probability distribution  $P(y|x)$  and the function  $y = f(x)$  can be converted into each other (the conditional probability distribution  $P(z|x)$  and the function  $z = g(x)$  can be converted, too). Specifically, the function is obtained after the conditional probability distribution is maximized, and the conditional probability distribution is obtained after the function is normalized in return. Therefore, the difference between the probabilistic and non-probabilistic models is not in the mapping relationship between input and output, but in the internal structure of the model. The probabilistic model can usually be expressed as a joint probability distribution, where the variables represent input, output, hidden variables and even parameters. Non-probabilistic models may not have such a joint probability distribution.

The representative of the probabilistic model is the probabilistic graphical model. It is a probabilistic model in which a directed or undirected graph indicates the joint

**Fig. 1.4** Basic probability formula

addition rule:	$P(x) = \sum_y P(x, y)$
multiplication rule:	$P(x, y) = P(x)P(y x)$
Where x and y are random variables	

probability distribution. The joint probability distribution can be decomposed into the form of factor products according to the graph's structure. Bayesian networks, Markov Random Fields, and Conditional Random Fields are all probabilistic graphical models. Regardless of the complexity of the model, probabilistic reasoning can be carried out using the most basic addition and multiplication rules (see Fig. 1.4).

### (2) The linear model and the non-linear model

Machine learning models, especially non-probabilistic models, can be divided into linear and nonlinear models. If the function  $y = f(x)$  or  $z = g(x)$  is a linear function, then the model is called a linear model; otherwise, it's called a non-linear model.

The Perceptron, linear SVM, KNN, k-mean, and LSA introduced in this book are linear models. The Kernel Function SVM, AdaBoost, and Neural Networks are non-linear models.

Deep learning is the learning of complex Neural Networks, i.e., the learning of complex non-linear models.

### (3) The parametric model and the non-parametric model

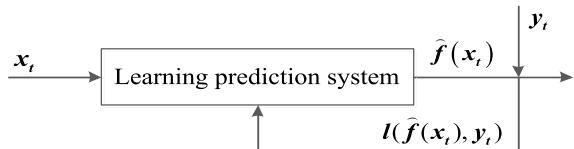
Machine learning models can be divided into parametric models and non-parametric models. Parametric models assume that the dimension of model parameters is fixed, and finite-dimensional parameters can completely describe the model. Non-parametric models assume that the dimension of model parameters is not fixed or infinite, increasing as training data increases.

The Perceptron, Naive Bayes, logistic regression, k-means, GMM, LSA, PLSA, and LDA introduced in this book are all parametric models. The Decision Tree, SVM, AdaBoost, and KNN are non-parametric models.

Parametric models are suitable for simple problems. However, non-parametric models can be more effective as real problems tend to be more complicated.

#### 1.2.3 Classification by Algorithm

According to the algorithm, machine learning can be divided into online learning and batch learning. Online learning refers to the machine learning that accepts one sample at a time, makes predictions, then learns the model and repeats the operation continuously. Correspondingly, batch learning accepts all data at once, learns the

**Fig. 1.5** Online learning

model, and then makes predictions. Some practical application scenarios require online learning. Such scenarios include where the data sequentially reach a point that they cannot be stored, and the system needs to make timely processing; or where the scale of data is too large to be processed all at once. Another case is where the mode of the data changes dynamically over time, requiring the algorithm to adapt to the new mode rapidly (not satisfying the independent identical distribution assumption).

Online learning can be either supervised or unsupervised, and reinforcement learning has the characteristics of online learning. Note that only online supervised learning is considered below.

Learning and prediction exist in a system that takes one input  $x_t$  at a time, generates the prediction  $\hat{f}(x_t)$  by using the existing model, and then obtains the corresponding feedback, i.e., the output  $y_t$  corresponding to the input. The system uses the loss function to compute the difference between them, updates the model, and repeats the above operations continuously (see Fig. 1.5).

The perceptron learning algorithm using stochastic gradient descent is an online learning algorithm.

Online learning is usually more complicated than batch learning, and it's hard for online learning to learn a model with higher prediction accuracy because the available data is limited in each model update.

#### 1.2.4 Classification by Technique

Machine learning methods can be classified according to the techniques used.

##### (1) Bayesian learning

Bayesian learning, also known as Bayesian inference, is a crucial method in statistics and machine learning. The main idea is to use Bayes theorem in the learning and inference of probabilistic models to compute the conditional probability of the model under given data conditions, i.e., the posterior probability, and to apply this principle to the model estimation and the data prediction. The representation of the model, unobserved elements and their parameters as variables, and the use of the prior distribution of the model are characteristics of Bayesian learning. Basic probability formulas are also used in Bayesian learning (see Fig. 1.4).

The naive Bayes and potential Dirichlet distributions discussed in this book belong to Bayesian learning.

Assume that the random variable  $D$  represents the data, and the random variable  $\theta$  represents the model parameters. According to Bayes theorem, the posterior probability  $P(\theta|D)$  can be computed with the following formula:

$$P(\theta|D) = \frac{P(\theta)P(D|\theta)}{P(D)} \quad (1.3)$$

where  $P(\theta)$  is the prior probability and  $P(D|\theta)$  is the likelihood function.

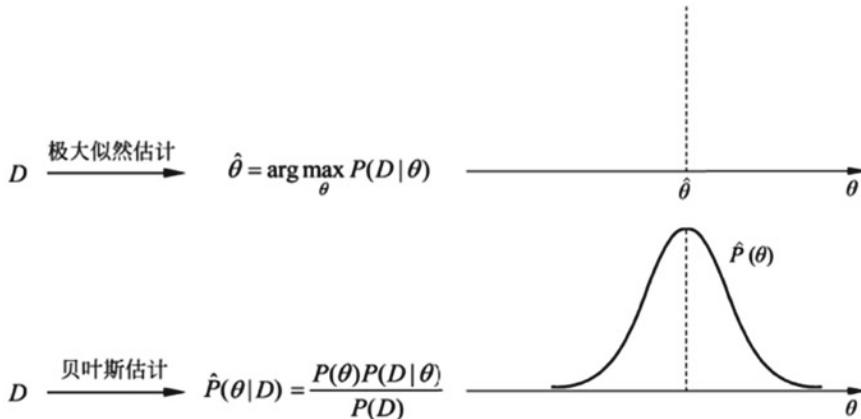
When the model is estimated, the entire posterior probability distribution  $P(\theta|D)$  is estimated. The model with the maximum posterior probability is usually taken if a model is required.

When forecasting, compute the expected value of the data to the posterior probability distribution:

$$P(x|D) = \int P(x|\theta, D)P(\theta|D)d\theta \quad (1.4)$$

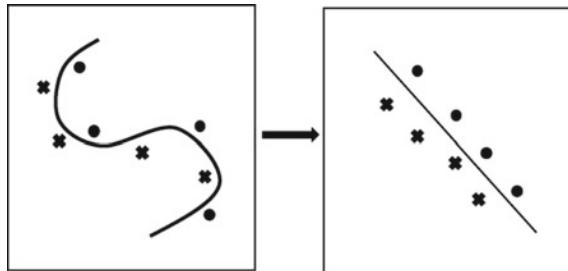
where  $x$  is the new sample.

Bayesian Estimation is quite different from Maximum Likelihood Estimation (MLE) in thought, representing the different understanding of Bayesian school and frequency school in statistics. In fact, the two can be simply connected. Assuming that the prior distribution is uniform, take the largest posterior probability, then the MLE can be obtained from the Bayesian Estimation. Figure 1.6 shows the comparison between Bayesian Estimation and MLE.



**Fig. 1.6** Bayesian estimation and maximum likelihood estimation

**Fig. 1.7** Mapping from the input space to the feature space



## (2) The Kernel method

The kernel method is a machine learning method that uses kernel functions to represent and learn nonlinear models and can be used for supervised and unsupervised learning. There are some learning methods of linear model based on similarity computation, more specifically, vector inner product computation. The Kernel method can extend them to learning nonlinear models, expanding their application scope.

The Kernel Function SVM introduced in this book, the kernel PCA and the kernel k-mean belong to the kernel method.

A straightforward approach to extending a linear model to a non-linear model is explicitly defining a mapping from the input space (low-dimensional space) to the feature space (high-dimensional space), where the inner product is computed. For example, the SVM transforms the linear inseparable problem in the input space into the linear separable problem in the feature space, as shown in Fig. 1.7. The technique of the kernel method is not to define the mapping explicitly, but to directly define the kernel function, that is, the inner product in the feature space after the mapping. This technique can simplify the computation and achieve the same effect.

Assuming that  $x_1$  and  $x_2$  are two instances (vectors) of the input space, the inner product is  $\langle x_1, x_2 \rangle$ . Suppose that the mapping from the input space to the feature space is  $\varphi$ , so the mapping of  $x_1$  and  $x_2$  in the feature space is  $\varphi(x_1)$  and  $\varphi(x_2)$ , the inner product is  $\langle \varphi(x_1), \varphi(x_2) \rangle$ . The kernel method directly defines the kernel function  $K(x_1, x_2)$  in the input space to satisfy  $K(x_1, x_2) = \langle \varphi(x_1), \varphi(x_2) \rangle$ . It shows the necessary and sufficient condition for the kernel technique given by the theorem.

## 1.3 Three Elements of Machine Learning Methods

Machine learning methods are all composed of models, strategies and algorithms; that is, machine learning methods are composed of three elements, which can be simply expressed as:

$$\text{Method} = \text{Model} + \text{Strategy} + \text{Algorithm}$$

The three elements of machine learning in supervised learning are discussed below. Unsupervised learning and reinforcement learning also have these three elements. It can be said that constructing a machine learning method determines the three elements of specific machine learning.

### 1.3.1 Model

The first consideration in machine learning is deciding the kind of model to learn. In the supervised learning process, the model is the conditional probability distribution or decision function to be learned. The hypothesis space of the model contains all possible conditional probability distributions or decision functions. For example, if the decision function is a linear function of the input variables, then the hypothesis space of the model is the set of functions formed by all these linear functions. Generally, there are infinite models in the hypothesis space.

The hypothesis space is denoted by  $\mathcal{F}$  and can be defined as a set of decision functions:

$$\mathcal{F} = \{f | Y = f(X)\} \quad (1.5)$$

Among them,  $X$  and  $Y$  are variables defined in the input space  $\chi$  and the output space  $\mathcal{Y}$ . At this time,  $\mathcal{F}$  is usually a family of functions determined by a parameter vector:

$$\mathcal{F} = \{f | Y = f_\theta(X), \theta \in \mathcal{R}^n\} \quad (1.6)$$

The parameter vector  $\theta$  is taken in the  $n$ -dimensional Euclidean space  $\mathcal{R}^n$ , which is called the parameter space.

The hypothesis space can also be defined as a set of conditional probabilities:

$$\mathcal{F} = \{P | P(Y|X)\} \quad (1.7)$$

Among them,  $X$  and  $Y$  are random variables defined in the input space  $\chi$  and the output space. In this case,  $\mathcal{F}$  is usually a conditional probability distribution family determined by a parameter vector:

$$\mathcal{F} = \{P | P_\theta(Y|X), \theta \in \mathcal{R}^n\} \quad (1.8)$$

The parameter vector  $\theta$  takes the value in the  $n$ -dimensional Euclidean space  $\mathcal{R}^n$ , also called the parameter space.

In this book, models represented by the decision function are referred to as non-probabilistic models, and models depicted by conditional probability are called probabilistic models. For the sake of simplicity, when discussing models, sometimes only one of those models is used.

### 1.3.2 Strategy

Once the hypothesis space of the model is available, machine learning then needs to consider the criteria to follow for learning or selecting the optimal model. And machine learning aims to select the optimal model from the hypothesis space.

Firstly, the concepts of the loss function and risk function are introduced. The loss function measures how well the model predicts once, and the risk function measures the quality of the average prediction.

#### (1) The loss function and the risk function

The supervised learning problem is to select model  $f$  as the decision function in the hypothesis space  $\mathcal{F}$ . For a given input  $X$ ,  $f(X)$  gives the corresponding output  $Y$ . The output predicted value  $f(X)$  may be consistent or inconsistent with the true value  $Y$ . Then a loss function or cost function is used to measure the degree of prediction error. The loss function is a non-negative real-valued function of  $f(X)$  and  $Y$ , denoted as  $L(Y, f(X))$ .

Loss functions commonly used in machine learning include:

(A) 0–1 loss function

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases} \quad (1.9)$$

(B) Quadratic loss function

$$L(Y, f(X)) = (Y - f(X))^2 \quad (1.10)$$

(C) Absolute loss function

$$L(Y, f(X)) = |Y - f(X)| \quad (1.11)$$

(D) Logarithmic loss function or log-likelihood loss function

$$L(Y, P(Y|X)) = -\log P(Y|X) \quad (1.12)$$

The smaller the loss function value, the better the model. Since the input and output  $(X, Y)$  of the model are random variables and follow the joint distribution  $P(X, Y)$ , the expected loss function is:

$$\begin{aligned} R_{\text{exp}}(f) &= E_P[L(Y, f(X))] \\ &= \int_{x \times y} L(y, f(x))P(x, y)dxdy \end{aligned} \quad (1.13)$$

This loss is the theoretical average loss of model  $f(X)$  concerning the joint distribution  $P(X, Y)$ , called the risk function or expected loss.

The goal of learning is to choose the model with the least expected risk. Since the joint distribution  $P(X, Y)$  is unknown,  $R_{\text{exp}}(f)$  cannot be computed directly. In fact, if you know the joint distribution  $P(X, Y)$ , you can directly find the conditional probability distribution  $P(Y|X)$  from the joint distribution, and you don't need to learn. It is just because the joint probability distribution is unknown that learning is needed. In this way, on the one hand, the joint distribution is used according to the expected risk minimum learning model. On the other hand, the joint distribution is unknown, so supervised learning becomes an ill-formed problem.

Given a training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

The average loss of model  $f(X)$  with respect to the training dataset is called empirical risk or empirical loss, denoted as  $R_{\text{emp}}(f)$ :

$$R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) \quad (1.14)$$

The expected risk  $R_{\text{exp}}(f)$  is the expected loss of the model concerning the joint distribution, and the empirical risk  $R_{\text{emp}}(f)$  is the average loss of the model with respect to the training sample set. According to the law of large numbers, the empirical risk  $R_{\text{emp}}(f)$  gets closer to the expected risk  $R_{\text{exp}}(f)$  when the sample size  $N$  verges on infinity. So, the natural idea is to use empirical risk to estimate expected risk. However, due to the limited or even small number of training samples in reality, using empirical risk to estimate expected risk is often not ideal, and correcting certain empirical risks is necessary. This relates to the two basic strategies of supervised learning: empirical risk minimization (ERM) and structural risk minimization (SRM).

## (2) Empirical risk minimization and structural risk minimization

When the hypothesis space, loss function, and training dataset are determined, the empirical risk function (1.14) can be determined. The empirical risk minimization (ERM) strategy believes that the model with the least empirical risk is the optimal model. According to this strategy, seeking the optimal model according to ERM is to solve the optimization problem:

$$\min_{f \in \mathcal{F}} = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) \quad (1.15)$$

Among them,  $\mathcal{F}$  is the hypothesis space.

When the sample size is large enough, the ERM strategy ensures a good learning effect and is widely used in reality. For example, Maximum Likelihood Estimation

is an example of ERM. When the model is a conditional probability distribution, and the loss function is a log loss function, the empirical risk minimization is equivalent to the Maximum Likelihood Estimation.

However, when the sample size is tiny, the effect of the ERM learning may not be excellent, and an “over-fitting” phenomenon will occur.

Structural risk minimization (SRM) is a strategy proposed to prevent over-fitting. Structural risk minimization is equivalent to regularization. Structural risk adds a regularizer or penalty term representing the model’s complexity to empirical risk. Under the circumstance where hypothesis space, the loss function and the training dataset are determined, the definition of structural risk is:

$$R_{\text{srm}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f) \quad (1.16)$$

where  $J(f)$  is the complexity of the model, which is a functional defined on the hypothesis space  $\mathcal{F}$ . The more complex the model  $f$ , the greater the complexity  $J(f)$ ; conversely, the simpler the model  $f$ , the smaller the complexity  $J(f)$ . In other words, complexity represents the penalty for complex models.  $\lambda \geq 0$  is a coefficient used to weigh empirical risk and model complexity. Small structural risk requires both experience risk and model complexity to be small. Models with small structural risks tend to have better predictions on training data and unknown test data.

The maximum posterior probability estimation (MAP) in Bayesian Estimation is an example of structural risk minimization. When the model is a conditional probability distribution, the loss function is a log loss function, and the model’s prior probability represents the model’s complexity, the structural risk minimization is equivalent to the maximum posterior probability estimation.

The strategy of structural risk minimization considers that the model with the least structural risk is the optimal model. Finding the optimal model is consequently to solve the optimization problem:

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f) \quad (1.17)$$

In this way, the supervised learning problem becomes the optimization problem (1.15) and (1.17) of the empirical or structural risk function. At this time, the empirical or structural risk function is the optimal objective function.

### 1.3.3 Algorithm

Algorithm refers to the specific computation method of the learning model. Based on the training dataset, machine learning selects the optimal model from the hypothesis space according to the learning strategy. Finally, it considers what computation method to use to solve the optimal model.

At this time, the machine learning problem is boiled down to the optimization problem, and the machine learning algorithm becomes the algorithm for solving the optimization problem. If the optimization problem has an explicit analytical solution, the optimization problem is relatively simple. But usually, the analytical solution does not exist, which requires a numerical computation method. Ensuring the optimal global solution is found, and the solution process is very efficient has become an important issue. Machine learning can use existing optimization algorithms, while sometimes, it is necessary to develop unique optimization algorithms.

The differences between machine learning methods mainly come from the differences in their models, strategies, and algorithms. Once the model, strategy, and algorithm are determined, the machine learning method is also determined. This is why they are called the three elements of machine learning methods. The following introduces several important concepts of supervised learning.

## 1.4 Model Evaluation and Model Selection

### 1.4.1 Training Error and Test Error

Machine learning aims to make the learned model have good predictive ability for known and unknown data. Different learning methods will give different models. When the loss function is fixed, the training error and the test error of the model based on the loss function will naturally become the evaluation criteria of the learning method. Note that the specific loss function used by the machine learning method may not be the loss function used in the evaluation. It is, of course, desirable that the two be aligned.

Suppose the learned model is  $Y = \hat{f}(X)$ , and the training error is the average loss of the model  $Y = \hat{f}(X)$  for the training dataset:

$$R_{\text{emp}}(\hat{f}) = \frac{1}{N} \sum_{i=1}^N L(y_i, \hat{f}(x_i)) \quad (1.18)$$

where  $N$  is the capacity of test samples.

The test error is the average loss of the model  $Y = \hat{f}(X)$  with respect to the test dataset:

$$e_{\text{test}} = \frac{1}{N'} \sum_{i=1}^{N'} L(y_i, \hat{f}(x_i)) \quad (1.19)$$

where  $N'$  is the capacity of test samples.

For example, when the loss function is 0–1 loss, the test error becomes the error rate on the common test dataset:

$$e_{\text{test}} = \frac{1}{N'} \sum_{i=1}^{N'} I(y_i \neq \hat{f}(x_i)) \quad (1.20)$$

Here  $I$  is the indicator function. When  $y_i \neq \hat{f}(x_i)$ , it is 1, otherwise it is 0.

Correspondingly, the accuracy of the common test dataset is

$$r_{\text{test}} = \frac{1}{N'} \sum_{i=1}^{N'} I(y_i = \hat{f}(x_i)) \quad (1.21)$$

Obviously,

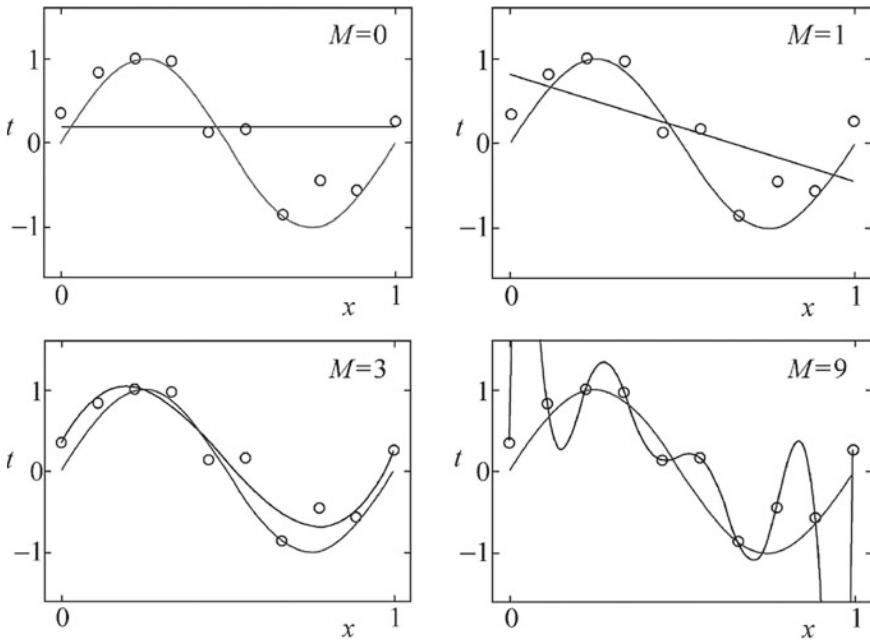
$$r_{\text{test}} + e_{\text{test}} = 1$$

The size of the training error is relevant in judging whether a given problem is an easy-to-learn problem, but it is not crucial. The test error reflects the predictive ability of the learning method on the unknown test dataset and is an essential concept in learning. Given two learning methods, the one with small test error has the better predictive ability and is more effective. The ability of learning methods to predict unknown data is usually called generalization ability. This problem will be discussed in Sect. 1.6.

#### 1.4.2 Over-Fitting and Model Selection

When the hypothesis space contains models of different complexity (e.g., different numbers of parameters), the problem of model selection arises. We hope to choose or learn a suitable model. If a True Model exists in the hypothesis space, then the selected model should approximate the True Model. Specifically, the chosen model must have the same number of parameters as the True Model, and the parameter vector of the selected model should be similar to the parameter vector of the True Model.

If you blindly seek to improve the predictive ability of training data, the complexity of the selected model will tend to be higher than that of the True Model. This phenomenon is known as over-fitting. Overfitting refers to the model chosen during learning containing too many parameters, so the model predicts the known data well



**Fig. 1.8** An example of a polynomial function fitting problem of degree  $M$

but the unknown data poorly. It can be argued that the model selection aims to avoid overfitting and improve the model's predictive ability.

The problem of polynomial function fitting is taken as an example here to illustrate overfitting and model selection. It is a regression problem.

Example 1.1 assuming a training dataset<sup>1</sup>:

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

Among them,  $x_i \in R$  is the observation value of input  $x$ ,  $y_i \in R$  is the corresponding observation value of output  $y$ ,  $i = 1, 2, \dots, N$ . The task of polynomial function fitting is to assume that the polynomial function of degree  $M$  generates the given data and to select the polynomial function of degree  $M$  that is most likely to generate those data. In other words, it selects a function with a good predictive ability for known and unknown data among the polynomial functions of degree  $M$ .

Assume that 10 data points, as shown in Fig. 1.8 are given, and the data are fitted with polynomial functions of degrees 0–9. The figure shows the data that needs to be fitted with a polynomial function curve.

Let the polynomial of degree  $M$  be

---

<sup>1</sup> This example is taken from [2].

$$f_M(x, \omega) = \omega_0 + \omega_1 x + \omega_2 x^2 + \cdots + \omega_M x^M = \sum_{j=0}^M \omega_j x^j \quad (1.22)$$

where  $x$  is a single variable input, and  $\omega_0, \omega_1, \dots, \omega_M$  are  $M + 1$  parameters.

The approach to solving this problem can be as follows. First, the complexity of the model is determined, i.e., the degree of the polynomial is determined; then, under the given complexity of the model, the parameters, that is, the coefficients of the polynomial, is solved according to the strategy of empirical risk minimization. To be specific, the following empirical risk minimization is sought:

$$L(\omega) = \frac{1}{2} \sum_{i=1}^N (f(x_i, \omega) - y_i)^2 \quad (1.23)$$

In this case, the loss function is square loss, and the coefficient 1/2 is for the convenience of computation.

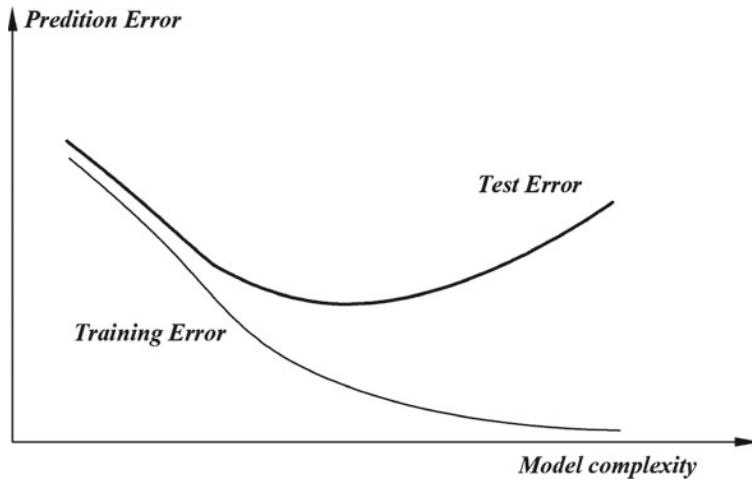
It is then a simple optimization problem. Substitute the model and training data into Eq. (1.23) and obtain

$$L(\omega) = \frac{1}{2} \sum_{i=1}^N \left( \sum_{j=0}^M \omega_j x_i^j - y_i \right)^2$$

The least square method can be used in this problem to find the unique solution to the fitted polynomial coefficients, denoted as  $\omega_1^*, \omega_2^*, \dots, \omega_M^*$ . The solution process is not described here, and interested readers can refer to relevant materials.

Figure 1.8 shows the fit of the polynomial function when  $M = 0, M = 1, M = 3$  and  $M = 9$ . If  $M = 0$ , the polynomial curve is a constant, and the data fitting effect is poor. If  $M = 1$ , the polynomial curve is a straight line, and the data fitting effect is also poor. In contrast, if  $M = 9$ , the polynomial curve passes through each data point, and the training error is 0, which is the best result in terms of fitting the given training data. However, because of the inherent noise in the training data, such fitted curves are often not the best predictors of unknown data and are not desirable for practical learning purposes. This is where overfitting occurs. It means that model selection should consider the predictive power not only for known data but also for unknown data. When  $M = 3$ , the polynomial curve fits the training data well enough, and the model is relatively simple, making it a better choice.

As can be seen in polynomial function fitting, the training error decreases as the number of polynomials (model complexity) increases until it approaches 0. But the test error is not the same, as it will first decrease and then increase with the increase of polynomial times (model complexity). The ultimate goal is to minimize the test error. In this way, the appropriate polynomial degree should be chosen to achieve this goal in the polynomial function fitting. This conclusion also holds for general model selection.



**Fig. 1.9** Relationship between training and test errors and the model complexity

Figure 1.9 depicts the relationship between training and test errors and the model complexity. When the model complexity increases, the training error decreases gradually and approaches 0, while the test error decreases, reaches a minimum, and then increases again. Overfitting occurs when the complexity of the chosen model is too high. In this way, it is necessary to prevent overfitting and select the optimal model in learning, i.e., to choose the one with appropriate complexity to minimize the test error. Here are two commonly used model selection methods: regularization and cross-validation.

## 1.5 Regularization and Cross-Validation

### 1.5.1 Regularization

The typical method of model selection is regularization. Regularization is the realization of a structural risk minimization strategy, which is to add a regularizer or penalty term to empirical risk. The regularization term is generally a monotonically increasing function of model complexity, and the more complex the model, the greater the regularization value. For example, the regularization term can be the norm of the model parameter vector.

Regularization is generally of the form:

$$\min_{f \in F} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f) \quad (1.24)$$

Among them, item 1 is the empirical risk, item 2 is the regularizer, and  $\lambda \geq 0$  is the coefficient to adjust the relationship between the two items.

The regularized item (regularizer) can take different forms. For example, in the regression problem, the loss function is the square loss, and the regularized item can be the  $L_2$  norm of the parameter vector:

$$L(w) = \frac{1}{N} \sum_{i=1}^N (f(x_i; w) - y_i)^2 + \frac{\lambda}{2} \|w\|^2 \quad (1.25)$$

Here,  $\|w\|$  represents the  $L_2$  norm of the parameter vector  $w$ .

The regularized item can also be the  $L_1$  norm of the parameter vector:

$$L(w) = \frac{1}{N} \sum_{i=1}^N (f(x_i; w) - y_i)^2 + \lambda \|w\|_1 \quad (1.26)$$

Here,  $\|w\|_1$  represents the  $L_1$  norm of the parameter vector  $w$ .

The model with less empirical risk in item 1 may be more complex (with multiple non-zero parameters), in which case the model complexity in item 2 will be greater. The function of regularization is to choose a model with a more negligible empirical risk and model complexity.

Regularization conforms to the principle of Occam's razor. When Occam's razor principle is applied to model selection, it becomes the following idea: Of all the possible models to be selected, the one that explains the known data well and is very simple is the best model that should be chosen. From the perspective of Bayesian Estimation, the regularized item corresponds to the prior probability of the model. It can be assumed that a complex model has a smaller prior probability, and a simple model has a larger prior probability.

### 1.5.2 Cross-Validation

Another commonly used method of model selection is cross-validation.

If the given sample data is sufficient, a simple method for model selection is to randomly divide the dataset into three parts: the training set, the validation set, and the test set. The training set is used to train the model, the validation set is for model selection, and the test set is for the final evaluation of the learning method. Among the learned models with different complexity, the one with the smallest prediction error on the validation set is selected. Since the validation set has enough data, it is also effective to use it for model selection.

However, the data is not sufficient in many practical applications. In order to select a good model, the cross-validation method can be used. The basic idea of cross-validation is to use data repeatedly, segment the given data, then combine the

segmented dataset into the training and test sets, on which training, testing and model selection are repeatedly performed.

### (1) Simple cross-validation

A simple cross-validation method comprises the following step:

- (A) Randomly divide the given data into two parts, one as the training set and the other as the test set (e.g., 70% of data is used as the training set, and 30% of data is used as the test set).
- (B) Train models by using the training set under various conditions (e.g., different parameter numbers) to obtain different models.
- (C) Evaluate the test error of each model on the test set, and select the model with the minimum test error.

### (2) S-fold cross-validation

The most commonly used method is  $S$ -fold cross-validation. The method is as follows: first, randomly divide the given data into  $S$  disjoint subsets of the same size; then, train the model using the data of  $S - 1$  subsets, and test the model with the remaining subsets. This process is repeated on  $S$  possible choices, and finally, the model with the smallest average test error in  $S$  evaluations is selected.

### (3) Leave-one-out cross-validation

The special case of  $S$ -fold cross-validation is  $S = N$ , called leave-one-out cross-validation, and is often used when there is a lack of data. Here,  $N$  refers to the capacity of a given dataset.

## 1.6 Generalization Ability

### 1.6.1 Generalization Error

The generalization ability of a learning method refers to the predictive ability of the model learned by this method for unknown data, which is an essentially important property of the learning method. In reality, the most commonly used method is to evaluate the generalization ability of learning methods through test errors. However, this evaluation depends on the test dataset. Since the test dataset is limited, the evaluation results from the set are likely to be unreliable. Machine learning theory attempts to analyze the generalization ability of learning methods theoretically.

First, the definition of generalization error is given. If the learned model is  $\hat{f}$ , then the error of predicting unknown data by using this model is the generalization error:

$$R_{\text{exp}}(\hat{f}) = E_P[L(Y, \hat{f}(X))]$$

$$= \int_{x \times y} L(y, \hat{f}(x)) P(x, y) dx dy \quad (1.27)$$

The generalization error reflects the generalization ability of the learning method. If the model learned by one method has a smaller generalization error than the model learned by another, then this method is more effective. In fact, the generalization error is the expected risk of the learned model.

### 1.6.2 Generalization Error Bound

The generalization ability of learning methods is often analyzed by studying the upper bound of the probability of generalization error, referred to as the generalization error bound for short. Specifically, the pros and cons of the two learning methods are compared by comparing the size of their generalization error bounds. The generalization error bound usually has the following properties: it is a function of the sample size, and when the sample size increases, the generalization upper bound approaches 0; it is a function of the hypothesis space capacity, and the larger the hypothesis space capacity, the more difficult the model is to learn, and the larger the generalization error bound.

A simple example of the generalization error bound is given below: the generalization error bound for binary classification problems.

Consider the binary classification problem. Given the training dataset as  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $N$  is the sample size, and  $T$  is generated from the joint probability distribution  $P(X, Y)$ , independent and identically distributed,  $X \in \mathbf{R}^n$ ,  $Y \in \{-1, +1\}$ . Hypothesis space is a finite set of functions  $\mathcal{F} = \{f_1, f_2, \dots, f_d\}$ , where  $d$  is the number of functions. Let  $f$  be a function selected from  $\mathcal{F}$ . The loss function is 0–1 loss. The expected risk and empirical risk of  $f$  are:

$$R(f) = E[L(Y, f(X))] \quad (1.28)$$

$$\hat{R}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) \quad (1.29)$$

The empirical risk minimization function is

$$f_N = \arg \min_{f \in F} \hat{R}(f) \quad (1.30)$$

$f_N$  depends on  $N$ , the sample size of the training dataset. What people are more concerned about is the generalization ability of  $f_N$ .

$$R(f_N) = E[L(Y, f_N(X))] \quad (1.31)$$

The following discusses the upper bound of the generalization error of the function  $f$  arbitrarily selected from the finite set  $\mathcal{F} = \{f_1, f_1, \dots, f_d\}$ .

**Theorem 1.1 (the generalization error bound)** For binary classification problems, when the hypothesis space is a set of finite functions  $\mathcal{F} = \{f_1, f_1, \dots, f_d\}$ , for any function  $f \in \mathcal{F}$ , at least with probability  $1 - \delta$ ,  $0 < \delta < 1$  the following inequality holds:

$$R(f) \leq \hat{R}(f) + \varepsilon(d, N, \delta) \quad (1.32)$$

where,

$$\varepsilon(d, N, \delta) = \sqrt{\frac{1}{2N} \left( \log d + \log \frac{1}{\delta} \right)} \quad (1.33)$$

The left end of Inequality (1.32)  $R(f)$  is the generalization error, and the right end is the generalization error bound. In generalization error bound, the first item is the training error. The smaller the training error, the smaller the generalization error. The second item  $\varepsilon(d, N, \delta)$  is a monotonically decreasing function of  $N$ , which verges on 0 when  $N$  verges on infinity; at the same time, it is also a  $\sqrt{\log d}$ -order function. The more functions contained in the hypothesis space  $\mathcal{F}$ , the greater the value of  $\varepsilon(d, N, \delta)$ .

**Proof** Hoeffding's inequality is used in the proof, which is described as follows.

Suppose  $X_1, X_2, \dots, X_N$  are independent random variables, and  $X_i \in [a_i, b_i]$ ,  $i = 1, 2, \dots, N$ ;  $\bar{X}$  is the empirical mean of  $X_1, X_2, \dots, X_N$ , that is,  $\bar{X} = \frac{1}{N} \sum_{i=1}^N X_i$ , then for any  $t > 0$ , the following inequality holds:

$$P[\bar{X} - E(\bar{X}) \geq t] \leq \exp\left(-\frac{2N^2t^2}{\sum_{i=1}^N (b_i - a_i)^2}\right) \quad (1.34)$$

$$P[E(\bar{X}) - \bar{X} \geq t] \leq \exp\left(-\frac{2N^2t^2}{\sum_{i=1}^N (b_i - a_i)^2}\right) \quad (1.35)$$

The Hoeffding's inequality, whose proof is omitted, is used here to derive the generalization error bound.

For any function  $f \in \mathcal{F}$ ,  $\hat{R}(f)$  is the sample mean of  $N$  independent random variables  $L(Y, f(X))$ , and  $R(f)$  is the expected value of the random variable  $L(Y, f(X))$ . If the loss function values in the interval  $[0, 1]$ , that is, for all  $i$ ,  $[a_i, b_i] = [0, 1]$ , then it is not difficult to know from Hoeffding's inequality (1.35) that for  $\varepsilon > 0$ , the following inequality holds:

$$P(R(f) - \hat{R}(f) \geq \varepsilon) \leq \exp(-2N\varepsilon^2) \quad (1.36)$$

Since  $\mathcal{F} = \{f_1, f_2, \dots, f_d\}$  is a finite set,

$$\begin{aligned} P(\exists f \in \mathcal{F} : R(f) - \hat{R}(f) \geq \varepsilon) &= P(\bigcup_{f \in \mathcal{F}} \{R(f) - \hat{R}(f) \geq \varepsilon\}) \\ &\leq \sum_{f \in \mathcal{F}} P(R(f) - \hat{R}(f) \geq \varepsilon) \\ &\leq d \exp(-2N\varepsilon^2) \end{aligned}$$

or equivalently, for any  $f \in \mathcal{F}$ , there is

$$P(R(f) - \hat{R}(f) < \varepsilon) \geq 1 - d \exp(-2N\varepsilon^2) \quad (1.37)$$

Suppose that,

$$\delta = d \exp(-2N\varepsilon^2) \quad (1.38)$$

then,

$$P(R(f) < \hat{R}(f) + \varepsilon) \geq 1 - \delta$$

That is, there is  $R(f) < \hat{R}(f) + \varepsilon$  at least with probability  $1 - \delta$ , where  $\varepsilon$  is obtained by Formula (1.38), which is Formula (1.33).

From the generalization error bound, it is clear that

$$R(f_N) \leq \hat{R}(f_N) + \varepsilon(d, N, \delta) \quad (1.39)$$

Among them,  $\varepsilon(d, N, \delta)$  is defined by Formula (1.33), and  $f_N$  is defined by Formula (1.30).

The above discussion is only about the generalization error bound when the hypothesis space contains a finite number of functions. It is not so simple to find the generalization error bound for the general hypothesis space, which is not introduced here.

## 1.7 Generative Approach and Discriminative Model

The task of supervised learning is to learn a model, which is applied to predict the corresponding output for a given input. The general form of this model is a decision function:

$$Y = f(X)$$

or the conditional probability distribution:

$$P(Y|X)$$

Supervised learning methods can be divided into the generative approach and the discriminative approach. The models learned are called generative models and discriminative models.

In principle, The generative approach is based on the principle of learning the joint probability distribution  $P(X, Y)$  from the data, and then obtaining the conditional probability distribution  $P(Y|X)$  as the prediction model, that is, the generative model:

$$P(Y|X) = \frac{P(X, Y)}{P(X)} \quad (1.40)$$

This approach is called generative approach because the model represents the generative relationship between a given input  $X$  and an output  $Y$ . Typical generative models include Naive Bayesian Method and Hidden Markov Model, which will be described later.

The discriminative approach learns the decision function  $f(X)$  or the conditional probability distribution  $P(Y|X)$  directly from the data as the prediction model, i.e., the discriminative model. The discriminative approach concerns what output  $Y$  should be predicted for a given input  $X$ . Typical discriminative models include KNN, Perceptron, Logistic Regression Model, Maximum Entropy Model, Support Vector Machine, Lifting Method and Conditional Random Field, etc., which will be described in later chapters.

In supervised learning, the generative approach and the discriminative approach have their advantages and disadvantages, and are suitable for learning problems under different conditions.

Characteristics of the generative approach are as follows: the generative approach can restore the joint probability distribution  $P(X, Y)$ , whereas the discriminative approach cannot; the learning of the generative approach converges faster, i.e., when the sample size increases, the learned model can converge to the true model faster; when there are hidden variables, the generative approach can still be used to learn, while the discriminative approach cannot be used then.

Features of the discriminative approach are listed as follows. The discriminative approach directly learns the conditional probability  $P(Y|X)$  or the decision function  $f(X)$ , and is directly confronted with the prediction. It tends to learn more accurately. Learning problem can be simplified by the direct learning of  $P(Y|X)$  or  $f(X)$ , allowing various degrees of data abstraction, feature definition and feature application.

## 1.8 Supervised Learning Application

The application of supervised learning is mainly in three aspects: classification problem, tagging problem and regression problem.

### 1.8.1 Classification

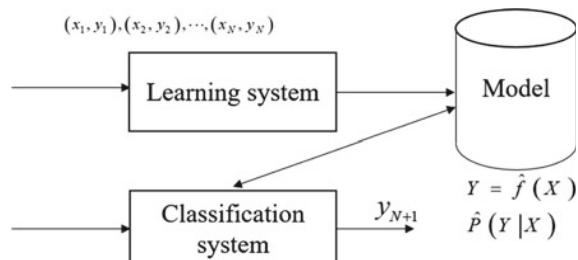
Classification is a core issue of supervised learning. In supervised learning, the prediction problem turns into a classification problem when the output variable  $Y$  takes a limited number of discrete values. At this time, the input variable  $X$  can be discrete or continuous. Supervised learning learns a classification model or classification decision function from data, called a classifier. The classifier predicts the new input's output, called classification. The possible outputs are called classes. When there are multiple classes of classification, it is called a multi-class classification problem. This book focuses on binary classification problems.

Classification problems involve both learning and classification processes. In the learning process, an effective learning method is used to learn a classifier based on the known training dataset; in the classification process, the learned classifier is used to classify new input instances. Classification problems can be described in Fig. 1.10. In the figure,  $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$  is the training dataset and the learning system learns a classifier  $P(Y|X)$  or  $Y = f(X)$  from training data; The classification system classifies a new input instance  $x_{N+1}$  by the learned classifier  $P(Y|X)$  or  $Y = f(X)$ , i.e., predicts its output class tag  $y_{N+1}$ .

A general index for evaluating classifier performance is classification accuracy, defined as the ratio of the number of samples correctly classified by the classifier to the total number of samples for a given test dataset, that is to say, the accuracy of the test dataset when the loss function is 0–1 loss [see Formula (1.21)].

The commonly used evaluation indicators for binary classification problems are Precision and Recall. The class of interest is usually taken as the positive class, and the other classes are the negative ones. The classifier's predictions on the test dataset are either correct or incorrect, with the total number of the four possible cases noted as:

**Fig. 1.10** The classification problem



TP—the positive class predicted as the positive class number;  
 FN—the positive class predicted as the negative class number;  
 FP—the negative class predicted as the positive class number;  
 TN—the negative class predicted as the negative class number.

The Precision is defined as

$$P = \frac{TP}{TP + FP} \quad (1.41)$$

The Recall is defined as

$$R = \frac{TP}{TP + FN} \quad (1.42)$$

In addition, there is the value of  $F_1$ , which is the harmonic mean of Precision and Recall, that is:

$$\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R} \quad (1.43)$$

$$F_1 = \frac{2TP}{2TP + FP + FN} \quad (1.44)$$

When the precision and recall rates are high, the  $F_1$  value will be high.

Many machine learning methods can be used for classification, including KNN, Perception, Naive Bayes, Decision Tree, Decision Table, Logistic Regression Model, Support Vector Machine, Boosting, Bayesian network, Neural Networks, Winnow, etc. This book will cover some main methods among them.

Classification is the ‘sorting’ of data based on its characteristics, so it has a wide range of applications in many fields. For instance, in banking, a customer classification mode can be built to sort customers according to their loan risk. In cyber security, the classification of log data can be used to detect illegal intrusions. In image processing, classification can be used to detect whether there is a face in the image. In handwriting recognition, classification can be used to recognize handwritten numbers. In the internet search, web page classification can help crawl, index and sort web pages.

An example of a classification application is text classification. The text here can be news reports, web pages, emails, academic papers, etc. Categories are often related to text content, such as politics, economics, sports, etc., or the characteristics of the text, such as positive and negative opinions. They can also be determined according to the application, such as spam, non-spam, etc. Text classification is to classify text into existing classes based on their characteristics. The input is the feature vector of the text, and the output is the category of the text. Words in the text are usually defined as features, with each word corresponding to a feature. A word feature can be binary, taking the value 1 if the word appears in the text and 0 otherwise. It can also be multivalued, indicating how often a word appears in the text. Intuitively, If

the words “stock”, “bank” and “currency” appear a lot, this text is likely to belong to the economic category; If the words “tennis”, “competition” and “player” appear frequently, this text may belong to the sports category.

### 1.8.2 Tagging

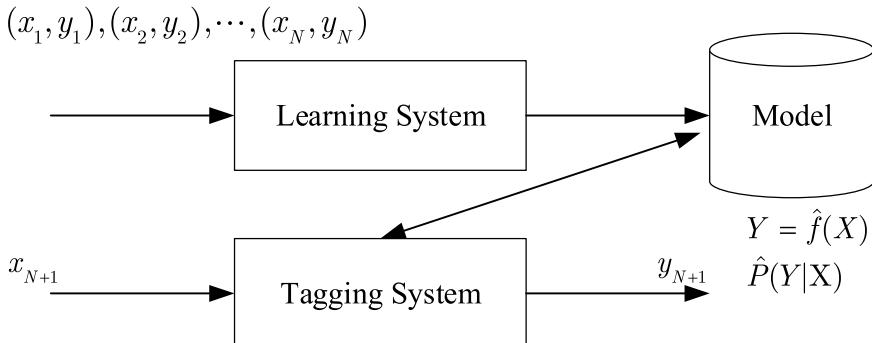
Tagging is also a supervised learning problem. It can be considered that the tagging problem is a generalization of the classification problem and a simple form of the more complex structural prediction problem. The input to the tagging problem is a sequence of observations, and the output is a tagged sequence or state sequence. The goal of tagging problems is to learn a model, which is possible to give a tagged sequence to the observation sequence as a prediction. Note that the number of possible tags is limited. However, the number of tagged sequences formed by the combination increases exponentially according to the length of the sequence.

The tagging problem is divided into two processes: tagging and learning (as shown in Fig. 1.11). First, given a training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

Here,  $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(N)})^T$ ,  $i = 1, 2, \dots, N$ , is the input observation sequence;  $y_i = (y_i^{(1)}, y_i^{(2)}, \dots, y_i^{(N)})^T$  is the corresponding output sequence;  $n$  is the length of the sequence, which has different values for different samples. The learning system builds a model based on the training dataset, expressed as a conditional probability distribution:

$$P(Y^{(1)}, Y^{(2)}, \dots, Y^{(n)}) |, X^{(1)}, X^{(2)}, \dots, X^{(n)}$$



**Fig. 1.11** The tagging problem

Here, each  $X^{(i)}$  ( $i = 1, 2, \dots, n$ ) takes the value of all possible observations, each  $Y^{(i)}$  ( $i = 1, 2, \dots, n$ ) takes the value of all possible tags, and commonly  $n \ll N$ . The tagging system follows the conditional probability distribution model gained by learning and finds the corresponding output tagged sequence for the new input observation sequence. Specifically, for an observation sequence as  $x_{N+1} = (x_{N+1}^{(1)}, x_{N+1}^{(2)}, \dots, x_{N+1}^{(n)})^T$ , it finds the tagged sequence  $y_{N+1} = (y_{N+1}^{(1)}, y_{N+1}^{(2)}, \dots, y_{N+1}^{(n)})^T$  which Maximizes the conditional probability  $P((y_{N+1}^{(1)}, y_{N+1}^{(2)}, \dots, y_{N+1}^{(n)})^T | (x_{N+1}^{(1)}, x_{N+1}^{(2)}, \dots, x_{N+1}^{(n)})^T)$ .

The index of tagging model evaluation is the same as that of classification model evaluation. The frequently used indexes are tagging accuracy rate, precision rate and recall rate. And the tagging model shares the same definition with the classification model.

The common-used machine learning methods in tagging are Hidden Markov Model and Conditional Random Field.

Tagging is widely used in information extraction, natural language processing and other fields, also being a basic problem of them. For example, part of speech tagging in natural language processing is a typical tagging problem: given a sentence composed of words, apply part of speech tagging to each word in the sentence, that is, predict the corresponding part of speech tagging sequence of a word sequence.

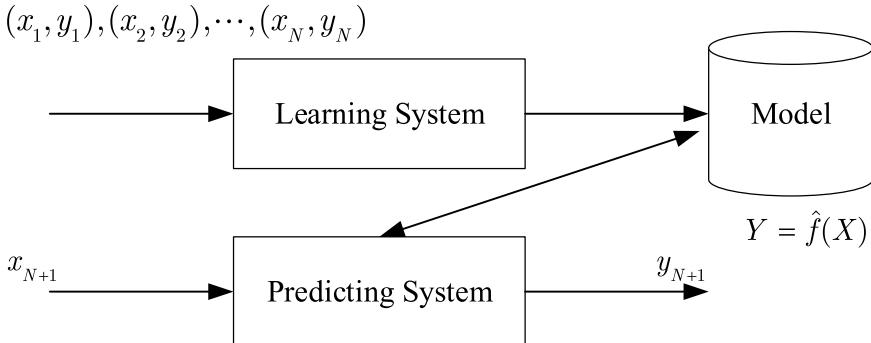
Take an example of information extraction—Extracting base noun phrases from an English article. To this end, the article should be tagged. The English word is an observation, and the English sentence is an observation sequence. The tags indicate the “beginning”, “end” or “other” (denoted respectively by B, E, O) of the noun phrase, and the tagged sequence indicates the location of the base noun phrase in the English sentence. During information extracting, the words between the tag “beginning” and the tag “end” are regarded as noun phrases. For example, given the following observation sequence, that is, an English sentence, the tagging system generates the corresponding tagged sequence, i.e., the base noun phrases in the sentence.

**Input:** At Microsoft Research, we have an insatiable curiosity and the desire to create new technology that will help define the computing experience.

**Output:** At/O Microsoft/B Research/E, we/O have/O an/O insatiable/B curiosity/E and/O the/O desire/BE to/O create/O new/B technology/E that/O will/O help/O define/O the/O computing/B experience/E.

### 1.8.3 Regression

Regression is another vital issue in supervised learning. Regression is used to predict the relationship between the input variable (Independent variable) and output variable (Dependent variable), especially the corresponding change of the value of the output variable while the value of the input variable changes. The regression model is the function that represents the mapping from input variables to output variables. The learning of regression problem is equivalent to function fitting: selecting a function



**Fig. 1.12** The regression problem

curve that fits the known data well and predicts the unknown data well (Refer to Sect. 1.4.2).

The regression problem is divided into two processes: learning and prediction (as shown in Fig. 1.12). Firstly, a training dataset is given:

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

Here,  $x_i \in \mathbf{R}^n$  is the input,  $y \in \mathbf{R}$  is the corresponding output,  $i = 1, 2, \dots, N$ . The learning system builds a model based upon training data, that is function  $Y = f(X)$ ; For the new output  $x_{N+1}$ , the predicting system generates the corresponding output  $y_{N+1}$  according to the learned model.

Regression problems can be classified into simple regression and multiple regression according to the number of input variables, and linear regression and non-linear regression according to the relationship between the input and output variables, i.e., the model type.

The most commonly used loss function in regression learning is the square loss function, in which the regression problem can be solved by the well-known least squares.

Tasks in many areas can be formalized as regression problems. For instance, regression can be used in business as a tool for market trend prediction, product quality management, customer satisfaction survey and investment risk analysis. As an example, the problem of stock price forecasting is briefly introduced here. Suppose that the stock price (e.g., the average stock price) of a company in the market at different points in time (e.g., every day) in the past is known, as well as information that may affect the company's stock price before each time point (e.g., the company's turnover and profit in the previous week). The objective is to learn a model from previous data to predict the company's stock price at the next point of time based on current information. This problem can be solved as a regression problem. Specifically, the information affecting the stock price is regarded as the independent variable (the input feature), and the stock price is considered the dependent variable (the output

value). Learning a regression model and predicting the future stock price is possible by using past data as training data. It is a difficult prediction problem because many factors affect the stock price, and we may not be able to determine and get helpful information (the input feature).

## Summary

1. Machine learning is a subject that involves computer building probability statistical models based on data and using the model to analyze and predict the data. Machine learning includes supervised learning, unsupervised learning and reinforcement learning.
2. The three elements of machine learning methods are model, strategy and algorithm, which play significant roles in understanding machine learning methods.
3. The first chapter of this book mainly discusses supervised learning. Supervised learning can be summarized as follows: it starts from the given limited training data, assumes that the data is independent and identically distributed, and the hypothesis model belongs to a specific hypothesis space. A certain evaluation criterion is applied to select an optimal model from the hypothesis space so that the model makes the most accurate prediction on the given training data and unknown test data under the given evaluation standard.
4. In machine learning, it is crucial to select models or improve the generalization ability of learning. The analysis of the generalization ability of learning methods is an essential topic in research on machine learning theory. If only reducing training errors is considered, over-fitting may occur. The methods of model selection include regularization and cross-validation.
5. Classification, tagging and regression are all crucial problems in supervised learning. The machine learning methods introduced in this chapter of the book include Perceptron, K-nearest Neighbor  $k$ , Naive Bayes Method, Decision Tree, Logistic Regression and Maximum Entropy Model, Support Vector Machine, Lifting Method, EM algorithm, Hidden Markov Model and Conditional Random Field. These methods are principal methods used in classification, tagging and regression. They can be classified into generative approaches and discriminative approaches.

## Further Reading

For a general introduction to machine learning methods, please refer to the references in this chapter.

## Exercises

- 1.1 Explain the three elements of machine learning methods in the Maximum Likelihood Estimation of the Bernoulli model and the Bayesian Estimation. Bernoulli model is a probability distribution defined on random variables with

values of 0 and 1. Suppose that  $n$  times independent data generation results of the Bernoulli model are observed, of which  $k$  results are 1. In this case, the Maximum Likelihood Estimation or the Bayesian Estimation can be used to estimate the probability of result 1.

- 1.2 Derive Maximum Likelihood Estimation by empirical risk minimization. Prove that the empirical risk minimization is equivalent to Maximum Likelihood Estimation when the model is in the conditional probability distribution, and the loss function is logarithmic.

## References

1. Hastie T, Tibshirani R, Friedman J. *The elements of statistical learning: data mining, inference, and prediction*. Springer; 2001.
2. Bishop C. M. *Pattern recognition and machine learning*. Springer; 2006.
3. Koller D, Friedman N. *Probabilistic graphical models: principles and techniques*. MIT Press; 2009.
4. Goodfellow I, Begion Y, Courville A, et al. *Deep learning*. MIT Press; 2016.
5. Mitchell TM. *Machine learning*. McGraw-Hill Companies, Inc.; 1997.
6. Barber D. *Bayesian reasoning and machine learning*. Cambridge University Press; 2012.
7. Thrun S, Littman ML. *Reinforcement learning: an introduction*. MIT Press; 1998.
8. Zhou ZH. Zhou Zhi-Hua. *Machine Learning*: Tsinghua University Press; 2017.

# Chapter 2

## Perceptron

The perceptron is a linear classification model for binary classification. Its input is the feature vector of the instance, and the output is the class of the instance, taking two values as  $+1$  and  $-1$ . The perceptron corresponds to the separating hyperplane in the input space (feature space) that divides the instances into positive and negative classes. It belongs to the discriminative model. Perceptron learning aims to find a separating hyperplane that linearly divides the training data. To this end, a loss function based on misclassification is introduced, and the loss function is minimized by the gradient descent method to obtain the perceptron model. Perceptron learning algorithms, divided into primal and dual forms, have the advantages of being simple and easy to implement. The perceptron prediction uses the learned perceptron model to classify new input instances. The perceptron, proposed by Rosenblatt in 1957, is the foundation of Neural Networks and Support Vector Machine.

This chapter first introduces the perceptron model, then describes the learning strategy of the perceptron, especially the loss function, and finally presents perceptron learning algorithms, including the primitive form and the dual form, and proves the algorithm's convergence.

### 2.1 The Perceptron Model

#### Definition 2.1 (*Perceptron*)

Suppose the input space (feature space) is  $\mathcal{X} \subseteq \mathbf{R}^n$ , and the output space is  $\mathcal{Y} = \{+1, -1\}$ . The input  $x \in \mathcal{X}$  represents the feature vector of the instance, corresponding to the point in the input space (feature space); the output  $y \in \mathcal{Y}$  represents the class of the instance. The following function from the input space to the output space is called the perceptron:

$$f(x) = \text{sign}(w \cdot x + b) \quad (2.1)$$

In this function,  $w$  and  $b$  are parameters of the perceptron model, and  $w \subseteq \mathbf{R}^n$  is called the weight or the weight vector;  $b \in \mathbf{R}$  is called the bias, and  $w \cdot x$  represents the inner product of  $w$  and  $x$ ; sign is a symbol-function, i.e.,

$$\text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases} \quad (2.2)$$

The perceptron is a linear classification model and belongs to the discriminative model. The hypothesis space of the perceptron model is equivalent to all the linear classification models or linear classifiers defined in the feature space, i.e., the set of functions  $\{f | f(x) = w \cdot x + b\}$ .

The perceptron has the following geometric interpretation: the linear equation

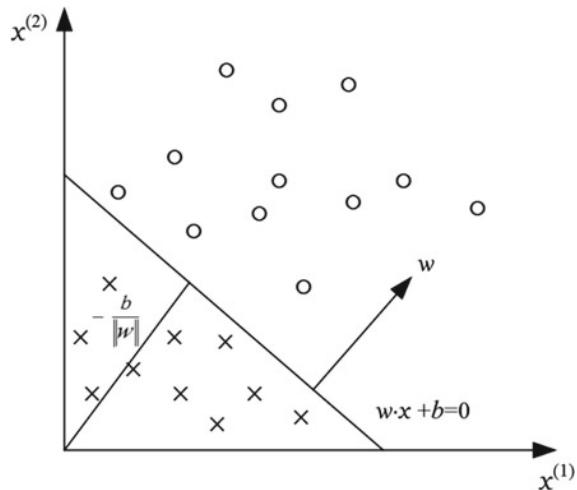
$$w \cdot x + b = 0 \quad (2.3)$$

corresponds to a hyperplane  $S$  in the feature space  $\mathbf{R}^n$ , where  $w$  and  $b$  represents the normal vector and the intercept of the hyperplane, respectively. This hyperplane divides the feature space into two parts. The points (feature vectors) in the two parts are divided into positive and negative classes. Therefore, the hyperplane  $S$  is called a separating hyperplane, as shown in Fig. 2.1.

In perceptron learning, the perceptron model (see Fig. 2.1) is obtained, i.e., the model parameters  $w$  and  $b$  are derived according to the training dataset (the feature vector and class of the instance):

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

**Fig. 2.1** The perceptron model



where  $x_i \in \mathcal{X} = \mathbb{R}^n$ ,  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ . The perceptron prediction gives the corresponding output class for the new input instance using the perceptron model obtained by learning.

## 2.2 Perceptron Learning Strategy

### 2.2.1 Linear Separability of the Dataset

**Definition 2.2 (linear separability of dataset)** Given a dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i \in \mathcal{X} = \mathbb{R}^n$ ,  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ . If there is a hyperplane  $S$

$$w \cdot x + b = 0$$

that can precisely divide the positive instance points and negative instance points of the dataset into both sides of the hyperplane, i.e., for all instances  $i$  with  $y_i = +1$ ,  $w \cdot x + b > 0$ , and for all instances with  $y_i = -1$ ,  $w \cdot x + b < 0$ , then the dataset  $T$  is called a linearly separable dataset; Otherwise, the dataset  $T$  is called a linearly inseparable one.

### 2.2.2 Perceptron Learning Strategy

Assuming that the training dataset is linearly separable, perceptron learning aims to find a separating hyperplane that can separate the training set's positive and negative instance points entirely and correctly. For finding such a hyperplane, i.e., determining the perceptron model parameters  $w$ ,  $b$ , it is necessary to determine a learning strategy, i.e., to define an empirical loss function and minimize it.

A natural choice for the loss function is the total number of misclassified points. However, such a loss function is not a continuously differentiable function of the parameters  $w$  and  $b$ , making it difficult to optimize. Another option for the loss function is the total distance from the misclassified point to the hyperplane  $S$ , which is what the perceptron employs. To that end, first, we can write down the distance from any point  $x_0$  in the input space  $\mathbb{R}^n$  to the hyperplane  $S$ :

$$\frac{1}{\|w\|} |w \cdot x_0 + b|$$

Here,  $\|w\|$  is the  $L_2$  norm of  $w$ .

Secondly, for misclassified data  $(x_i, y_i)$ ,

$$-y_i(w \cdot x_i + b) > 0$$

is true. This is because when  $w \cdot x + b > 0$ ,  $y_i = -1$  and when  $w \cdot x + b < 0$ ,  $y_i = +1$ . Therefore, the distance from the misclassified point  $x_i$  to the hyperplane  $S$  is

$$-\frac{1}{\|w\|} y_i(w \cdot x_i + b)$$

Thus, assuming that the set of misclassified points in the hyperplane  $S$  is  $M$ , then the total distance from all misclassified points to the hyperplane  $S$  is

$$-\frac{1}{\|w\|} \sum_{x_i \in M} y_i(w \cdot x_i + b)$$

without considering  $\frac{1}{\|w\|}$ , and the loss function learned by the perceptron is obtained.<sup>1</sup>

Given the training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i \in \mathcal{X} = \mathbf{R}^n$ ,  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ . The loss function learned by the perceptron  $sign(w \cdot x + b)$  is defined as

$$L(w, b) = - \sum_{x_i \in M} y_i(w \cdot x_i + b) \quad (2.4)$$

where  $M$  is the set of misclassified points. This loss function is the empirical risk function learned by the perceptron.

Obviously, the loss function  $L(w, b)$  is non-negative. If there is no misclassified point, the loss function value is 0. Moreover, the fewer the misclassified points are, the closer they are to the hyperplane, and the smaller the loss function value is. The loss function of a specific sample point is a linear function of the parameters  $w$  and  $b$  when misclassified, while it is 0 when correctly classified. Therefore, given the training dataset  $T$ , the loss function  $L(w, b)$  is a continuously differentiable function of  $w, b$ .

The perceptron learning strategy is to select the model parameter  $w$  and  $b$  that minimize the loss function (2.4) in the hypothesis space, i.e., the perceptron model.

---

<sup>1</sup> Chapter 7 will introduce  $y(w \cdot x_i + b)$ , the functional margin of sample points.

## 2.3 Perceptron Learning Algorithm

The perceptron learning problem is transformed into the optimization problem of solving the loss function (2.4), and the optimization method is stochastic gradient descent (SGD). This section describes the specific algorithms of perceptron learning, including the primal and dual forms. It then proves the convergence of the perceptron learning algorithms under the condition that the training data is linearly separable.

### 2.3.1 The Primal Form of the Perceptron Learning Algorithm

The perceptron learning algorithm is an algorithm for the following optimization problem. Given a training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i \in \mathcal{X} = \mathbb{R}^n$ ,  $y_i \in \mathcal{Y} = \{-1, 1\}$ ,  $i = 1, 2, \dots, N$ . Find the parameters  $w, b$  to be the solution of the following loss function minimization problem:

$$\min_{w, b} L(w, b) = - \sum_{x_i \in M} y_i (w \cdot x_i + b) \quad (2.5)$$

where  $M$  is the set of misclassified points.

The perceptron learning algorithm is misclassification driven, specifically using stochastic gradient descent. First, a hyperplane  $w_0, b_0$  is chosen arbitrarily, and then the objective function (2.5) is continuously minimized by the gradient descent method. In the process of minimization, instead of making the gradient of all misclassified points in  $M$  descend at once, it randomly selects one misclassified point to make its gradient descend.

Assuming that  $M$ , the set of a misclassified points, is fixed, the gradient of the loss function  $L(w, b)$  is given by

$$\begin{aligned} \nabla_w L(w, b) &= - \sum_{x_i \in M} y_i x_i \\ \nabla_b L(w, b) &= - \sum_{x_i \in M} y_i \end{aligned}$$

A misclassified point  $(x_i, y_i)$  is randomly selected to update  $w, b$ :

$$w \leftarrow w + \eta y_i x_i \quad (2.6)$$

$$b \leftarrow b + \eta y_i \quad (2.7)$$

The  $\eta(0 < \eta \leq 1)$  in the formula is the step size, also known as the learning rate in machine learning. Thus, the loss function  $L(w, b)$  can be expected to decrease continuously until it reaches 0 by iteration. In summary, the following algorithm is obtained:

**Algorithm 2.1 (the primal form of the perceptron learning algorithm)**

Input: the training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \chi = R^n$ ,  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ ; and the learning rate  $\eta(0 < \eta \leq 1)$ ;

Output:  $w, b$ ; the perceptron model  $f(x) = \text{sign}(w \cdot x + b)$ .

- (1) Select initial values  $w_0, b_0$ ;
- (2) Select data  $(x_i, y_i)$  in the training set;
- (3) If  $y_i(w \cdot x_i + b) \leq 0$ ,

$$\begin{aligned} w &\leftarrow w + \eta y_i x_i \\ b &\leftarrow b + \eta y_i \end{aligned}$$

- (4) Go to (2) until there are no misclassified points in the training set.

This learning algorithm is intuitively explained as follows: when an instance point is misclassified, i.e., on the wrong side of the separating hyperplane, the values of  $w$ ,  $b$  are adjusted to move the separating hyperplane towards the side of the misclassified point, reducing the distance between the misclassified point and the hyperplane until the hyperplane crosses the misclassified point, which is then correctly classified.

Algorithm 2.1 is the basic algorithm of perceptron learning. It corresponds to the dual form and is called the primal form. The perceptron learning algorithm is simple and easy to implement.

**Example 2.1** For the training dataset shown in Fig. 2.2, the positive instance points are  $x_1 = (3, 3)^T$ ,  $x_2 = (4, 3)^T$ , the negative instance point is  $x_3 = (1, 1)^T$ . Try to find the perceptron model  $f(x) = \text{sign}(w \cdot x + b)$  using the primal form of the perceptron learning algorithm. Here,  $w = (w^{(1)}, w^{(2)})^T$ ,  $x = (x^{(1)}, x^{(2)})^T$ .

**Solution** Constructing an optimization problem:

$$\min_{w,b} L(w, b) = - \sum_{x_i \in M} y_i (w \cdot x_i + b)$$

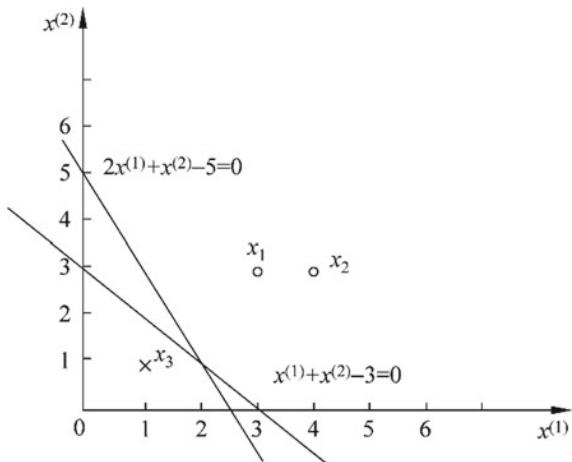
Solve for  $w, b$  according to Algorithm 2.1. Here, we have  $\eta = 1$ .

- (1) Take initial values  $w_0 = 0, b_0 = 0$ .
- (2) For  $x_1 = (3, 3)^T$ ,  $y_1(w_0 \cdot x_1 + b_0) = 0$ , indicating that the point was not correctly classified. We can then update  $w, b$

$$w_1 = w_0 + y_1 x_1 = (3, 3)^T, b_1 = b_0 + y_1 = 1$$

and obtain a linear model

**Fig. 2.2** An example of the perceptron



$$w_1 \cdot x + b_1 = 3x^{(1)} + 3x^{(2)} + 1$$

- (3) For  $x_1, x_2$ , apparently, we have  $y_i(w_1 \cdot x_i + b_1) > 0$ , indicating that the point was correctly classified, therefore,  $w, b$  stay unchanged;

For  $x_3 = (1, 1)^T$ ,  $y_3(w_1 \cdot x_3 + b_1) < 0$ , indicating that the point was misclassified. We can update  $w, b$

$$w_2 = w_1 + y_3 x_3 = (2, 2)^T, b_2 = b_1 + y_3 = 0$$

and obtain a linear model

$$w_2 \cdot x + b_2 = 2x^{(1)} + 2x^{(2)}$$

We continue like this until we have

$$w_7 = (1, 1)^T, b_7 = -3$$

$$w_7 \cdot x + b_7 = x^{(1)} + x^{(2)} - 3$$

For all data points, if  $y_i(w_7 \cdot x_i + b_7) > 0$ , there is no misclassified point, then the loss function reaches a minimum.

The separating hyperplane is:  $x^{(1)} + x^{(2)} - 3 = 0$

The perceptron model is:  $f(x) = \text{sign}(x^{(1)} + x^{(2)} - 3)$

The iterative process can be seen in Table 2.1.

In the computation, the separating hyperplane and the perceptron model are obtained by taking  $x_1, x_3, x_3, x_3, x_1, x_3, x_3$  as the misclassified points in sequence. If

**Table 2.1** The iterative solution of Example 2.1

Number of iterations	Misclassified points	$\omega$	$b$	$w \cdot x + b$
0		0	0	0
1	$x_1$	$(3, 3)^T$	1	$3x^{(1)} + 3x^{(2)} + 1$
2	$x_3$	$(2, 2)^T$	0	$2x^{(1)} + 2x^{(2)}$
3	$x_3$	$(1, 1)^T$	-1	$x^{(1)} + x^{(2)} - 1$
4	$x_3$	$(0, 0)^T$	-2	-2
5	$x_1$	$(3, 3)^T$	-1	$3x^{(1)} + 3x^{(2)} - 1$
6	$x_3$	$(2, 2)^T$	-2	$2x^{(1)} + 2x^{(2)} - 2$
7	$x_3$	$(1, 1)^T$	-3	$x^{(1)} + x^{(2)} - 3$
8	0	$(1, 1)^T$	-3	$x^{(1)} + x^{(2)} - 3$

the misclassified points in the computation are  $x_1, x_3, x_3, x_3, x_2, x_3, x_3, x_1, x_3, x_3$ , then the separating hyperplane obtained is  $2x^{(1)} + x^{(2)} - 5 = 0$ .

It can be seen that the perceptron learning algorithm may have different solutions due to different initial values or different misclassified points.

### 2.3.2 Convergence of the Algorithm

What we need to prove now is that the primal form of the perceptron learning algorithm for the linearly separable dataset is convergent. That is to say, after a finite number of iterations, a perceptron model and a separating hyperplane, which separate the training dataset completely and correctly, can be obtained.

For ease of description and deduction, the bias  $b$  is incorporated into the weight vector  $w$ , denoted as  $\hat{w} = (w^T, b)^T$ , and the input vector is also extended with the constant 1, denoted as  $\hat{x} = (x^T, 1)^T$ . In this way, we have  $\hat{x} \in R^{n+1}$ ,  $\hat{w} \in R^{n+1}$ . Obviously,  $\hat{w} \cdot \hat{x} = w \cdot x + b$ .

#### Theorem 2.1 (Novikoff)

Suppose the training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$  is linearly separable, where  $x_i \in \mathcal{X} = R^n$ ,  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ , then

- (1) There is a hyperplane  $\hat{w}_{opt} \cdot \hat{x} = w_{opt} \cdot x + b_{opt} = 0$  that meets the condition of  $\|\hat{w}_{opt}\| = 1$  to separate the training dataset completely and correctly; and there is  $\gamma > 0$ , for all  $i = 1, 2, \dots, N$

$$y_i(\hat{w}_{opt} \cdot \hat{x}_i) = y_i(w_{opt} \cdot x_i + b_{opt}) \geq \gamma \quad (2.8)$$

- (2) Let  $R = \max_{1 \leq i \leq N} \|\hat{x}_i\|$ , then  $k$ , the number of misclassification of the perceptron Algorithm 2.1 on the training dataset satisfies the inequality

$$k \leq \left( \frac{R}{\gamma} \right)^2 \quad (2.9)$$

**Proof** (1) Since the training dataset is linearly separable, according to Definition 2.2, there is a hyperplane that separates the training dataset completely and correctly. Take this hyperplane to be  $\hat{w}_{opt} \cdot \hat{x} = w_{opt} \cdot x_i + b_{opt} = 0$  to satisfy  $\|\hat{w}_{opt}\| = 1$ . For finite  $i = 1, 2, \dots, N$ , there is

$$y_i(\hat{w}_{opt} \cdot \hat{x}_i) = y_i(w_{opt} \cdot x_i + b_{opt}) > 0$$

Therefore, there is

$$\gamma = \min_i \{y_i(w_{opt} \cdot x_i + b_{opt})\}$$

making

$$y_i(\hat{w}_{opt} \cdot \hat{x}_i) = y_i(w_{opt} \cdot x_i + b_{opt}) \geq \gamma$$

(2) The perceptron algorithm starts with  $\hat{w}_0 = 0$  and updates the weight if the instance is misclassified. Let  $\hat{w}_{k-1}$  be the extended weight vector before the  $k$ -th misclassified instance, i.e.

$$\hat{w}_{k-1} = (w_{k-1}^T, b_{k-1})^T$$

Then the condition of the  $k$ -th misclassified instance is

$$y_i(\hat{w}_{k-1} \cdot \hat{x}_i) = y_i(w_{k-1} \cdot x_i + b_{k-1}) \leq 0 \quad (2.10)$$

If  $(x_i, y_i)$  is the data misclassified by  $\hat{w}_{k-1} = (w_{k-1}^T, b_{k-1})^T$ , then the update of  $w$  and  $b$  are

$$w_k \leftarrow w_{k-1} + \eta y_i x_i$$

$$b_k \leftarrow b_{k-1} + \eta y_i$$

i.e.,

$$\hat{w}_k = \hat{w}_{k-1} + \eta y_i \hat{x}_i \quad (2.11)$$

The two inequalities (2.12) and (2.13) are derived as follows:

$$\hat{w}_k \cdot \hat{w}_{opt} \geq k \eta \gamma \quad (2.12)$$

From Eqs. (2.11) and (2.8), we have

$$\hat{w}_k \cdot \hat{w}_{opt} = \hat{w}_{k-1} \cdot \hat{w}_{opt} + \eta y_i \hat{w}_{opt} \cdot \hat{x}_i \geq \hat{w}_{k-1} \cdot \hat{w}_{opt} + \eta \gamma$$

Then Inequality (2.12) is obtained by this recursion

$$\hat{w}_k \cdot \hat{w}_{opt} \geq \hat{w}_{k-1} \cdot \hat{w}_{opt} + \eta \gamma \geq \hat{w}_{k-2} \cdot \hat{w}_{opt} + 2\eta \gamma \geq \dots \geq k\eta \gamma$$

$$\|\hat{w}_k\|^2 \leq k\eta^2 R^2 \quad (2.13)$$

From Eqs. (2.11) and (2.10), we have

$$\begin{aligned} \|\hat{w}_k\|^2 &= \|\hat{w}_{k-1}\|^2 + 2\eta y_i \hat{w}_{k-1} \cdot \hat{x}_i + \eta^2 \|\hat{x}_i\|^2 \\ &\leq \|\hat{w}_{k-1}\|^2 + \eta^2 \|\hat{x}_i\|^2 \\ &\leq \|\hat{w}_{k-1}\|^2 + \eta^2 R^2 \\ &\leq \|\hat{w}_{k-2}\|^2 + 2\eta^2 R^2 \leq \dots \\ &\leq k\eta^2 R^2 \end{aligned}$$

Combining Inequality (2.12) and Eq. (2.13), we have

$$k\eta \gamma \leq \hat{w}_k \cdot \hat{w}_{opt} \leq \|\hat{w}_k\| \|\hat{w}_{opt}\| \leq \sqrt{k} \eta R$$

$$k^2 \gamma^2 \leq k R^2$$

Thus

$$k \leq \left( \frac{R}{\gamma} \right)^2$$

The theorem shows that the number of misclassifications  $k$  has an upper bound, and a separating hyperplane that separates the training data completely and correctly can be found after a finite number of searches. That is, the primal iteration of the perceptron learning algorithm is convergent when the training dataset is linearly separable. However, Example 2.1 illustrates many solutions of the perceptron learning algorithm. They depend on the choice of initial values and the order in which the misclassified points are selected during the iteration. Consequently, constraints need to be added to the separating hyperplane to obtain the unique hyperplane. This is the idea of linear Support Vector Machine (SVM) that will be described in Chap. 7. When the training set is linearly inseparable, the perceptron learning algorithm does not converge, and the results of iterations will oscillate.

### 2.3.3 The Dual Form of the Perceptron Learning Algorithm

Now we will discuss the dual form of the perceptron learning algorithm. The perceptron learning algorithm's primal and dual forms correspond to the SVM learning algorithm's primal and dual forms introduced in Chap. 7.

The basic idea of the dual form is that  $w$  and  $b$  are represented as the linear combination of instance  $x_i$  and tag  $y_i$ , then  $w$  and  $b$  can be obtained by solving their coefficients. For the sake of generality, the initial values  $w_0$  and  $b_0$  can be assumed to be 0 in Algorithm 2.1. Modify  $w$  and  $b$  progressively on misclassification point  $(x_i, y_i)$  via

$$\begin{aligned} w &\leftarrow w + \eta y_i x_i \\ b &\leftarrow b + \eta y_i \end{aligned}$$

Assuming that they are modified  $n$  times, then the increments of  $w$  and  $b$  with respect to  $(x_i, y_i)$  are  $\alpha_i y_i x_i$  and  $\alpha_i y_i$ , respectively. Here  $\alpha_i = n_i \eta$ , and  $n_i$  is the number of times that the point  $(x_i, y_i)$  is misclassified. Thus, as seen from the learning process, the learned  $w$  and  $b$  can be expressed as

$$w = \sum_{i=1}^N \alpha_i y_i x_i \quad (2.14)$$

$$b = \sum_{i=1}^N \alpha_i y_i \quad (2.15)$$

Here,  $\alpha_i \geq 0$ ,  $i = 1, 2, \dots, N$ . When  $\eta = 1$ ,  $\alpha_i$  indicates the number of times the  $i$ -th instance point is updated due to misclassification. The more times the instance point is updated, the closer it is to the separating hyperplane, and the more difficult it is to classify correctly. In other words, such instances have the greatest impact on learning results.

The dual form of the perceptron learning algorithm is described below by reference to the primal form.

#### Algorithm 2.2 (the dual form of perceptron learning algorithm)

Input: A linearly separable dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in R^n$ ,  $y_i \in \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ ; and the learning rate  $\eta (0 < \eta \leq 1)$ ;

Output:  $\alpha, b$ ; the perceptron model  $f(x) = sign\left(\sum_{j=1}^N \alpha_j y_j x_j \cdot x + b\right)$ , where  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_N)^T$ .

- (1)  $\alpha \leftarrow 0, b \leftarrow 0$ ;
- (2) Select data  $(x_i, y_i)$  from the training set;

- (3) If  $y_i \left( \sum_{j=1}^N \alpha_j y_j x_j \cdot x_i + b \right) \leq 0$ , then

$$\begin{aligned}\alpha_i &\leftarrow \alpha_i + \eta \\ b &\leftarrow b + \eta y_i\end{aligned}$$

- (4) Go to (2) until there is no misclassification data

The training instances in the dual form only appear as inner products. For convenience, the inner products between instances in the training set can be computed in advance and stored in the form of a matrix, the so-called Gram matrix.

$$G = [x_i \cdot x_j]_{N \times N}$$

**Example 2.2** The data are the same as in Example 2.1. The positive sample points are  $x_1 = (3, 3)^T$ ,  $x_2 = (4, 3)^T$ , and the negative sample point is  $x_3 = (1, 1)^T$ . Try to use the dual form of the perceptron learning algorithm to find the perceptron model.

**Solution** According to Algorithm 2.2,

- (1) Take  $\alpha_i = 0$ ,  $i = 1, 2, 3$ ,  $b = 0$ ,  $\eta = 1$ ;  
 (2) compute Gram matrix

$$G = \begin{bmatrix} 18 & 21 & 6 \\ 21 & 25 & 7 \\ 6 & 7 & 2 \end{bmatrix}$$

- (3) The misclassification condition is

$$y_i \left( \sum_{j=1}^N \alpha_j y_j x_j \cdot x_i + b \right) \leq 0$$

Update parameters

$$\alpha_i \leftarrow \alpha_i + 1, b \leftarrow b + y_i$$

- (4) Iteration. The procedure is omitted, and the results are shown in Table 2.2.  
 (5)  $w = 2x_1 + 0x_2 - 5x_3 = (1, 1)^T$   
 $b = -3$

The separating hyperplane is

$$x^{(1)} + x^{(2)} - 3 = 0$$

**Table 2.2** The iterative solution of Example 2.2

$k$	0	1	2	3	4	5	6	7
		$x_1$	$x_3$	$x_3$	$x_3$	$x_1$	$x_3$	$x_3$
$\alpha_1$	0	1	1	1	1	2	2	2
$\alpha_2$	0	0	0	0	0	0	0	0
$\alpha_3$	0	0	1	2	3	3	4	5
$b$	0	1	0	-1	-2	-1	-2	-3

The perceptron model is

$$f(x) = \text{sign}(x^{(1)} + x^{(2)} - 3)$$

Compared with Example 2.1, the results are consistent, and the iteration steps correspond.

Same as the primal form, the dual iteration of the perceptron learning algorithm is convergent and has multiple solutions.

## Summary

1. A perceptron is a linear classification model that classifies input instances into two classes according to their feature vectors  $x$ :

$$f(x) = \text{sign}(w \cdot x + b)$$

The perceptron model corresponds to the separating hyperplane  $w \cdot x + b = 0$  in the input space (feature space).

2. The strategy of the perceptron learning is to minimize the loss function:

$$\min_{w,b} L(w, b) = - \sum_{x_i \in M} y_i (w \cdot x_i + b)$$

The loss function corresponds to the total distance from the misclassified point to the separating hyperplane.

3. The perceptron learning algorithm is an optimization algorithm for the loss function based on stochastic gradient descent and is available in both primal and dual forms. The algorithm is simple and easy to implement. In the primal form, an arbitrary hyperplane is first selected, and then the objective function is continuously minimized by gradient descent. In this process, one misclassified point is randomly selected at a time to descend its gradient.
4. The perceptron learning algorithm is convergent when the training dataset is linearly separable. The number of misclassification of the perceptron algorithm on the training dataset,  $k$ , satisfies the inequality:

$$k \leq \left(\frac{R}{\gamma}\right)^2$$

When the training dataset is linearly separable, the perceptron learning algorithm has an infinite number of solutions, which may vary due to different initial values or different iteration orders.

## Further Reading

The perceptron was first proposed by Rosenblatt in 1957 [1] and has been the subject of several theoretical studies by Novikoff [2], Minsky and Papert [3] and others. Extended learning methods for perceptron include the pocket algorithm [4], the voted perceptron [5], and the perceptron with margin [6]. More introduction to the perceptron can be found in references [7, 8].

## Exercises

- 2.1 Minsky and Papert pointed out that, since the perceptron is a linear model, it cannot represent complex functions such as exclusive OR (XOR). Verify why the perceptron cannot represent XOR.
- 2.2 Imitate Example 2.1 and construct an example of solving a perceptron model from a training dataset.
- 2.3 Prove the following theorem: the necessary and sufficient condition for the sample set to be linearly separable is that the convex hulls<sup>2</sup> formed by the set of positive instance points do not intersect with the convex hulls formed by the set of negative instance points.

## References

1. Rosenblatt F. The Perceptron: a probabilistic model for information storage and organization in the Brain. Cornell aeronautical laboratory. Psychol Rev. 1958;65(6):386–408.
2. Novikoff AB. On convergence proofs on perceptrons. Symposium on the mathematical theory of automata. Polytech Inst Brooklyn. 1962;12:615–22.
3. Minsky ML, Papert SA. Perceptrons. Cambridge, MA: MIT Press; 1969.
4. Gallant SI. Perceptron-based learning algorithms. IEEE Trans Neural Netw. 1990;1(2):179–91.
5. Freund Y, Schapire RE. Large margin classification using the perceptron algorithm. In: Proceedings of the 11th annual conference on computational learning theory (COLT' 98). ACM Press; 1998.
6. Li YY, Zaragoza H, Herbrich R et al. The perceptron algorithm with uneven margins. In: Proceedings of the 19th international conference on machine learning; 2002, pp. 379–386.

---

<sup>2</sup> Let the set  $S \subset R^n$  be a set of  $k$  points in  $R^n$ , i.e.,  $S = \{x_1, x_2, \dots, x_k\}$ . Define convex hull  $\text{conv}(S)$  of  $S$  as.

$$\text{conv}(S) = \left\{ x = \sum_{i=1}^k \lambda_i x_i \middle| \sum_{i=1}^k \lambda_i = 1, \lambda_i \geq 0, i = 1, 2, \dots, k \right\}$$

7. Widrow B, Lehr MA. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. Proc IEEE. 1990;78(9):1415–42.
8. Cristianini N, Shawe-Taylor J. An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press; 2000.

# Chapter 3

## K-Nearest Neighbor

*k*-Nearest-Neighbor (*k*-NN), proposed by Cover and Hart in 1968, is a basic classification and regression method. This book deals only with the *k*-NN algorithm in classification. The input of *k*-NN is the instance's feature vector, corresponding to the point in the feature space. The output is the instance's class, which can be taken as multiple. *K*-NN assumes that the instance's class is determined for a given training dataset. When classifying a new instance, it is predicted by majority voting and other methods based on the class of *k* training instances nearest to it. Therefore, *k*-NN does not have an explicit learning process. It uses the training dataset to separate the feature vector space and serves as a “model” for its classification. The choice of *k* value, distance metrics and classification decision rules are the three basic elements of *k*-NN.

This chapter first describes the *k*-NN algorithm, then discusses the model and three basic elements of *k*-NN, and finally describes an implementation method of *k*-NN—the *kd*-tree, focusing on algorithms for constructing and searching the *kd*-tree.

### 3.1 The K-Nearest Neighbor Algorithm

The *k*-NN algorithm is simple and intuitive. A training dataset is first given when classifying a new input instance. The next step is finding *k* instances nearest to it in the training dataset. If the majority of these *k* nearest instances belong to a certain class, the new input instance will be classified into this class. The *k*-NN algorithm is described below, followed by a discussion of its details.

**Algorithm 3.1 (*k*-Nearest Neighbor)** Input: Training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i \in \mathcal{X} \subseteq R^n$  is the feature vector of the instance,  $y_i \in \mathcal{Y} = \{c_1, c_2, \dots, c_k\}$  is the class of the instance,  $i = 1, 2, \dots, N$ ;  $x$  is the instance feature vector;

Output:  $y$ , the class to which the instance  $x$  belongs.

- (1) According to the given distance metric,  $k$  points nearest to  $x$  are found in the training set  $T$ , and the neighborhood of point  $x$  covering these  $k$  points is denoted as  $N_k(x)$ ;
- (2) Determine  $y$ , the class of  $x$  in  $N_k(x)$  according to classification decision rules (such as majority voting):

$$y = \arg \max_{c_j} \sum_{x_i \in N_k(x)} I(y_i = c_j), i = 1, 2, \dots, N; j = 1, 2, \dots, K \quad (3.1)$$

In Formula (3.1),  $I$  is the indicator function, i.e.,  $I$  is 1 when  $y_i = c_j$ ; otherwise,  $I$  is 0.

The special case of  $k$ -NN is the situation when  $k = 1$ , known as the nearest neighbor algorithm. For the input instance point (feature vector)  $x$ , the nearest neighbor algorithm takes the class of the nearest point to  $x$  in the training dataset as the class of  $x$ .

The  $K$ -NN algorithm has no explicit learning process.

## 3.2 The $K$ -Nearest Neighbor Model

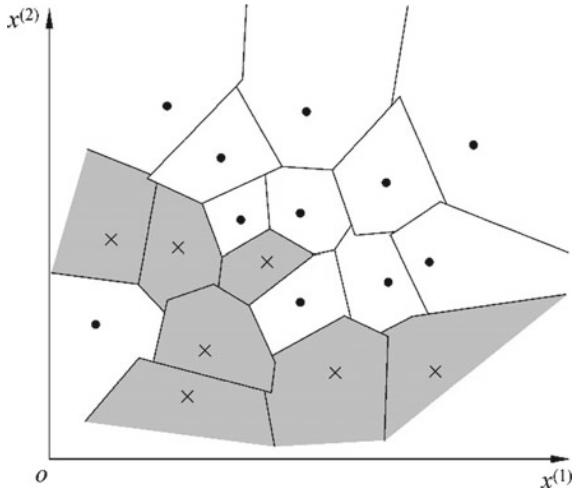
The model used by  $k$ -NN actually corresponds to the partition of the feature space. The model is determined by three basic elements—distance metrics, the choice of  $k$  value and classification decision rules.

### 3.2.1 Model

In  $k$ -NN, a new input instance's class is uniquely determined when the training set, distance metrics (such as Euclidean distance),  $k$  value and classification decision rules (such as majority voting rule) are determined. It is equivalent to dividing the feature space into some subspaces based on the above elements and determining the class to which each point in the subspace belongs. This fact can be seen very clearly from the nearest neighbor algorithm.

In the feature space, for each training instance point  $x_i$ , all points closer to this point than other points form a region called a cell. Each training instance point has a cell, and cells of all training instance points constitute a partition of the feature space. The nearest neighbor algorithm takes  $y_i$ , the class of the instance  $x_i$  as the class label of all points in its cell. In this way, the class of each cell's instance point is determined. Figure 3.1 depicts an example of the two-dimensional feature space partition.

**Fig. 3.1** A partition of the feature space corresponding to the  $k$ -NN mode



### 3.2.2 Distance Metrics

The distance between two instance points in the feature space reflects their similarity. The feature space of the  $k$ -NN model is generally the  $n$ -dimensional real vector space  $\mathbf{R}^n$ . The distance used is Euclidean distance, but it can also be other distances, such as the more general  $L_p$  distance or Minkowski distance.

Suppose the feature space  $\mathcal{X}$  is an  $n$ -dimensional real vector space  $\mathbf{R}^n$ ,  $x_i, x_i \in \mathcal{X}, x_i = \left( x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)} \right)^T, x_j = \left( x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(n)} \right)^T$  and the  $L_p$  distance of  $x_i, x_j$  is defined as

$$L_p(x_i, x_j) = \left( \sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^p \right)^{\frac{1}{p}} \quad (3.2)$$

Here  $p \geq 1$ . When  $p = 2$ , it is called Euclidean distance, i.e.,

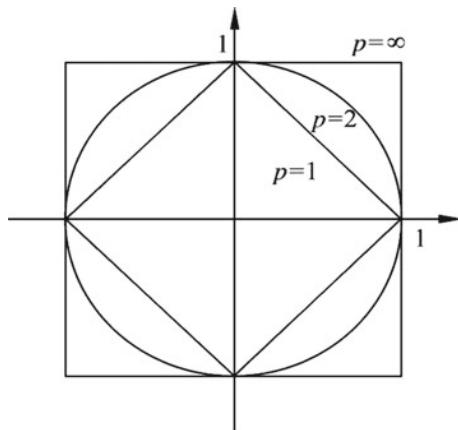
$$L_2(x_i, x_j) = \left( \sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^2 \right)^{\frac{1}{2}} \quad (3.3)$$

When  $p = 1$ , it is called Manhattan distance, i.e.,

$$L_1(x_i, x_j) = \sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}| \quad (3.4)$$

When  $p = \infty$ , it is the maximum value of each coordinate distance, i.e.,

**Fig. 3.2** The relationship of  $L_p$  distance



$$L_\infty(x_i, x_j) = \max_l |x_i^{(l)} - x_j^{(l)}| \quad (3.5)$$

Figure 3.2 shows the graph of points whose  $L_p$  distance to the origin is 1 ( $L_p = 1$ ) when  $p$  takes different values in the two-dimensional space.

The following example shows that the nearest neighbors determined by different distance metrics are different.

**Example 3.1** Given 3 points  $x_1 = (1, 1)^T$ ,  $x_2 = (5, 1)^T$ ,  $x_3 = (4, 4)^T$  in a two-dimensional space, try to find the nearest neighbor of  $x_1$  at  $L_p$  distance when  $p$  takes different values.

**Solution** Since  $x_1$  and  $x_2$  differ only in the first dimension,  $L_p(x_1, x_2) = 4$  holds for any value of  $p$ . We also have  $L_1(x_1, x_3) = 6$ ,  $L_2(x_1, x_3) = 4.24$ ,  $L_3(x_1, x_3) = 3.78$ ,  $L_4(x_1, x_3) = 3.57$ . Thus, when  $p$  is equal to 1 or 2,  $x_2$  is the nearest neighbor of  $x_1$ ; when  $p$  is greater than or equal to 3,  $x_3$  is the nearest neighbor of  $x_1$ .

### 3.2.3 The Selection of k Value

The choice of  $k$  value will have a significant impact on the results of  $k$ -NN.

If a smaller value of  $k$  is chosen, it is equivalent to predicated with training instances in the smaller neighborhood. Accordingly, the approximation error of “learning” will be reduced, and only training instances that are closer (similar) to the input instance will contribute to the prediction. The disadvantage is that the estimation error of “learning” will increase, and the prediction result will be very sensitive to those nearest instance points [2]. If the neighboring instance points happen to be noise, the prediction will be wrong. In other words, decreasing the  $k$  value means that the holistic model becomes complicated and is prone to overfitting.

If a larger value of  $k$  is chosen, it is equivalent to predicting with the training instances in the larger neighborhood. It has the advantage of reducing the estimation error of learning, while the disadvantage is that the approximation error of learning will increase. At this time, training instances far (dissimilar) from the input instance will also affect the prediction, making the prediction wrong. The increase of  $k$  value means that the whole model becomes simpler.

If  $k = N$ , then whatever the input instance is, it will simply be predicted that this instance belongs to the class with the most training instances. In this case, the model is too simple and completely ignores a large amount of useful information in the training instances, which is undesirable.

In applications, the value of  $k$  is generally taken as a relatively small value, and the cross-validation method is usually used to select the optimal  $k$  value.

### 3.2.4 Classification Decision Rule

The classification decision rule in  $k$ -NN is usually the majority voting, i.e., the majority class of  $k$  neighboring training instances around the input instance determines the class of it.

The majority voting rule is interpreted as follows: if the classification loss function is 0–1 the loss function, the classification function is

$$f : \mathbf{R}^n \rightarrow \{c_1, c_2, \dots, c_K\}$$

Then the probability of misclassification is

$$P(Y \neq f(X)) = 1 - P(Y = f(X))$$

For a given instance  $x \in \mathcal{X}$ , its  $k$ -NN training instance points form a set  $N_k(x)$ . If the class of the region covering  $N_k(x)$  is  $c_j$ , then the misclassification rate is

$$\frac{1}{k} \sum_{x_i \in N_k(x)} I(y_i \neq c_j) = 1 - \frac{1}{k} \sum_{x_i \in N_k(x)} I(y_i = c_j)$$

To minimize the misclassification rate (i.e., minimize the empirical risk), the  $\sum_{x_i \in N_k(x)} I(y_i = c_j)$  must be maximized, so the majority voting rule is equivalent to minimizing the empirical risk.

### 3.3 Implementation of K-Nearest Neighbor: The *kd*-Tree

When implementing  $k$ -NN, the primary concern is how to perform a fast  $k$ -NN search on the training data. It is indispensable when the feature space dimensionality and the training data capacity are large.

The simplest implementation method of  $k$ -NN is the linear scan, which involves computing the distance between the input instances and each of the training instances. When the training set is large, the computation is time-consuming and this method is not feasible.

To improve the efficiency of  $k$ -NN search, we can consider using special structures to store training data and reduce distance computation. Among many specific methods, the *kd*-tree method<sup>1</sup> is introduced below.

#### 3.3.1 Constructing the *kd*-Tree

The *kd*-tree is a tree data structure that stores instance points in  $k$ -dimensional space for quick retrieval. It is a binary tree representing a partition of the  $k$ -dimensional space. Constructing a *kd*-tree is equivalent to continuously splitting the  $k$ -dimensional space with a hyperplane perpendicular to the coordinate axis and forming a series of  $k$ -dimensional hyperrectangular regions. Each node of the *kd*-tree corresponds to a  $k$ -dimensional hyperrectangular region.

The *kd*-tree is constructed as follows. The root node is constructed so that it corresponds to a hyper-rectangular region in  $k$ -dimensional space containing all the instance points; the  $k$ -dimensional space is continually separated by the following recursive method to generate child nodes. Then a hyperplane is determined by selecting a coordinate axis along with a splitting point on it in the hyperrectangular region (node). It passes through the selected tangent point and is perpendicular to the selected axis, splitting the current hyper-rectangular region into two sub-regions (child nodes), left and right; instances are then divided into these two sub-regions. This process terminates when there are no instances in the sub-region (the terminating node is the leaf node). In this process, instances are stored on the corresponding node.

Generally, the coordinate axis is selected successively to separate the space, and the median<sup>2</sup> of training instance points on the chosen coordinate axis is selected as the splitting point so that the obtained *kd*-tree is balanced. Note that the efficiency of searching with a balanced *kd*-tree is not necessarily optimal.

The algorithm for constructing the *kd*-tree is given below.

<sup>1</sup> The *kd*-tree is a tree structure for storing  $k$ -dimensional spatial data. The meaning of  $k$  here is different from that of  $k$  in  $k$ -NN. To be consistent with custom, the name of the *kd*-tree is still used in this book.

<sup>2</sup> The number or the average of the two numbers in the middle of a set of data arranged in ascending order.

**Algorithm 3.2 (Constructing the Balanced  $kd$ -Tree)** Input: The  $k$ -dimensional spatial dataset  $T = \{x_1, x_2, \dots, x_N\}$ , where  $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(k)})^T$ ,  $i = 1, 2, \dots, N$ ;

Output: the  $kd$ -tree.

- (1) Start: construct the root node, which corresponds to the hyperrectangular region of the  $k$ -dimensional space containing  $T$ .

Select  $x^{(1)}$  as the coordinate axis, take the median of  $x^{(1)}$  coordinates of all instances in  $T$  as the splitting point and separate the hyperrectangular region corresponding to the root node into two sub-regions. The splitting is achieved by the hyperplane passing through the splitting point and perpendicular to the coordinate axis  $x^{(1)}$ .

Left and right child nodes with depth of 1 are generated from the root node. The left child node corresponds to a sub-region whose coordinate  $x^{(1)}$  is smaller than the splitting point, and the right child node corresponds to a sub-region whose coordinate  $x^{(1)}$  is greater than the splitting point.

Save instance points falling on the splitting hyperplane at the root node.

- (2) Repeat: For the node with the depth of  $j$ ,  $x^{(l)}$  is selected as the coordinate axis of splitting,  $l = j \pmod k + 1$ . The median of all instances'  $x^{(l)}$  coordinates in the region of this node is taken as the splitting point, and the hyperrectangular region corresponding to this node is separated into two sub-regions. The splitting is realized by the hyperplane passing through the splitting point and perpendicular to the coordinate axis  $x^{(l)}$ .

Left and right child nodes with the depth of  $j + 1$  are generated from this node. The corresponding coordinate  $x^{(l)}$  of the left child node is smaller than the sub-region of the splitting point, and the corresponding coordinate  $x^{(l)}$  of the right child node is greater than the sub-region of the splitting point.

Save instance points falling on the splitting hyperplane at this node.

- (3) Stop when there is no instance in these two sub-regions. Thus, the region partition of  $kd$ -tree is formed.

**Example 3.2** Given a dataset in a two-dimensional space:

$$T = \{(2, 3)^T, (5, 4)^T, (9, 6)^T, (4, 7)^T, (8, 1)^T, (7, 2)^T\}$$

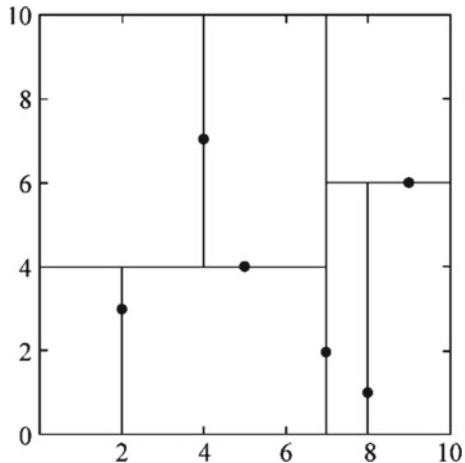
Construct a balanced  $kd$ -tree.<sup>3</sup>

**Solution** The root node corresponds to the rectangle containing dataset  $T$ . Select the  $x^{(1)}$  axis, and the median of  $x^{(1)}$  coordinates of six data points is 7.<sup>4</sup> Take the plane  $x^{(1)} = 7$  to divide the space into left and right sub-rectangles (child nodes); then, the left rectangle is divided by  $x^{(2)} = 4$  into two sub-rectangles, and the right rectangle

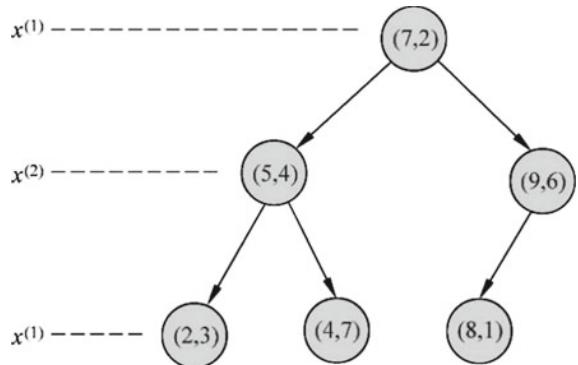
<sup>3</sup> From Wikipedia.

<sup>4</sup>  $x^{(1)} = 6$  is the median, but  $x^{(2)} = 7$  is selected because there are no data points on  $x^{(1)} = 6$ .

**Fig. 3.3** Feature space partition



**Fig. 3.4** An example of *kd*-tree



is divided by  $x^{(2)} = 6$  into two sub-rectangles. Finally, a recursion like this helps to obtain the feature space partition shown in Fig. 3.3 and the *kd*-tree shown in Fig. 3.4.

### 3.3.2 Searching for kd-Tree

The following describes using the *kd*-tree to perform the *k*-NN search. As we can see, by using the *kd*-tree, the search for most data points can be avoided, and the computation of searching can be reduced. Here we take the nearest neighbor as an example, the same method can be applied to *k*-NN.

Given a target point, search for its nearest neighbor. First, find the leaf node that contains the target point; then start from the leaf node and go back to the parent node in turn; continue to search for the node nearest to the target point, and stop when it

is determined that there is no closer node. In this way, the search is restricted to a local area of the space, and the efficiency is greatly improved.

The leaf node containing the target point corresponds to the smallest hyperrectangular region containing the target point. Take the instance point of this leaf node as the current nearest point. The nearest neighbor of the target point must be inside the hypersphere centered on the target point and passing through the current nearest point (see Fig. 3.5). Then return to the parent node of the current node, if the hyperrectangular region of the other child node of the parent node intersects the hypersphere, then find the instance point nearer to the target point within the intersecting region. If such a point exists, this point is taken as the new current nearest point. The algorithm goes to the parent node of a higher level and continues the above process. The search stops if the hyperrectangular region of another child node belonging to the parent node does not intersect the hypersphere or if there is no point nearer than the current nearest point.

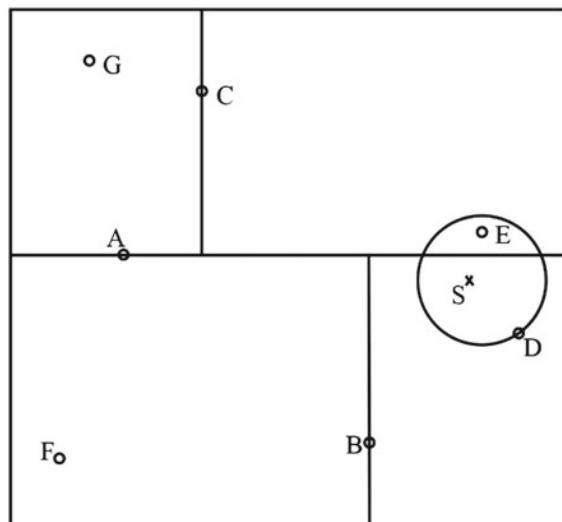
The following describes the nearest neighbor search algorithm using  $kd$ -tree.

**Algorithm 3.3 (Searching with the Nearest Neighbor of  $kd$ -Tree)** Input: the constructed  $kd$ -tree, target point  $x$ .

Output: the nearest neighbor of  $x$ .

- (1) Find the leaf node in the  $kd$ -tree that contains the target point  $x$ . Start from the root node and recursively access the  $kd$ -tree downwards. If the coordinate of the target point  $x$  in the current dimension is smaller than the coordinate of the splitting point, move to the left child node; otherwise, move to the right one. The above process goes on until the child node is a leaf node.
- (2) Take the leaf node as the “current nearest point”.

**Fig. 3.5** Searching for the nearest neighbor by  $kd$ -tree



- (3) Perform recursive fallback upwards and then perform the following operations at each node:
- If the node holds an instance point closer to the target than the current nearest point, then the instance point is considered the “current nearest point”.
  - The current nearest point must exist in the region corresponding to one of its child nodes. Check whether there is a nearer point in the region corresponding to the other child node of the parent node. Specifically, check whether the region corresponding to the other child node intersects a hypersphere whose center is the target point and radius is the distance between the target point and the current nearest point.

If they intersect, there may be a point nearer to the target in the region corresponding to the other child node. Move to this node and recursively perform the nearest neighbor search.

Otherwise, perform fallback upwards.

- (4) The search is over when the root node is reached. The last “current nearest point” is the nearest neighbor of  $x$ .

If instance points are randomly distributed, the average computational complexity of the  $kd$ -tree search is  $O(\log N)$ , where  $N$  is the number of training instances.  $kd$ -tree is more suitable for  $k$ -NN search when the number of training instances is much larger than the space dimension. When the space dimension is close to the number of training instances, its efficiency decreases rapidly, almost approaching the linear scan.

Here is an example to illustrate the search method.

**Example 3.3** Given a  $kd$ -tree as shown in Fig. 3.5, the root node is  $A$ , whose child nodes are  $B$ ,  $C$ , etc. A total of 7 instances are stored in the tree; besides, there is an input target instance  $S$ . Find the nearest neighbor of  $S$ .

**Solution** First, find the leaf node  $D$  (the lower right region in the figure) containing point  $S$  in the  $kd$ -tree, and take point  $D$  as the approximate nearest neighbor. The true nearest neighbor must be inside the circle centered at point  $S$  and passing through point  $D$ . Then return to the parent node  $B$  of node  $D$ , and search for the nearest neighbor in the region of the other child node  $F$  (of node  $B$ ). The region of node  $F$  does not intersect the circle, so there is no nearest neighbor point in it. Continue back to the previous parent node  $A$ , and search for the nearest neighbor in the region of node  $A$ ’s another child,  $C$ . The region of node  $C$  intersects the circle; The instance point of the region in the circle is point  $E$ , which is closer than point  $D$  and becomes the new nearest neighbor. Finally, point  $E$  is the nearest neighbor of point  $S$ .

## Summary

1.  $k$ -NN is an essential and simple classification and regression method. The basic practice of  $k$ -NN is as follows. For the given training instance points and input

- instance points, first, determine the  $k$  nearest training instance points of the input instance, and then use the majority of the classes of these  $k$  training instances to predict the class of the input instance.
2. The  $k$ -NN model corresponds to a partition of the feature space based on the training dataset. The result of  $k$ -NN is uniquely determined when the training set, distance metric,  $k$ -value, and classification decision rule are determined.
  3. The three elements of  $k$ -NN are the distance metric, the selection of  $k$ -value, and the classification decision rule. Commonly used distance metrics are Euclidean distance and the more general  $L_p$  distance. When  $k$  value is small, the  $k$ -NN model is more complex; when  $k$  value is large, the  $k$ -NN model is simpler. The choice of  $k$  value reflects the trade-off between the approximation error and the estimation error, and the optimal  $k$  is usually chosen by cross-validation. The commonly used classification decision rule is majority voting, corresponding to empirical risk minimization.
  4. Implementing  $k$ -NN requires a quick search for  $k$  nearest neighbors.  $kd$ -tree is a data structure that facilitates fast data retrieval in a  $k$ -dimensional space. It is a binary tree that represents a partition of a  $k$ -dimensional space, with each node corresponding to a hyperrectangular region of the  $k$ -dimensional space partition.  $kd$ -tree can be used to reduce the computation by avoiding searching for most data points.

## Further Reading

The  $k$ -Nearest Neighbor method was proposed by Cover and Hart [1]. The  $k$ -NN theory has been discussed in Refs. [2, 3]. Extensions of  $k$ -NN are described in Ref. [4].  $kd$ -tree and other fast search algorithms can be found in Ref. [5]. The introduction to  $k$ -NN can be found in Ref. [2].

## Exercises

- 3.1 Referring to Fig. 3.1, give the instance points in the two-dimensional space, draw the space partition formed by  $k$ -NN when  $k$  is 1 and 2, and compare the results to understand the relationship between the choice of  $k$  value and the model complexity along with prediction accuracy.
- 3.2 Find the nearest point of  $x = (3, 4.5)^T$  using the  $kd$ -tree constructed in Example 3.2.
- 3.3 Referring to Algorithm 3.3, write out an algorithm whose output is the  $k$  nearest neighbor of  $x$ .

## References

1. Cover T, Hart P. Nearest neighbor pattern classification. IEEE Trans Inf Theory. 1967;13(1):21–7.

2. Hastie T, Tibshirani R, Friedman JH. The elements of statistical learning: data mining, inference, and prediction; 2001.
3. Friedman J. Flexible metric nearest neighbor classification. Technical Report; 1994.
4. Weinberger KQ, Blitzer J, Saul LK. Distance metric learning for large margin nearest neighbor classification. In: Proceedings of the NIPS; 2005.
5. Samet H. The design and analysis of spatial data structures. Reading, MA: Addison Wesley; 1990.

# Chapter 4

## The Naïve Bayes Method

The Naïve Bayes method is a classification method based on the Bayes theorem and conditional independence assumption of features.<sup>1</sup> For a given training dataset, the joint probability distribution of inputs and outputs is first learned based on the features' conditional independence assumption. Then, based on this model, the output  $y$  with the maximum posterior probability is found for a given input  $x$  by the Bayes theorem. The Naïve Bayes method is commonly used because of its simplicity of implementation and high efficiency in learning and prediction.

This chapter describes the Naïve Bayes method, including the learning and classification of Naïve Bayes and parameter estimation algorithms for Naïve Bayes.

### 4.1 The Learning and Classification of Naïve Bayes

#### 4.1.1 Basic Methods

Let the input space  $\mathcal{X} \subseteq \mathbf{R}^n$  be a set of  $n$ -dimensional vectors and the output space be a class label collection  $\mathcal{Y} = \{c_1, c_2, \dots, c_K\}$ . The input is a feature vector  $x \in \mathcal{X}$ , and the output is class label  $y \in \mathcal{Y}$ .  $X$  is a random vector defined on the input space  $\mathcal{X}$ , and  $Y$  is a random variable defined on the output space  $\mathcal{Y}$ .  $P(X, Y)$  is the joint probability distribution of  $X$  and  $Y$ . The training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

is generated by  $P(X, Y)$ , independent identically distributed.

Naïve Bayes learns the joint probability distribution  $P(X, Y)$  from the training dataset. Specifically, the following prior probability distribution and conditional probability distribution are learned. The prior probability distribution is

---

<sup>1</sup> Note: Naïve Bayes and Bayesian Estimation are different concepts.

$$P(Y = c_k), \quad k = 1, 2, \dots, K \quad (4.1)$$

The conditional probability distribution is

$$P(X = x|Y = c_k) = P(X^{(1)} = x^{(1)}, \dots, X^{(n)} = x^{(n)}|Y = c_k), \quad k = 1, 2, \dots, K \quad (4.2)$$

Then the joint probability distribution  $P(X, Y)$  is learned.

The conditional probability distribution  $P(X = x|Y = c_k)$  has an exponential number of parameters, so its estimation is practically infeasible. In fact, assume that  $x^{(j)}$  has  $S_j$  possible values,  $j = 1, 2, \dots, n$ , and  $Y$  has  $K$  possible values, then the number of parameters is  $K \prod_{j=1}^n S_j$ .

Naïve Bayes assumes conditional independence on the conditional probability distribution. Since this is a strong assumption, Naïve Bayes is named after it. Specifically, the conditional independence assumption is

$$\begin{aligned} P(X = x|Y = c_k) &= P(X^{(1)} = x^{(1)}, \dots, X^{(n)} = x^{(n)}|Y = c_k) \\ &= \prod_{j=1}^n P(X^{(j)} = x^{(j)}|Y = c_k) \end{aligned} \quad (4.3)$$

Naïve Bayes actually learns the mechanism of generating data, so it belongs to the generative model. The conditional independence assumption is equivalent to assuming that the features used for classification are conditionally independent under the condition where the classes are determined. This assumption simplifies Naïve Bayes but sometimes sacrifices a certain degree of classification accuracy.

In Naïve Bayes classification, for a given input  $x$ , the posterior probability distribution  $P(Y = c_k|X = x)$  is computed through the learned model. The class with the maximum posterior probability is determined as the class output of  $x$ . The posterior probability is calculated according to the Bayes theorem as follows.

$$P(Y = c_k|X = x) = \frac{P(X = x|Y = c_k)P(Y = c_k)}{\sum_k P(X = x|Y = c_k)P(Y = c_k)} \quad (4.4)$$

Substituting Eqs. (4.3) into (4.4), we have

$$P(Y = c_k|X = x) = \frac{P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)}|Y = c_k)}{\sum_k P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)}|Y = c_k)}, \quad k = 1, 2, \dots, K \quad (4.5)$$

This is the basic formula of Naive Bayes classification. Thus, the Naive Bayes classifier can be expressed as

$$y = f(x) = \arg \max_{c_k} \frac{P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k)}{\sum_k P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k)} \quad (4.6)$$

Note that the denominator in Eq. (4.6) is the same for all  $c_k$ , thus,

$$y = \arg \max_{c_k} P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k) \quad (4.7)$$

#### 4.1.2 Implications of Posterior Probability Maximization

Naïve Bayes groups the instance into the class with the maximum posterior probability. It is equivalent to the expected risk minimization. Suppose we choose the 0–1 loss function:

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases}$$

where  $f(X)$  is the classification decision function. At this time, the expected risk function is

$$R_{\text{exp}}(f) = E[L(Y, f(X))]$$

The expectation is taken with respect to the joint distribution  $P(X, Y)$ . Thus, the conditional expectation is

$$R_{\text{exp}}(f) = E_X \sum_{k=1}^K [L(c_k, f(X))] P(c_k | X)$$

In order to minimize the expected risk, it is sufficient to minimize  $X = x$  one by one, from which we obtain:

$$\begin{aligned} f(x) &= \arg \min_{y \in \mathcal{Y}} \sum_{k=1}^K L(c_k, y) P(c_k | X = x) \\ &= \arg \min_{y \in \mathcal{Y}} \sum_{k=1}^K P(y \neq c_k | X = x) \end{aligned}$$

$$\begin{aligned}
&= \arg \min_{y \in \mathcal{Y}} (1 - P(y = c_k | X = x)) \\
&= \arg \min_{y \in \mathcal{Y}} P(y \neq c_k | X = x)
\end{aligned}$$

In this way, the posterior probability maximization criterion is obtained according to the expected risk minimization criterion:

$$f(x) = \arg \max_{c_k} P(c_k | X = x)$$

i.e., the principle adopted by the Naïve Bayes method.

## 4.2 Parameter Estimation of the Naïve Bayes Method

### 4.2.1 Maximum Likelihood Estimation

In Naïve Bayes, to learn means to estimate  $P(Y = c_k)$  and  $P(X^{(j)} = x^{(j)} | Y = c_k)$ . The Maximum Likelihood Estimation (MLE) can be applied to estimate the corresponding probability. The MLE of the prior probability  $P(Y = c_k)$  is

$$P(Y = c_k) = \frac{\sum_{i=1}^N I(y_i = c_k)}{N}, \quad k = 1, 2, \dots, K \quad (4.8)$$

Suppose the set of possible values of the  $j$ -th feature  $x^{(j)}$  is  $\{a_{j1}, a_{j2}, \dots, a_{js_j}\}$ , and the MLE of the conditional probability  $P(X^{(j)} = a_{jl} | Y = c_k)$  is

$$\begin{aligned}
P(X^{(j)} = a_{jl} | Y = c_k) &= \frac{\sum_{i=1}^N I(x_i^{(j)} = a_{jl}, y_i = c_k)}{\sum_{i=1}^N I(y_i = c_k)} \\
j &= 1, 2, \dots, n; l = 1, 2, \dots, S_j; k = 1, 2, \dots, K
\end{aligned} \quad (4.9)$$

where  $x_i^{(j)}$  is the  $j$ -th feature of the  $i$ -th sample;  $a_{jl}$  is the  $l$ -th value that the  $j$ -th feature may take;  $I$  is the indicator function.

### 4.2.2 Learning and Classification Algorithms

The learning and classification algorithms of Naive Bayes is given below.

**Algorithm 4.1 (Naïve Bayes Algorithm)** Input: Training data  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)})^T$ ,  $x_i^{(j)}$  is the  $j$ -th feature of the  $i$ -th sample,  $x_i^{(j)} \in \{a_{j1}, a_{j2}, \dots, a_{JS_j}\}$ , and  $a_{jl}$  is the  $l$ -th value that the  $i$ -th feature may take,  $j = 1, 2, \dots, n$ ,  $l = 1, 2, \dots, S_j$ ,  $y_i \in \{c_1, c_2, \dots, c_K\}$ ; instance  $x$ ;

Output: The classification of instance  $x$ .

- (1) Calculate prior probability and conditional probability

$$P(Y = c_k) = \frac{\sum_{i=1}^N I(y_i = c_k)}{N}, \quad k = 1, 2, \dots, K$$

$$P(X^{(j)} = a_{jl} | Y = c_k) = \frac{\sum_{i=1}^N I(x_i^{(j)} = a_{jl}, y_i = c_k)}{\sum_{i=1}^N I(y_i = c_k)}$$

$$j = 1, 2, \dots, n; l = 1, 2, \dots, S_j; k = 1, 2, \dots, K$$

- (2) For a given instance  $x = (x^{(1)}, x^{(2)}, \dots, x^{(n)})^T$ , calculate

$$P(Y = c_k) \prod_{j=1}^n P(X^{(j)} = x^{(j)} | Y = c_k), \quad k = 1, 2, \dots, K$$

- (3) Determine the class of instance  $x$ .

$$y = \arg \max_{c_k} P(Y = c_k) \prod_{j=1}^n P(X^{(j)} = x^{(j)} | Y = c_k)$$

**Example 4.1** Try to learn a Naive Bayes classifier from the training data in Table 4.1 and determine the class label  $y$  of  $x = (2, S)^T$ . In the table,  $X^{(1)}$  and  $X^{(2)}$  are features with the set of values  $A_1 = \{1, 2, 3\}$ ,  $A_2 = \{S, M, L\}$ , and  $Y$  is the class label,  $Y \in C = \{1, -1\}$ .

**Table 4.1** Training data

**Solution** According to Algorithm 4.1 and Table 4.1, it is easy to calculate the following probabilities:

$$\begin{aligned} P(Y = 1) &= \frac{9}{15}, P(Y = -1) = \frac{6}{15} \\ P(X^{(1)} = 1|Y = 1) &= \frac{2}{9}, P(X^{(1)} = 2|Y = 1) = \frac{3}{9}, P(X^{(1)} = 3|Y = 1) = \frac{4}{9} \\ P(X^{(2)} = S|Y = 1) &= \frac{2}{9}, P(X^{(2)} = M|Y = 1) = \frac{3}{9}, P(X^{(2)} = L|Y = 1) = \frac{4}{9} \\ P(X^{(1)} = 1|Y = -1) &= \frac{3}{6}, P(X^{(1)} = 2|Y = -1) = \frac{2}{6}, P(X^{(1)} = 3|Y = -1) = \frac{1}{6} \\ P(X^{(2)} = S|Y = -1) &= \frac{3}{6}, P(X^{(2)} = M|Y = -1) = \frac{2}{6}, P(X^{(2)} = L|Y = -1) = \frac{1}{6} \end{aligned}$$

For a given  $x = (2, S)^T$ , calculate

$$\begin{aligned} P(Y = 1)P(X^{(1)} = 2|Y = 1)P(X^{(2)} = S|Y = 1) &= \frac{9}{15} \cdot \frac{3}{9} \cdot \frac{1}{9} = \frac{1}{45} \\ P(Y = -1)P(X^{(1)} = 2|Y = -1)P(X^{(2)} = S|Y = -1) &= \frac{6}{15} \cdot \frac{2}{6} \cdot \frac{3}{6} = \frac{1}{15} \end{aligned}$$

Since  $P(Y = -1)P(X^{(1)} = 2|Y = -1)P(X^{(2)} = S|Y = -1)$  is the maximum, we have  $y = -1$ .

### 4.2.3 Bayesian Estimation

With the Maximum Likelihood Estimation, the probability value to be estimated may be 0. At this time, it will affect the computation results of the posterior probability and make the classification biased. The way to solve this problem is to adopt Bayesian Estimation. Specifically, the Bayesian Estimation of conditional probability is

$$P_\lambda(X^{(j)} = a_{jl}|Y = c_k) = \frac{\sum_{i=1}^N I(x_i^{(j)} = a_{jl}, y_i = c_k) + \lambda}{\sum_{i=1}^N I(y_i = c_k) + S_j + \lambda} \quad (4.10)$$

where  $\lambda \geq 0$ . It is equivalent to assigning a positive number  $\lambda > 0$  to the frequency of each value of the random variable. It is the Maximum Likelihood Estimation when  $\lambda = 0$ . Generally, the value of  $\lambda$  is taken as 1, and the estimation is called Laplacian Smoothing at this time. Obviously, for any  $l = 1, 2, \dots, S_j$ ,  $k = 1, 2, \dots, K$ , we have

$$P_\lambda(X^{(j)} = a_{jl}|Y = c_k) > 0$$

$$\sum_{l=1}^{S_j} P_\lambda(X^{(j)} = a_{jl}|Y = c_k) = 1$$

showing that Formula (4.10) is indeed a probability distribution. Similarly, the Bayesian Estimation of the prior probability is

$$P_\lambda(Y = c_k) = \frac{\sum_{i=1}^N I(y_i = c_k) + \lambda}{N + K\lambda} \quad (4.11)$$

**Example 4.2** The problem is the same as Example 4.1. Estimate the probability according to Laplace Smoothing, i.e.,  $\lambda=1$ .

**Solution**  $A_1 = \{1, 2, 3\}$ ,  $A_2 = \{S, M, L\}$ ,  $C = \{1, -1\}$ . Calculate the following probabilities according to Eqs. (4.10) and (4.11)

$$\begin{aligned} P(Y = 1) &= \frac{10}{17}, P(Y = -1) = \frac{7}{17} \\ P(X^{(1)} = 1|Y = 1) &= \frac{3}{12}, P(X^{(1)} = 2|Y = 1) = \frac{4}{12}, P(X^{(1)} = 3|Y = 1) = \frac{5}{12} \\ P(X^{(2)} = S|Y = 1) &= \frac{2}{12}, P(X^{(2)} = M|Y = 1) = \frac{5}{12}, P(X^{(2)} = L|Y = 1) = \frac{5}{12} \\ P(X^{(1)} = 1|Y = -1) &= \frac{4}{9}, P(X^{(1)} = 1|Y = -1) = \frac{3}{9}, P(X^{(1)} = 3|Y = -1) = \frac{2}{9} \\ P(X^{(2)} = S|Y = -1) &= \frac{4}{9}, P(X^{(2)} = M|Y = -1) = \frac{3}{9}, \\ P(X^{(2)} = L|Y = -1) &= \frac{2}{9} \end{aligned}$$

For a given  $x = (2, S)^T$ , calculate:

$$\begin{aligned} P(Y = 1)P(X^{(1)} = 2|Y = 1)P(X^{(2)} = S|Y = 1) \\ = \frac{10}{17} \cdot \frac{4}{12} \cdot \frac{2}{12} = \frac{5}{153} = 0.0327 \\ P(Y = -1)P(X^{(1)} = 2|Y = -1)P(X^{(2)} = S|Y = -1) \\ = \frac{7}{17} \cdot \frac{3}{9} \cdot \frac{4}{9} = \frac{28}{459} = 0.0610 \end{aligned}$$

Since  $P(Y = -1)P(X^{(1)} = 2|Y = -1)P(X^{(2)} = S|Y = -1)$  is the maximum,  $y = -1$ .

## Summary

1. Naive Bayes is a typical generative learning method. The generative method learns the joint probability distribution  $P(X, Y)$  from the training data and then obtains the posterior probability distribution  $P(Y|X)$ . Specifically, the training data is used to learn the estimation of  $P(X|Y)$  and  $P(Y)$  to obtain the joint probability distribution:

$$P(X, Y) = P(Y)P(X|Y)$$

The probability estimation method can be either Maximum Likelihood Estimation or Bayesian Estimation.

2. The basic assumption of Naive Bayes is conditional independence

$$\begin{aligned} P(X = x|Y = c_k) &= P(X^{(1)} = x^{(1)}, \dots, X^{(n)} = x^{(n)}|Y = c_k) \\ &= \prod_{j=1}^n P(X^{(j)} = x^{(j)}|Y = c_k) \end{aligned}$$

It is a strong assumption. Because of this assumption, the number of conditional probabilities contained in the model is greatly reduced, and the learning and prediction of Naive Bayes are greatly simplified. The Naive Bayes method is, therefore, efficient and easy to implement. The disadvantage is that the performance of classification is not necessarily high.

3. Naive Bayes Uses Bayesian Theorem and Learned Joint Probability Model for Classification Prediction

$$P(Y|X) = \frac{P(X, Y)}{P(X)} = \frac{P(Y)P(X|Y)}{\sum_Y P(Y)P(X|Y)}$$

The input  $x$  is divided into the class  $y$  with the largest posterior probability.

$$y = \arg \max_{c_k} P(Y = c_k) \prod_{j=1}^n P(X_j = x^{(j)}|Y = c_k)$$

The maximum posterior probability is equivalent to minimizing the expected risk of the 0–1 loss function.

### Further Reading

The introduction of Naïve Bayes can be seen in Refs. [1, 2]. Naïve Bayes assumes that the input variables are all conditionally independent. If it is assumed that there is a probability dependence between them, the model becomes a Bayesian network, see Ref. [3].

### Exercises

- 4.1 Derive the probability estimation Formulae (4.8) and (4.9) in the Naïve Bayes method by the Maximum Likelihood Estimation.
- 4.2 Deduce the probability estimation Formulae (4.10) and (4.11) in Naïve Bayes using Bayesian Estimation.

## References

1. Mitchell TM. Chapter 3: Generative and discriminative classifiers: Naïve Bayes and logistic regression. In: Machine Learning. Draft; 2005. <http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf>.
2. Hastie T, Tibshirani R, Friedman J. The elements of statistical learning: data mining, inference, and prediction. Springer-Verlag; 2001.
3. Bishop CM. Pattern recognition and machine learning. Springer; 2006.

# Chapter 5

## Decision Tree

The decision tree is a basic classification and regression method. This chapter focuses on the decision tree for classification. The decision tree model has a tree structure, representing the process of classifying instances based on features in the classification problem. It can be regarded as the set of if-then rules or the conditional probability distribution defined in feature and class space. Its main advantages are that the model is readable and the classification is fast. During learning, the training data are used to establish the decision tree model according to the principle of minimizing the loss function. During prediction, the new data is classified by the decision tree model. Decision tree learning usually consists of 3 steps: feature selection, decision tree generation and decision tree pruning. These ideas of decision tree learning are mainly derived from the ID3 algorithm in 1986 and the C4.5 algorithm in 1993, both proposed by Quinlan, and the CART algorithm proposed by Breiman et al. in 1984.

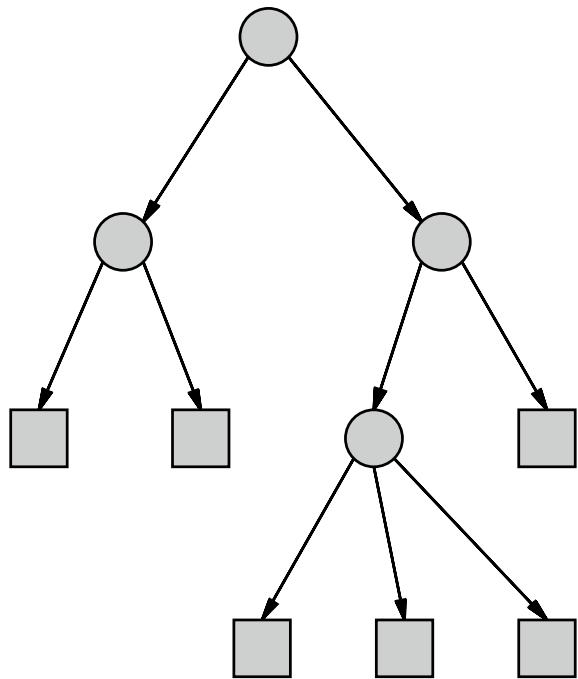
This chapter first introduces the basic concept of the decision tree, then introduces feature selection, tree-generation and tree-pruning through ID3 and C4.5 algorithms, and finally introduces the CART algorithm.

### 5.1 Decision Tree Model and Learning

#### 5.1.1 *Decision Tree Model*

**Definition 5.1 (Decision Tree)** The classification decision tree model is a tree structure that describes the classification of instances. A decision tree consists of nodes and directed edges. There are two types of node: the internal node and the leaf node. The internal node represents a feature or an attribute, and the leaf node represents a class.

**Fig. 5.1** The decision tree model



The classification with a decision tree starts from the root node, testing a feature of the instance. According to the test results, the instance is assigned to its child nodes. At this time, each child node corresponds to a feature value. The instance is then tested and assigned recursively until the leaf node is reached. Finally, the instance is divided into the class of leaf nodes.

Figure 5.1 is a diagram of a decision tree in which the circle and the box represent the internal node and the leaf node, respectively.

### 5.1.2 Decision Tree and If-Then Rules

The decision tree can be regarded as a set of if-then rules. The process of converting the decision tree into if-then rules is as follows. A rule is constructed for each path from the root node to the decision tree's leaf node; the internal nodes' features on the path correspond to the conditions of the rule, while the classes of the leaf nodes correspond to the conclusions of the rule. The paths of a decision tree or their corresponding set of if-then rules have an important property: they are mutually exclusive and complete. That is, every instance is covered by a path or a rule and only by one path or one rule. Coverage here means the features of the instance are consistent with the features on the path, or the instance satisfies the rule's conditions.

### 5.1.3 Decision Tree and Conditional Probability Distributions

The decision tree also represents the conditional probability distribution of the class under the given feature condition. This conditional probability distribution is defined on a partition of the feature space. A conditional probability distribution is formed by dividing the feature space into disjoint cells or regions and defining the probability distribution of a class in each cell. A path in the decision tree corresponds to a cell in the partition. The conditional probability distribution represented by the decision tree consists of the conditional probability distributions of the classes under the given conditions of each cell. Assuming that  $X$  is a random variable representing features and  $Y$  is a random variable representing classes, then the conditional probability distribution can be expressed as  $P(Y|X)$ .  $X$  takes the value from the set of cells under the given partition, and  $Y$  takes the value from the set of classes. The conditional probability of each leaf node (cell) tends to be biased towards a specific class, i.e., the probability of belonging to a certain class is higher. In decision tree classification, the node instances are forcibly grouped into the class with the higher conditional probability.

Figure 5.2a shows a partition of the feature space schematically. The large square in the figure represents the feature space and is divided by several small rectangles, each of which represents a cell. The cells on the feature space partition form a set, where  $X$  takes its value. For simplicity, it is assumed that there are only two classes: positive and negative, i.e., the value of  $Y$  takes  $+1$  and  $-1$ . The numbers in small rectangles indicate the class of each cell. Figure 5.2b represents the conditional probability distribution of classes with given features (cells) when the partition of the feature space is determined. The conditional probability distribution in Fig. 5.2b corresponds to the partition in Fig. 5.2a. When the conditional probability of a cell  $c$  satisfies  $P(Y = +1|X = c) > 0.5$ , the cell is considered as a positive class, i.e., all the instances falling in this cell are considered positive classes. Figure 5.2c shows the decision tree corresponding to the conditional probability distribution in Fig. 5.2b.

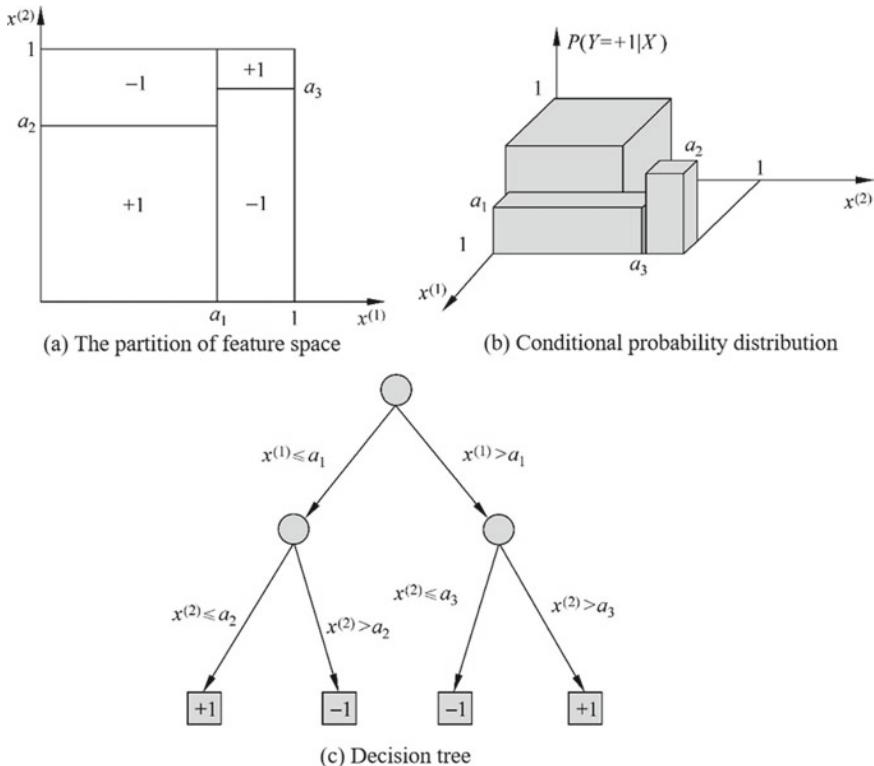
### 5.1.4 Decision Tree Learning

Assume the training dataset is given as

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)})^T$  is the input instances (feature vector),  $n$  is the number of features,  $y_i \in \{1, 2, \dots, K\}$  is the class label,  $i = 1, 2, \dots, N$ ,  $N$  is the sample capacity. The goal of decision tree learning is to construct a decision tree model according to the given training dataset to classify the instances correctly.

Decision tree learning essentially induces a set of classification rules from a training dataset. There could be more than one decision tree that does not contradict



**Fig. 5.2** The decision tree corresponding to the conditional probability distribution

the training dataset (i.e., the decision tree that correctly classifies the training data) or none. What we need is a decision tree that is less contradictory to the training data and has good generalization capabilities. From another perspective, decision tree learning is the estimation of conditional probabilistic models from the training dataset. There are infinite conditional probabilistic models for classes based on feature space partition. The conditional probability model we choose should be a good fit for the training data and a good predictor of the unknown data.

Decision tree learning represents this objective with a loss function. As described below, the loss function for decision tree learning is usually a regularized maximum likelihood function. The strategy of decision tree learning is to minimize the loss function as the objective function.

When the loss function is determined, the learning problem becomes the problem of choosing the optimal decision tree in the sense of the loss function. Since selecting the optimal decision tree from all possible decision trees is an NP-Complete problem, in reality, decision tree learning algorithms usually use heuristics to approximately solve this optimization problem. The resulting decision tree is sub-optimal.

The algorithm for decision tree learning is typically a process of recursively selecting the optimal feature and partitioning the training data according to that feature so that we can get the best classification for each sub-dataset. This process corresponds to the partition of the feature space and the construction of the decision tree. First, build the root node and put all the training data in it. The optimal feature is selected, and the training dataset is divided into subsets according to this feature so that each subset has the best classification under the current condition. If these subsets can be classified correctly, then construct leaf nodes and classify these subsets into the corresponding leaf nodes; if there are still subsets that cannot be classified correctly, then select new optimal features for these subsets, continue to divide them, and construct the corresponding nodes. Conduct it recursively until all training data subsets are classified correctly or there are no suitable features. Finally, each sub-set is assigned to a leaf node, i.e., all of them have a specific class. Then a decision tree is generated.

The decision tree generated by the above method may have good classification ability for training data, but not for unknown test data, i.e., over-fitting may occur. We need to prune the generated tree from the bottom up to make the tree simpler and thus have better generalization ability. Specifically, over-subdivided leaf nodes are removed, reverting to the parent node or even higher nodes, and then the parent nodes or higher nodes are replaced with new leaf nodes.

If the number of features is large, features can also be selected at the beginning of the decision tree learning process, leaving only those features that have sufficient classification ability for the training data.

It can be seen that the decision tree learning algorithm consists of feature selection, decision tree generation, and decision tree pruning processes. Since the decision tree represents a conditional probability distribution, decision trees of different depths correspond to probabilistic models of different complexity. Decision tree generation corresponds to the local selection of the model, and decision tree pruning corresponds to the global selection of the model. Thus, decision tree generation only considers the local optimum, while decision tree pruning considers the global optimum.

The commonly used algorithms for decision tree learning are ID3, C4.5, and CART. The feature selection, decision tree generation, and pruning processes of decision tree learning are described below in conjunction with these algorithms.

## 5.2 Feature Selection

### 5.2.1 *The Feature Selection Problem*

Feature selection is selecting features that can classify training data to improve the efficiency of decision tree learning. A feature is said to have no classification power if the results of classification using it do not differ significantly from the results of random classification. Empirically, excluding such features has little effect on the

**Table 5.1** A sample data sheet of loan applications

ID	Age	Has_job	Own_house	Credit_rating	Class
1	Young	False	False	Fair	No
2	Young	False	False	Good	No
3	Young	True	False	Good	Yes
4	Young	True	True	Fair	Yes
5	Young	False	False	Fair	No
6	Middle	False	False	Fair	No
7	Middle	False	False	Good	No
8	Middle	True	True	Good	Yes
9	Middle	False	True	Excellent	Yes
10	Middle	False	True	Excellent	Yes
11	Old	False	True	Excellent	Yes
12	Old	False	True	Good	Yes
13	Old	True	False	Good	Yes
14	Old	True	False	Excellent	Yes
15	Old	False	False	Fair	No

accuracy of decision tree learning. Usually, the criterion of feature selection is the information gain or information gain ratio.

First, an example is used to illustrate the feature selection problem.

**Example 5.1**<sup>1</sup> Table 5.1 shows a loan application training dataset consisting of 15 samples. It includes four features (attributes) of loan applicants. The first feature is Age, with three possible values: young, middle and old. The second one is Has\_Job (job status), which has two possible values, true and false. The third one is Own\_house (housing ownership), with two possible values: true and false. The fourth feature is the Credit\_rating, which has three possible values, excellent, good, and fair. The last column of the table is the Class, indicating whether the loan application was approved, with two values: Yes and No.

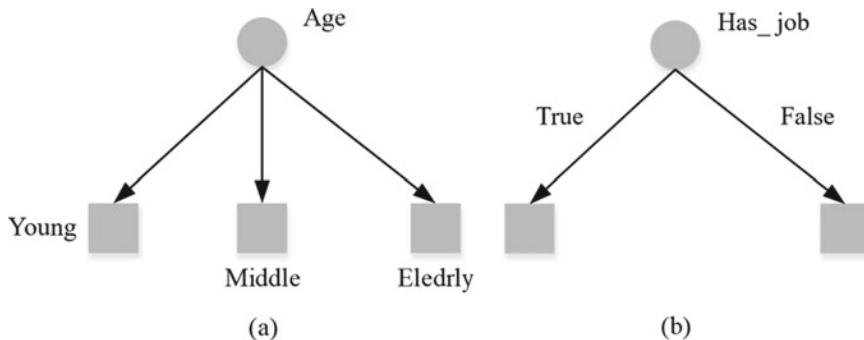
It is expected to learn a decision tree for loan applications through the training data to classify future loan applications. That is, when a new customer applies for a loan, the decision tree can be used to decide whether to approve the loan application based on the applicant's features.

Feature selection decides which feature to use to partition the feature space.

Figure 5.3 shows two possible decision trees learned from the data in Table 5.1, each consisting of two root nodes with different features. The root node feature shown in Fig. 5.3a is Age, with three values and different sub-nodes corresponding to different values. The root node feature shown in Fig. 5.3b is Has\_job (job status), with two values and different sub-nodes corresponding to different values. Both decision

---

<sup>1</sup> This example is taken from reference [5].



**Fig. 5.3** Different decision trees determined by different features

trees can continue from there. The problem is: Which feature is better to choose? It requires determining the criteria for selecting features. Intuitively, if a feature has better classification ability, or if the training dataset is divided into subsets according to this feature, which leads to the best classification for each subset under the current conditions, then this feature should be selected. Information gain can well represent this intuitive criterion.

### 5.2.2 *Information Gain*

For the convenience of explanation, the definitions of entropy and conditional entropy are given first.

In information theory and probability statistics, entropy measures the uncertainty of random variables. Let  $X$  be a discrete random variable with a finite number of values, and its probability distribution is

$$P(X = x_i) = p_i, \quad i = 1, 2, \dots, n$$

Then the entropy of the random variable  $X$  is defined as:

$$H(X) = - \sum_{i=1}^n p_i \log p_i \tag{5.1}$$

In Formula (5.1), if  $p_i = 0$ , define  $0 \log 0 = 0$ . Normally, the logarithm in Formula (5.1) has the base of 2 or the base e. The natural logarithm is the logarithm with the base e. At this time, the unit of entropy is called bit or nat, respectively. It can be seen from the definition that entropy only depends on the distribution of  $X$  and has nothing to do with the value of  $X$ , so the entropy of  $X$  can also be denoted as  $H(p)$ , i.e.,

$$H(p) = -\sum_{i=1}^n p_i \log p_i \quad (5.2)$$

The greater the entropy, the greater the uncertainty of the random variable. It is verifiable from the definition that

$$0 \leq H(p) \leq \log n \quad (5.3)$$

When the random variable only takes two values, such as 1, 0, that is, the distribution of  $X$  is

$$P(X = 1) = p, P(X = 0) = 1 - p, 0 \leq p \leq 1$$

The entropy is

$$H(p) = -p \log_2 p - (1 - p) \log_2(1 - p) \quad (5.4)$$

At this time, the curve indicating that entropy  $H(p)$  varies with probability  $p$  is shown in Fig. 5.4 (in bit).

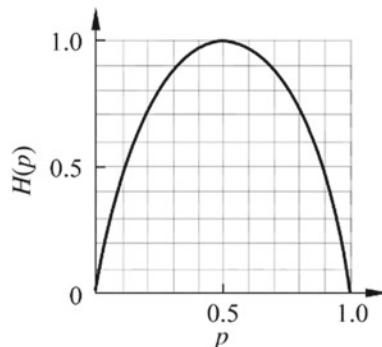
When  $p = 0$  or  $p = 1$ ,  $H(p) = 0$ , the random variable has no uncertainty at all. When  $p = 0.5$ ,  $H(p) = 1$ , the value of entropy is the largest, and the uncertainty of random variables is the largest.

Given random variables  $(X, Y)$ , and the joint probability distribution is

$$P(X = xi, Y = yj) = pij, \quad i = 1, 2, \dots, n; \quad j = 1, 2, \dots, m$$

Conditional entropy  $H(Y|X)$  represents the uncertainty of the random variable  $Y$  conditional on the known random variable  $X$ . The conditional entropy  $H(Y|X)$  of the random variable  $Y$  with the given random variable  $X$  is defined as the mathematical expectation of the entropy of the conditional probability distribution of  $Y$  with the given  $X$ .

**Fig. 5.4** The relationship between entropy and probability in Bernoulli distribution



$$H(Y|X) = \sum_{i=1}^n p_i H(Y|X = x_i) \quad (5.5)$$

Here,  $p_i = P(X = x_i)$ ,  $i = 1, 2, \dots, n$ .

When the probability in entropy and conditional entropy is obtained by data estimation (especially Maximum Likelihood Estimation), the corresponding entropy and conditional entropy are called empirical entropy and empirical conditional entropy, respectively. At this time, if there is a probability of 0, let  $0 \log 0 = 0$ .

Information gain indicates the degree of uncertainty reduction of the information of class  $Y$  by knowing the feature  $X$ .

**Definition 5.2 (Information Gain)** The information gain  $g(D, A)$  of feature  $A$  to the training dataset  $D$  is defined as the difference between the empirical entropy  $H(D)$  of set  $D$  and the empirical conditional entropy  $H(D|A)$  with the given feature  $A$ , i.e.,

$$g(D, A) = H(D) - H(D|A) \quad (5.6)$$

Generally, the difference between the entropy  $H(Y)$  and the conditional entropy  $H(Y|X)$  is called mutual information. The information gain in decision tree learning is equivalent to the mutual information between classes and features in the training dataset.

Decision tree learning employs information gain criterion to select features. Given the training dataset  $D$  and feature  $A$ , the empirical entropy  $H(D)$  represents the uncertainty of classifying the dataset  $D$ . The empirical conditional entropy  $H(D|A)$  represents the uncertainty of classifying dataset  $D$  with the given feature  $A$ . Then their difference, i.e., the information gain, represents the degree of uncertainty reduction in the classification of dataset  $D$  due to feature  $A$ . Obviously, the information gain for dataset  $D$  depends on the features, and different features often have different information gains. Features with greater information gain tend to have stronger classification power.

The feature selection method based on the information gain criterion is: for the training dataset (or subset)  $D$ , compute the information gain of each feature, compare their sizes, and select the feature with the largest information gain.

Let the training dataset be  $D$ , and  $|D|$  represents its sample size, i.e., the number of samples. Suppose there are  $K$  classes, denoted as  $C_k$ ,  $k = 1, 2, \dots, K$ , and  $|C_k|$  is the number of samples belonging to class  $C_k$ , then we have  $\sum_{k=1}^K |C_k| = |D|$ . Let feature  $A$  have  $n$  different values  $\{a_1, a_2, \dots, a_n\}$ , and divide  $D$  into  $n$  subsets  $D_1, D_2, \dots, D_n$  according to the value of feature  $A$ . With  $|D_i|$  as the number of samples of  $D_i$ , we have  $\sum_{i=1}^n |D_i| = |D|$ . The set of samples belonging to class  $C_k$  in the subset  $D_i$  is denoted as  $D_{ik}$ , i.e.,  $D_{ik} = D_i \cap C_k$ , and  $|D_{ik}|$  is the number of samples of  $D_{ik}$ . Then the information gain algorithm is as follows.

### Algorithm 5.1 (Information Gain Algorithm)

Input: training dataset  $D$  and feature  $A$ ;

Output: the information gain information gain  $g(D, A)$  of feature  $A$  to training dataset  $D$ .

- (1) Calculate the empirical entropy  $H(D)$  of the dataset  $D$

$$H(D) = - \sum_{k=1}^K \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|} \quad (5.7)$$

- (2) Calculate the empirical conditional entropy  $H(D|A)$  of feature  $A$  to dataset  $D$

$$H(D|A) = \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) = - \sum_{i=1}^n \frac{|D_i|}{|D|} \sum_{k=1}^K \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|} \quad (5.8)$$

- (3) Calculate the information gain

$$g(D, A) = H(D) - H(D|A) \quad (5.9)$$

**Example 5.2** For the training dataset  $D$  given in Table 5.1, select the optimal feature according to the information gain criterion.

**Solution** First, calculate the empirical entropy  $H(D)$ .

$$H(D) = - \frac{9}{15} \log_2 \frac{9}{15} - \frac{6}{15} \log_2 \frac{6}{15} = 0.971$$

Then calculate the information gain of each feature to the dataset  $D$ .  $A_1, A_2, A_3$ , and  $A_4$  represent the four features: Age, Has\_job (job status), Own\_house (housing ownership), and Credit rating, then.

$$\begin{aligned} g(D, A_1) &= H(D) - \left[ \frac{5}{15} H(D_1) + \frac{5}{15} H(D_2) + \frac{5}{15} H(D_3) \right] \\ (1) \quad &= 0.971 - \left[ \frac{5}{15} \left( -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} \right) \right. \\ &\quad \left. + \frac{5}{15} \left( -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} \right) + \frac{5}{15} \left( -\frac{4}{5} \log_2 \frac{4}{5} - \frac{1}{5} \log_2 \frac{1}{5} \right) \right] \\ &= 0.971 - 0.888 = 0.083 \end{aligned}$$

Here  $D_1, D_2$ , and  $D_3$  are the sample subsets in which  $A_1$  (Age) in  $D$  is young, middle, and old, respectively. Similarly,

$$\begin{aligned}
 g(D, A_2) &= H(D) - \left[ \frac{5}{15} H(D_1) + \frac{10}{15} H(D_2) \right] \\
 (2) \quad &= 0.971 - \left[ \frac{5}{15} \times 0 + \frac{10}{15} \left( -\frac{4}{10} \log_2 \frac{4}{10} - \frac{6}{10} \log_2 \frac{6}{10} \right) \right] \\
 &= 0.324 \\
 (3) \quad g(D, A_3) &= 0.971 - \left[ \frac{6}{15} \times 0 + \frac{9}{15} \left( -\frac{3}{9} \log_2 \frac{3}{9} - \frac{6}{9} \log_2 \frac{6}{9} \right) \right] \\
 &= 0.971 - 0.551 = 0.420 \\
 (4) \quad g(D, A_4) &= 0.971 - 0.608 = 0.363
 \end{aligned}$$

Finally, compare the information gain values of features. Since feature  $A_3$  (Own\_house) has the largest information gain value, it is selected as the optimal feature.

### 5.2.3 Information Gain Ratio

Regarding information gain as the feature of dividing the training dataset, there is a problem of bias in selecting features with more possible values. This problem can be corrected by the information gain ratio, another feature selection criterion.

**Definition 5.3 (Information Gain Ratio)** The information gain ratio of feature  $A$  to training dataset  $D$ ,  $g_R(D, A)$ , is defined as the ratio of its information gain  $g(D, A)$  to training dataset  $D$ 's entropy  $H_A(D)$  concerning the value of feature  $A$ , i.e.,

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)} \quad (5.10)$$

where  $H_A(D) = -\sum_{i=1}^n \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$ , and  $n$  is the number of values of feature  $A$ .

## 5.3 Generation of Decision Tree

This section describes generative algorithms for decision tree learning. The generative algorithm of ID3 is introduced first, followed by the generative algorithm in C4.5. They are both classic algorithms of decision tree learning.

### 5.3.1 ID3 Algorithm

The core of the ID3 algorithm is to apply the information gain criterion to selecting features on each node of the decision tree and build the decision tree recursively. The

specific method is as follows. Start from the root node and calculate the information gains of all possible features for the node. Select the feature with the maximum information gain as the feature of the node, and establish sub-nodes according to different values of the feature. Then apply the above method recursively on child nodes to build a decision tree until all features have little information gain or can't be selected. Finally, a decision tree is obtained. ID3 is equivalent to choosing a probability model by the maximum likelihood method.

### Algorithm 5.2 (ID3 Algorithm)

Input: Training dataset  $D$ , the threshold  $\varepsilon$  of feature set  $A$ ;

Output: Decision tree  $T$ .

- (1) If all instances in  $D$  belong to the same class  $C_k$ , then  $T$  is a single-node tree.  
Return  $T$  with class  $C_k$  as the class tag of the node;
- (2) If  $A = \emptyset$ , then  $T$  is a single-node tree, and the class  $C_k$  with the largest number of instances in  $D$  is taken as the class tag of the node. Return  $T$ ;
- (3) Otherwise, calculate the information gain of each feature in  $A$  to  $D$  by algorithm 5.1, and select  $A_g$ , the feature with the maximum information gain;
- (4) If the information gain of  $A_g$  is less than the threshold  $\varepsilon$ , set  $T$  as a single-node tree, use the class  $C_k$  with the largest number of instances in  $D$  as the class tag of the node, and return  $T$ ;
- (5) Otherwise, for each possible value  $a_i$  of  $A_g$ , partition  $D$  into a number of non-empty subsets  $D_i$  according to  $A_g = a_i$ , use the class with the largest number of instances in  $D_i$  as the tag to construct child nodes. The tree  $T$  is formed by the node and its child nodes and then returned;
- (6) For the  $i$ -th child node, use  $D_i$  as the training set and  $A - \{A_g\}$  as the feature set, call step (1) ~ step (5) recursively, get the subtree  $T_i$ , and return  $T_i$ .

**Example 5.3** For the training dataset in Table 5.1, use the ID3 algorithm to build a decision tree.

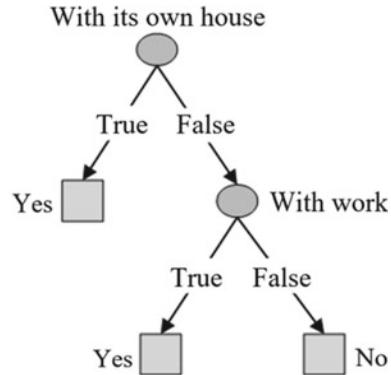
**Solution** Using the result of Example 5.2, feature  $A_3$ (Own\_house) is selected as the feature of the root node because it has the largest information gain value. It divides the training dataset  $D$  into two subsets,  $D_1$  (the value of  $A_3$  is “true”) and  $D_2$  (the value of  $A_3$  is “false”). Since  $D_1$  only has sample points of the same class, it becomes a leaf node, and the class of the node is marked as “Yes”.

For  $D_2$ , we need to select new characteristics from features  $A_1$  (Age),  $A_2$  (Has\_job), and  $A_4$  (Credit\_rating). Calculate the information gain of each feature:

$$g(D_2, A_1) = H(D_2) - H(D_2|A_1) = 0.918 - 0.667 = 0.251$$

$$g(D_2, A_2) = H(D_2) - H(D_2|A_2) = 0.918$$

**Fig. 5.5** Decision tree generation



$$g(D_2, A_4) = H(D_2) - H(D_2|A_4) = 0.474$$

The feature  $A_2$  (Has\_job) with the largest information gain is selected as the feature of the node. Since  $A_2$  has two possible values, two child nodes are derived from this node. One is a child node corresponding to “true” (has a job) and contains three samples, which belong to the same class. Thus, it is a leaf node, and the class is marked as “Yes”. The other is a child node corresponding to “false” (has no job) and contains six samples, which also belong to the same class. It is also a leaf node, and the class is marked as “No”.

In this way, a decision tree shown in Fig. 5.5 is generated. It uses only two features (two internal nodes).

The ID3 algorithm only deals with tree generation, so the trees generated by this algorithm are prone to over-fitting.

### 5.3.2 C4.5 Generation Algorithm

The C4.5 algorithm is similar to the ID3 algorithm, with the C4.5 algorithm improving the ID3 algorithm. In the generation process of the C4.5 algorithm, the information gain ratio is used to select features.

#### Algorithm 5.3 (C4.5 Generation Algorithm)

Input: Training dataset  $D$ , feature set  $A$  and the threshold  $\varepsilon$ ;

Output: Decision tree  $T$ .

- (1) If all the instances in  $D$  belong to the same class  $C_k$ , set  $T$  as a single-node tree, take  $C_k$  as the class of the node, and return  $T$ ;
- (2) If  $A = \emptyset$ , set  $T$  as a single-node tree, use the class  $C_k$  with the largest number of instances in  $D$  as the class of the node, and return  $T$ ;

- (3) Otherwise, calculate the information gain ratio of each feature in  $A$  to  $D$  according to Formula (5.10), and select  $A_g$ , the feature with the largest information gain ratio;
- (4) If the information gain ratio of  $A_g$  is less than the threshold  $\varepsilon$ , set  $T$  as a single-node tree, take the class  $C_k$  with the largest number of instances in  $D$  as the class of the node, and return  $T$ ;
- (5) Otherwise, for each possible value  $a_i$  of  $A_g$ , partition  $D$  into a number of non-empty subsets  $D_i$  according to  $A_g = a_i$ , take the class with the largest number of instances in  $D_i$  as a tag to construct child nodes, build the tree  $T$  by nodes and their child nodes, and return  $T$ ;
- (6) For the  $i$ -th child node, use  $D_i$  as the training set and  $A - \{A_g\}$  as the feature set, call step (1) – step (5) recursively to obtain the subtree  $T_i$ , and return  $T_i$ .

## 5.4 Pruning of Decision Tree

The decision tree generation algorithm recursively generates the decision tree until it cannot continue. The tree generated in this way is often very accurate in classifying training data but not so accurate for the classification of unknown test data, i.e., over-fitting occurs. The reason for over-fitting is that too much consideration is given to improving the correct classification of training data during learning, thereby constructing an overly complex decision tree. The solution to this problem is to consider the complexity of the decision tree and simplify the generated decision tree.

The process of simplifying the generated tree in decision tree learning is called pruning. Specifically, pruning simplifies the classification tree model by cutting off some subtrees or leaf nodes from the generated tree and using its root or parent node as new leaf nodes.

This section introduces a simple pruning algorithm for decision tree learning.

Decision tree pruning is often achieved by minimizing the overall loss function or cost function of the tree. Let the number of leaf nodes of tree  $T$  be  $|T|$ ,  $t$  is the leaf node of tree  $T$ , the leaf node has  $N_t$  sample points, among which there are  $N_{tk}$  sample points of class  $k$ ,  $k = 1, 2, \dots, K$ .  $H_t(T)$  is the empirical entropy of the leaf node  $t$ , and  $\alpha \geq 0$  is the parameter, then the loss function of decision tree learning can be defined as

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha |T| \quad (5.11)$$

where the empirical entropy is

$$H_t(T) = - \sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t} \quad (5.12)$$

In the loss function, the first item at the right-hand end of Formula (5.11) is denoted as

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = - \sum_{t=1}^{|T|} \sum_{k=1}^K N_{tk} \log \frac{N_{tk}}{N_t} \quad (5.13)$$

At this moment

$$C_\alpha(T) = C(T) + \alpha |T| \quad (5.14)$$

In Formula (5.14),  $C(T)$  represents the prediction error of the model on the training data, i.e., the degree of fitting between the model and the training data.  $|T|$  represents the complexity of the model, and the parameter  $\alpha \geq 0$  controls the influence between the two. A larger  $\alpha$  prompts the selection of a simpler model (tree), and a smaller  $\alpha$  motivates the selection of a more complex model (tree).  $\alpha=0$  means that only the fitting degree between the model and the training data is considered, ignoring the complexity of the model.

Pruning is to select the model with the minimum loss function when  $\alpha$  is determined, i.e., the subtree with the minimum loss function. When the value of  $\alpha$  is determined, the larger the subtree is, the better it tends to fit the training data, but the higher the complexity of the model is. On the contrary, the smaller the subtree is, the lower the complexity of the model is, but the worse it tends to fit the training data. The loss function just indicates exactly the balance between the two.

It can be seen that decision tree generation only considers a better fitting to the training data by improving the information gain (or information gain ratio). Decision tree pruning also considers reducing the model complexity by optimizing the loss function. Decision tree generates and learns the partial model, while decision tree pruning learns the overall model.

The minimization of the loss function defined by Formula (5.11) or (5.14) is equivalent to the regularized maximum likelihood estimation. Therefore, using the minimum loss function principle for pruning is to implement regularized maximum likelihood estimation for model selection.

Figure 5.6 is a schematic diagram of the decision tree pruning process. The following describes the pruning algorithm.

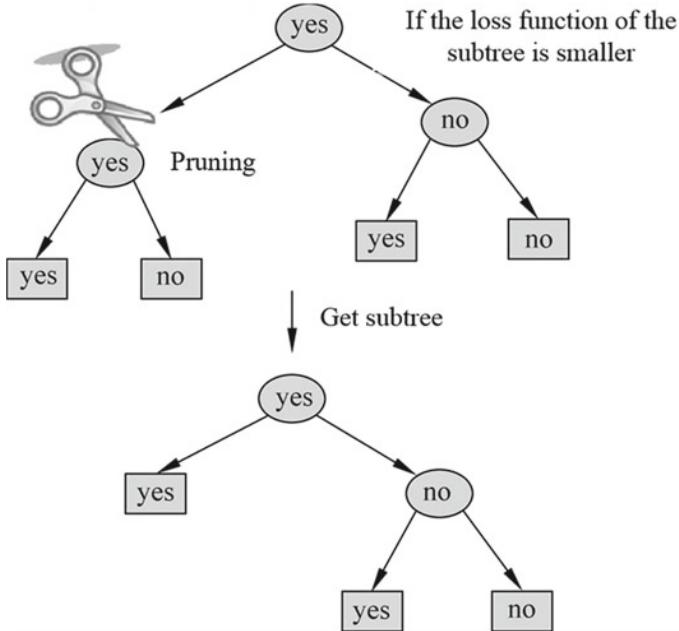
#### Algorithm 5.4 (Tree Pruning Algorithm)

Input: the entire tree  $T$  generated by the generation algorithm, the parameter  $\alpha$ ;

Output: pruned subtree  $T_\alpha$ .

- (1) Calculate the empirical entropy of each node.
- (2) Recursively retract upwards from the leaf nodes of the tree.

Let the overall tree before and after a set of leaf nodes retract to their parent nodes be  $T_B$  and  $T_A$ , respectively. Their corresponding loss function values are  $C_\alpha(T_B)$  and  $C_\alpha(T_A)$ . If



**Fig. 5.6** Pruning of the decision tree

$$C_\alpha(T_A) \leq C_\alpha(T_B) \quad (5.15)$$

then the pruning is performed, i.e., the parent node is changed into a new leaf node.

- (3) Return to (2) until it cannot continue, and get the subtree  $T_\alpha$  with the smallest loss function.

Note that Eq. (5.15) only needs to consider the difference between the loss functions of the two trees, and the calculation can be performed locally. Therefore, the pruning algorithm of the decision tree can be implemented by a dynamic programming algorithm. A similar dynamic programming algorithm can be found in literature [10].

## 5.5 CART Algorithm

The classification and regression tree (CART) model, proposed by Breiman et al. in 1984, is a widely used decision tree learning method. CART also consists of feature selection, tree generation, and pruning, and can be used for both classification and regression. The trees used for classification and regression are collectively referred to as decision trees in the following.

CART is a learning method that outputs the conditional probability distribution of random variable  $Y$  under the condition of a given input random variable  $X$ . CART assumes the decision tree is a binary tree, and the values of the internal node features are “Yes” and “No”. The left branch is the branch with the value “Yes”, and the right one is the branch with the value “No”. This decision tree is equivalent to recursively dichotomizing each feature. It divides the input space (i.e., the feature space) into a finite number of units, on which the predicted probability distribution, i.e. the conditional probability distribution of the output under the given input conditions, is determined.

The CART algorithm consists of two steps as follows:

- (1) Decision tree generation: the decision tree is generated based on the training dataset, and the generated decision tree should be as large as possible;
- (2) Decision tree pruning: The validation dataset is used to prune the generated tree and select the optimal subtree, using the minimum loss function as the pruning criterion.

### 5.5.1 ***CART Generation***

Decision tree generation is the process of recursively constructing a binary decision tree. A binary tree is generated by using the squared error minimization criterion for the regression tree and the Gini index minimization criterion is used for the classification tree to select features and generate the binary tree.

#### 5.5.1.1 ***The Generation of Regression Tree***

Suppose that  $X$  and  $Y$  are input and output variables respectively, and  $Y$  is a continuous variable, given the training dataset.

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

Consider how the regression tree is generated.

A regression tree corresponds to a partition of the input space (i.e., the feature space) and the output value on the partition unit. Suppose that the input space has been divided into  $M$  units  $R_1, R_2, \dots, R_M$ , and there is a fixed output value  $c_m$  on each unit  $R_m$ , then the regression tree model can be expressed as

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m) \quad (5.16)$$

When the partition of input space is determined, the prediction error of the regression tree for the training data can be represented by the square error

$\sum_{x_i \in R_m} (y_i - f(x_i))^2$ , and the optimal output value on each unit can be solved by the minimum square error criterion. It is easy to know that the optimal value  $\hat{c}_m$  of  $c_m$  on cell  $R_m$  is the mean value of output  $y_i$  corresponding to all input instances  $x_i$  on  $R_m$ , i.e.,

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m) \quad (5.17)$$

The problem is how to partition the input space. Here a heuristic method is used to select the  $j$ -th variable  $x^{(j)}$  and its value  $s$  as the splitting variable and the splitting point, and two regions are defined as:

$$R_1(j, s) = \{x | x^{(j)} \leq s\} \text{ and } R_2(j, s) = \{x | x^{(j)} > s\} \quad (5.18)$$

Then find the optimal splitting variable  $j$  and the optimal splitting point  $s$ . Specifically, solve the formula

$$\min_{j,s} \left[ \min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right] \quad (5.19)$$

For the fixed input variable  $j$ , the optimal splitting point  $s$  can be found.

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s)) \quad (5.20)$$

Iterate through all the input variables to find the optimal splitting point  $j$ , and form a pair  $(j, s)$ . The input space is then divided into two regions. Then repeat the above partition process for each area until the stopping condition is satisfied. This generates a regression tree, which is usually called the least squares regression tree. The algorithm is described as follows.

### Algorithm 5.5 (The Least Squares Regression Tree Generation Algorithm)

Input: The training dataset  $D$ ;

Output: The regression tree  $f(x)$ .

In the input space where the training dataset is located, recursively divide each region into two sub-regions and determine the output value on each sub-region to construct a binary decision tree:

- (1) Select the optimal splitting variable  $j$  and the splitting point  $s$ , and solve

$$\min_{j,s} \left[ \min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right] \quad (5.21)$$

Iterate through all the values of variable  $j$ , scan the splitting point  $s$  for the fixed splitting variable  $j$ , and select the pair  $(j, s)$  that minimizes the value of Eq. (5.21).

- (2) Divide the area with the selected pair  $(j, s)$  and determine the corresponding output value:

$$R_1(j, s) = \{x | x^{(j)} \leq s\}, R_2(j, s) = \{x | x^{(j)} > s\}$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m(j, s)} y_i, x \in R_m, m = 1, 2$$

- (3) Continue to call steps (1) and (2) for the two sub-regions until the stopping condition is met.  
 (4) Divide the input space into  $M$  areas  $R_1, R_2, \dots, R_M$ , and generate a decision tree:

$$f(x) = \sum_{m=1}^M \hat{c}_m I(x \in R_m)$$

### 5.5.1.2 The Generation of Classification Tree

The classification tree uses the Gini index to select the optimal feature and to determine the optimal binary splitting point of the feature.

**Definition 5.4 (Gini Index)** In the classification problem, suppose there are  $K$  classes and the probability that the sample point belongs to the  $k$ -th class is  $p_k$ , then the Gini index of the probability distribution is defined as

$$\text{Gini}(p) = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2 \quad (5.22)$$

For the binary classification problem, if the probability that the sample point belongs to the first class is  $p$ , then the Gini index of the probability distribution is:

$$\text{Gini}(p) = 2p(1 - p) \quad (5.23)$$

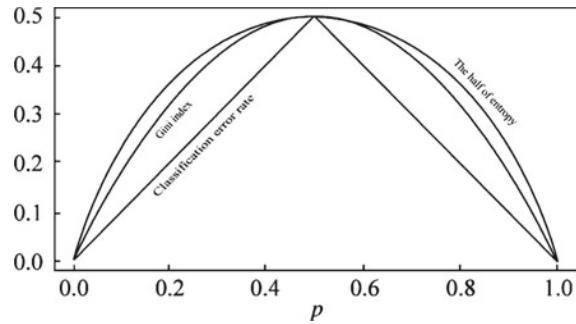
For a given sample set  $D$ , the Gini index is:

$$\text{Gini}(D) = 1 - \sum_{k=1}^K \left(\frac{|C_k|}{|D|}\right)^2 \quad (5.24)$$

Here,  $C_k$  is the subset of  $D$  whose samples belong to the  $k$ -th class, and  $K$  is the number of classes.

If the sample set  $D$  is divided into two parts  $D_1$  and  $D_2$  according to whether feature  $A$  takes a certain possible value  $a$ , i.e.,

**Fig. 5.7** The relationship between the Gini index, the half of entropy and the classification error rate



$$D_1 = \{(x, y) \in D | A(x) = a\}, D_2 = D - D_1$$

Then under the condition of feature  $A$ , the Gini index of set  $D$  is defined as:

$$\text{Gini}(D, A) = \frac{|D_1|}{|D|} \text{Gini}(D_1) + \frac{|D_2|}{|D|} \text{Gini}(D_2) \quad (5.25)$$

The Gini index  $\text{Gini}(D)$  represents the uncertainty of set  $D$ . The Gini index  $\text{Gini}(D, A)$  represents the uncertainty of set  $D$  after being partitioned by  $A = a$ . The larger the Gini index value, the greater the uncertainty of the sample set, which is similar to entropy.

Figure 5.7 shows the relationship between the Gini index  $\text{Gini}(p)$ , the half of entropy (in bits)  $H(p)/2$  and the classification error rate in the two-class classification problem. The horizontal ordinate represents the probability  $p$ , and the vertical ordinate represents the loss. It can be seen that the curves for the Gini index and the half of entropy are very close, and both approximate the classification error probability.

#### Algorithm 5.6 (CART Generation Algorithm)

Input: Training dataset  $D$  and conditions for stopping the calculation;

Output: CART decision tree.

Based on the training dataset, a binary decision tree is constructed by recursively performing the following operations on each node, starting from the root node:

- (1) Let the training dataset of the node be  $D$ . Calculate the Gini index of the existing features for this dataset. At this point, for each feature  $A$  and for each possible value  $a$  that  $A$  might take, divide  $D$  into two parts ( $D_1$  and  $D_2$ ) according to whether the sample points test “yes” or “no” for  $A = a$ , and use Eq. (5.25) to calculate the Gini index at  $A = a$ .
- (2) Among all the possible features  $A$  and all possible splitting points  $a$ , select the feature with the smallest Gini index and its corresponding splitting point as the optimal feature and the optimal splitting point. According to the optimal feature and the optimal splitting point, two sub-nodes are generated from the existing

node, and the training data is allocated to the two sub-nodes according to the features.

- (3) Call steps (1) and (2) recursively on the two sub-nodes until the stopping condition is satisfied.
- (4) Generate a CART decision tree.

The algorithm stops when the number of samples in the node is less than the predetermined threshold, the Gini index of the sample set is less than the predetermined threshold (samples basically belong to the same class), or there are no more features.

**Example 5.4** Apply the CART algorithm to the training dataset given in Table 5.1 to generate a decision tree.

**Solution** Firstly, the Gini index of each feature is calculated, and the optimal feature and its optimal splitting point are selected. Using the notations of Example 5.2,  $A_1$ ,  $A_2$ ,  $A_3$  and  $A_4$  indicate age, job status, housing ownership and credit rating. The values of 1, 2 and 3 for age are young, middle age and old. The numbers 1 and 2 indicate “Yes” and “No” for having a job and owning a house, and the numbers 1, 2 and 3 indicate having excellent, good, and fair credit, respectively.

Find the Gini index of feature  $A_1$ :

$$\text{Gini}(D, A_1 = 1) = \frac{5}{15} \left( 2 \times \frac{2}{5} \times \left( 1 - \frac{2}{5} \right) \right) + \frac{10}{15} \left( 2 \times \frac{7}{10} \times \left( 1 - \frac{7}{10} \right) \right) = 0.44$$

$$\text{Gini}(D, A_1 = 2) = 0.48$$

$$\text{Gini}(D, A_1 = 3) = 0.44$$

Since  $\text{Gini}(D, A_1 = 1)$  and  $\text{Gini}(D, A_1 = 3)$  are equal and minimal, both  $A_1 = 1$  and  $A_1 = 3$  can be chosen as optimal splitting points of  $A_1$ .

Find the Gini index of features  $A_2$  and  $A_3$ :

$$\text{Gini}(D, A_2 = 1) = 0.32$$

$$\text{Gini}(D, A_3 = 1) = 0.27$$

Since both  $A_2$  and  $A_3$  have only one splitting point, they are the optimal splitting points.

Solve for the Gini index of feature  $A_4$ :

$$\text{Gini}(D, A_4 = 1) = 0.36$$

$$\text{Gini}(D, A_4 = 2) = 0.47$$

$$\text{Gini}(D, A_4 = 3) = 0.32$$

$\text{Gini}(D, A_4 = 3)$  is the smallest, so  $A_4 = 3$  is the optimal splitting point for  $A_4$ .

Among the features of  $A_1$ ,  $A_2$ ,  $A_3$  and  $A_4$ ,  $\text{Gini}(D, A_3 = 1) = 0.27$  is the smallest, so feature  $A_3$  is selected as the optimal feature and  $A_3 = 1$  as its optimal

splitting point. The root node then generates two child nodes, one of which is a leaf node. For the other node, we continue to use the above method to select the optimal feature and its optimal splitting point from  $A_1, A_2, A_4$ , and the result is  $A_2 = 1$ . This computation shows that the resulting nodes are all leaf nodes.

For this problem, the decision tree generated by the CART algorithm is consistent with that generated by the ID3 algorithm.

### 5.5.2 CART Pruning

The CART pruning algorithm prunes some subtrees from the bottom of the “fully grown” decision tree, making the decision tree smaller (making the model simpler) and allowing for more accurate predictions of unknown data. The CART pruning algorithm consists of two steps: First, the decision tree  $T_0$  generated by the generation algorithm is pruned from the bottom of  $T_0$  to the root node of  $T_0$  to form a subtree sequence  $\{T_0, T_1, \dots, T_n\}$ ; then, the subtree sequence is tested on an independent validation dataset by cross-validation, from which the optimal subtree is selected.

#### 5.5.2.1 To Prune and Form a Subtree Sequence

During pruning, the loss function of the subtree is calculated as

$$C_\alpha(T) = C(T) + \alpha|T| \quad (5.26)$$

where  $|T|$  is an arbitrary subtree,  $C(T)$  is the prediction error of training data (e.g. Gini index),  $|T|$  is the number of leaf nodes of the subtree,  $\alpha \geq 0$  is the parameter, and  $C_\alpha(T)$  is the overall loss of  $T$  when the parameter is  $\alpha$ . The parameter  $\alpha$  weighs the degree of fitting on the training data and the complexity of the model.

For a fixed  $\alpha$ , there must be a subtree that minimizes the loss function  $C_\alpha(T)$ , denoted as  $T_\alpha$ .  $T_\alpha$  is optimal when the loss function  $C_\alpha(T)$  is minimal. It is easy to verify that such an optimal subtree is unique. When  $\alpha$  is large, the optimal subtree  $T_\alpha$  tends to be small; when  $\alpha$  is small, the optimal subtree  $T_\alpha$  tends to be large. In the extreme case, when  $\alpha = 0$ , the overall tree is optimal. When  $\alpha \rightarrow \infty$ , a single-node tree consisting of root nodes is optimal.

Breiman et al. show that the tree can be pruned recursively. Increase  $\alpha$  from small to large ( $0 = \alpha_0 < \alpha_1 < \dots < \alpha_n < +\infty$ ), producing a series of intervals  $[\alpha_i, \alpha_{i+1}]$ ,  $i = 0, 1, \dots, n$ ; the sequence of subtrees obtained by pruning corresponds to the optimal subtree sequence  $\{T_0, T_1, \dots, T_n\}$  of the interval  $\alpha \in [\alpha_i, \alpha_{i+1}]$ ,  $i = 0, 1, \dots, n$ , and the subtrees in the sequence are nested.

Specifically, the pruning starts with the overall tree  $T_0$ . For any internal node  $t$  of  $T_0$ , the loss function for a single node tree with  $t$  is

$$C_\alpha(t) = C(t) + \alpha \quad (5.27)$$

The loss function of a subtree  $T_t$  with  $t$  as the root node is

$$C_\alpha(T_t) = C(T_t) + \alpha|T_t| \quad (5.28)$$

When  $\alpha = 0$  and  $\alpha$  is sufficiently small, we have the inequality

$$C_\alpha(T_t) < C_\alpha(t) \quad (5.29)$$

As  $\alpha$  increases, at a certain  $\alpha$  there is

$$C_\alpha(T_t) = C_\alpha(t) \quad (5.30)$$

When  $\alpha$  increases again, the Inequality (5.29) is reversed. As long as  $\alpha = \frac{C(t) - C(T_t)}{|T_t| - 1}$ ,  $T_t$  has the same loss function value as  $t$ . As  $t$  has fewer nodes, it is preferable to  $T_t$ , on which the pruning is performed.

To do this, for each internal node  $t$  in  $T_0$ , compute

$$g(t) = \frac{C(t) - C(T_t)}{|T_t| - 1} \quad (5.31)$$

It indicates the extent to which the overall loss function is reduced after pruning. Prune the  $T_t$  with the smallest  $g(t)$  in  $T_0$  and label the resulting subtree as  $T_1$ , with the smallest  $g(t)$  set as  $\alpha_1$ .  $T_1$  is the optimal subtree for the interval  $[\alpha_1, \alpha_2]$ .

This pruning continues until obtaining the root node. In the process, the value of  $\alpha$  is continuously increased to produce new intervals.

### 5.5.2.2 The Optimal Subtree $T_\alpha$ is Selected by Cross-Validation Among the Pruned Subtree Sequences $T_0, T_1, \dots, T_n$

Specifically, the squared error or the Gini index of each subtree in the subtree sequence  $T_0, T_1, \dots, T_n$  is tested with the independent validation dataset. The decision tree with the smallest squared error or Gini index is considered to be the optimal decision tree. In the subtree sequence, each subtree  $T_1, \dots, T_n$  corresponds to a parameter  $\alpha_1, \alpha_2, \dots, \alpha_n$ . Therefore, when the optimal subtree  $T_k$  is determined, the corresponding  $\alpha_k$  is also determined, i.e., the optimal decision tree  $T_\alpha$  is obtained.

The CART pruning algorithm is written as follows.

#### Algorithm 5.7 (The CART Pruning Algorithm)

**Input:** The decision tree  $T_0$  generated by the CART algorithm;

**Output:** The optimal decision tree  $T_\alpha$ .

- (1) Set  $k = 0$  and  $T = T_0$ .
- (2) Set  $\alpha = +\infty$ .
- (3) Compute  $C(T_t)$  bottom-up for each internal node  $t$ ,  $|T_t|$  and

$$g(t) = \frac{C(t) - C(T_t)}{|T_t| - 1}$$

$$\alpha = \min(\alpha, g(t))$$

Here,  $T_t$  indicates the subtree with  $t$  as the root node,  $C(T_t)$  is the prediction error on the training data, and  $|T_t|$  is the number of leaf nodes of  $T_t$ .

- (4) The tree  $T$  is obtained by pruning the internal node  $t$  of  $g(t) = \alpha$  and deciding its class by majority voting on the leaf node  $t$ .
- (5) Set  $k = k + 1$ ,  $\alpha_k = \alpha$ ,  $T_K = T$ .
- (6) If  $T_K$  is not a tree consisting of the root node and two leaf nodes, go back to step (2); otherwise let  $T_K = T_n$ .
- (7) Use the cross-validation method to select the optimal subtree  $T_\alpha$  in the subtree sequence  $T_0, T_1, \dots, T_n$ .

## Summary

- (1) A classification decision tree model is a tree structure representing the classification of instances based on features. A decision tree can be converted into a collection of if-then rules, or it can be seen as a conditional probability distribution of classes defined over the feature space partition.
- (2) Decision tree learning aims to construct a decision tree that fits the training data well and has low complexity. The direct selection of the optimal decision tree from the possible decision trees is an NP-complete problem. In reality, heuristics are used to learn sub-optimal decision trees.

The decision tree learning algorithm consists of 3 parts: feature selection, tree generation and tree pruning. Commonly used algorithms are ID3, C4.5 and CART.

- (3) The purpose of feature selection is to select features that can classify the training data. The key to feature selection is the criterion. The following guidelines are commonly used:

- (1) The information gain of sample set  $D$  to feature  $A$  (ID3)

$$g(D, A) = H(D) - H(D|A)$$

$$H(D) = - \sum_{k=1}^K \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

$$H(D|A) = \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i)$$

$H(D)$  is the entropy of dataset  $D$ ,  $H(D_i)$  is the entropy of dataset  $D_i$ , and  $H(D|A)$  is the conditional entropy of dataset  $D$  to feature  $A$ .  $D_i$  is the sample subset in  $D$  when  $D_i$  feature  $A$  takes the  $i$ -th value, and  $C_k$  is the sample subset in  $D$  that belongs to the  $k$ -th class.  $n$  is the number of values that feature  $A$  takes, and  $K$  is the number of classes.

- (2) The information gain ratio of sample set  $D$  to feature  $A$  (C4.5)

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)}$$

$g(D, A)$  is the information gain, and  $H_A(D)$  is the entropy of  $D$  concerning the value of feature  $A$ .

- (3) The Gini index of sample set  $D$  (CART)

$$\text{Gini}(D) = 1 - \sum_{k=1}^K \left( \frac{|C_k|}{|D|} \right)^2$$

The Gini index of set  $D$  under the condition of feature  $A$ :

$$\text{Gini}(D, A) = \frac{|D_1|}{|D|} \text{Gini}(D_1) + \frac{|D_2|}{|D|} \text{Gini}(D_2)$$

- (4) The generation of decision trees. The commonly-used criteria of feature selection are maximum information gain, maximum information gain ratio, or minimum Gini index. decision trees are often generated recursively from the root node by calculating the information gain or other indices. This is equivalent to using the information gain or other criteria to continuously select locally optimal features or to split the training set into subsets that can be correctly classified.
- (5) The pruning of decision trees. We need to prune the generated decision tree to simplify it due to the problem of over-fitting. The pruning of decision trees is often done by cutting some leaf nodes or subtrees above the leaf nodes and taking the parent node or root node as the new leaf node.

## Further Reading

There are many references introducing decision tree learning, such as ID3 [1], C4.5 [2], and CART [3, 4]. A general introduction to decision tree learning can be found in the references [5–7]. A similar classification method to the decision tree is the decision list, which is interchangeable with the decision tree [8]. The learning method of decision list can be seen in reference [9, 10].

**Table 5.2** Training data

$x_i$	1	2	3	4	5	6	7	8	9	10
$y_i$	4.50	4.75	4.91	5.34	5.80	7.05	7.90	8.23	8.70	9.00

## Exercises

- 5.1 Generate the decision tree using information gain ratio (Algorithm C4.5) based on the training dataset given in Table 5.1.
- 5.2 Generate a binary regression tree using Squared Error Loss criterion and the training data given in Table 5.2.
- 5.3 Prove that in the CART pruning algorithm, when  $\alpha$  is determined, there exists a unique minimal subtree  $T_\alpha$  that minimizes the loss function  $C_\alpha(T)$ .
- 5.4 Prove that the subtree sequence  $\{T_0, T_1, \dots, T_n\}$  found by the CART pruning algorithm is the optimal subtree  $T_\alpha$  in the interval  $\alpha \in [\alpha_i, \alpha_{i+1})$ , where  $i = 0, 1, \dots, n, 0 = \alpha_0 < \alpha_1 < \dots < \alpha_n < +\infty$ .

## References

1. Quinlan JR. Induction of decision trees. *Mach Learn.* 1986;1(1):81–106.
2. Quinlan J R. C4. 5: programs for machine learning. Morgan Kaufmann, 1992.
3. Breiman L, Friedman J, Olshen RA, et al. Classification and regression trees. Wadsworth, 1984.
4. Ripley BD. Pattern recognition and neural networks. Cambridge University Press, 1996.
5. Liu B. Web data mining: exploring hyperlinks, contents and usage data. Springer Verlag, 2006.
6. Laurent H, Rivest RL. Constructing optimal binary decision trees is NP-complete. *Inf Process Lett.* 1976;5(1):15–7.
7. Hastie T, Tibshirani R, Friedman J. The elements of statistical learning: data mining, inference, and prediction. Springer-Verlag, 2001.
8. Yamanishi K. A learning criterion for stochastic rules. *Mach Learn.* 1992;9(2–3):165–203.
9. Li H, Yamanishi K. Text classification using ESC-based stochastic decision lists. *Inf Process Manage.* 2002;38(3):343–61.
10. Li H, Abe N. Generalizing case frames using a thesaurus and the MDL principle. *Comput Linguis.* 1998;24(2):217–244.

# Chapter 6

## Logistic Regression and Maximum Entropy Model

Logistic regression is a classical classification method in machine learning. The maximum entropy is a criterion for probabilistic model learning, which is extended to classification problems to obtain the maximum entropy model. The logistic regression model and the maximum entropy model are both log-linear models. This chapter first introduces the logistic regression model, then the maximum entropy model, and finally describes the learning algorithms for logistic regression and maximum entropy models, including the improved iterative scaling algorithm and the Quasi-Newton method.

### 6.1 Logistic Regression Model

#### 6.1.1 Logistic Distribution

First, we introduce the logistic distribution.

**Definition 6.1 (Logistic Distribution)** Let  $X$  be a continuous random variable.  $X$  obeys a logistic distribution means that  $X$  has the following distribution function and density function:

$$F(x) = P(X \leq x) = \frac{1}{1 + e^{-(x-\mu)/\gamma}} \quad (6.1)$$

$$f(x) = F'(x) = \frac{e^{-(x-\mu)/\gamma}}{\gamma(1 + e^{-(x-\mu)/\gamma})^2} \quad (6.2)$$

where  $\mu$  is the position parameter, and  $\gamma > 0$  is the shape parameter.

The graphs of the logistic distribution's density function  $f(x)$  and distribution function  $F(x)$  are shown in Fig. 6.1. The distribution function is a logistic function,



**Fig. 6.1** The probability density function and distribution function of the logistic distribution

and its graph is an S-shaped curve (sigmoid curve). The curve is centrosymmetric at the point  $(\mu, \frac{1}{2})$ , and it satisfies

$$F(-x + \mu) - \frac{1}{2} = -F(x + \mu) + \frac{1}{2}$$

The curve grows faster near the center and slower at the ends. The smaller the value of the shape parameter  $\gamma$ , the faster the curve grows near the center.

### 6.1.2 Binomial Logistic Regression Model

The binomial logistic regression model is a classification model represented by the conditional probability distribution  $P(Y|X)$  in the form of a parametric logistic distribution. Here, the random variable  $X$  is a real number and the random variable  $Y$  takes the value of 1 or 0. We estimate the model parameters by means of supervised learning.

**Definition 6.2 (Logistic Regression Model)** The binomial logistic regression model is a conditional probability distribution as follows:

$$P(Y = 1|x) = \frac{\exp(w \cdot x + b)}{1 + \exp(w \cdot x + b)} \quad (6.3)$$

$$P(Y = 0|x) = \frac{1}{1 + \exp(w \cdot x + b)} \quad (6.4)$$

where  $x \in \mathbf{R}^n$  is the input,  $Y \in \{0, 1\}$  is the output,  $w \in \mathbf{R}^n$  and  $b \in \mathbf{R}^n$  are the parameters,  $w$  is called the weight vector,  $b$  is called the bias, and  $w \cdot x$  is the inner product of  $w$  and  $x$ .

For a given input instance  $x$ ,  $P(Y = 1|x)$  and  $P(Y = 0|x)$  can be obtained according to Eqs. (6.3) and (6.4). Logistic regression compares the magnitude of the two conditional probability values and assigns the instance  $x$  to the class with the larger probability value.

Sometimes the weight vector and the input vector are expanded for convenience. They are still denoted as  $w$  and  $x$ , i.e.,  $w = (w^{(1)}, w^{(2)}, \dots, w^{(n)}, b)^T$ ,  $x = (x^{(1)}, x^{(2)}, \dots, x^{(n)}, 1)^T$ . At this point, the logistic regression model is as follows:

$$P(Y = 1|x) = \frac{\exp(w \cdot x)}{1 + \exp(w \cdot x)} \quad (6.5)$$

$$P(Y = 0|x) = \frac{1}{1 + \exp(w \cdot x)} \quad (6.6)$$

The characteristics of the logistic regression model are now examined. The odds of an event are the ratio of the probability that the event will occur to the one that the event will not. If the probability of an event occurring is  $p$ , then the odds of the event are expressed as  $\frac{p}{1-p}$ . The log odds or logit function of that event is:

$$\text{log it}(p) = \log \frac{p}{1-p}$$

For logistic regression, from Eqs. (6.5) and (6.6), we can obtain

$$\log \frac{P(Y = 1|x)}{1 - P(Y = 1|x)} = w \cdot x$$

It means that in a logistic regression model, the log odds of the output  $Y = 1$  is a linear function of the input  $x$ . In other words, the log odds of the output  $Y = 1$  is a model represented by a linear function of the input  $x$ , i.e., a logistic regression model.

From another perspective, consider the linear function  $w \cdot x$  that classifies the input  $x$ , whose domain of values is all real numbers. Note that here  $x \in \mathbb{R}^{n+1}$ ,  $w \in \mathbb{R}^{n+1}$ . By the logistic regression model definition Formula (6.5), the linear function  $w \cdot x$  can convert into probability:

$$P(Y = 1|x) = \frac{\exp(w \cdot x)}{1 + \exp(w \cdot x)}$$

At this time, the closer the linear function value is to positive infinity, the closer the probability value is to 1; the closer the linear function value is to negative infinity, the closer the probability value is to 0 (as shown in Fig. 6.1). Such a model is a logistic regression model.

### 6.1.3 Model Parameter Estimation

When the logistic regression model is learning, for a given training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \mathbf{R}^n$ ,  $y_i \in \{0, 1\}$ , the model parameters can be estimated by applying the maximum likelihood estimation method to obtain the logistic regression model.

Assume that:

$$P(Y = 1|x) = \pi(x), P(Y = 0|x) = 1 - \pi(x)$$

The likelihood function is

$$\prod_{i=1}^N [\pi(x_i)]^{y_i} [1 - \pi(x_i)]^{1-y_i}$$

the log-likelihood function is

$$\begin{aligned} L(w) &= \sum_{i=1}^N [y_i \log \pi(x_i) + (1 - y_i) \log(1 - \pi(x_i))] \\ &= \sum_{i=1}^N [y_i \log \frac{\pi(x_i)}{1 - \pi(x_i)} + \log(1 - \pi(x_i))] \\ &= \sum_{i=1}^N [y_i(w \cdot x_i) - \log(1 + \exp(w \cdot x_i))] \end{aligned}$$

Find the maximum value of  $L(w)$  to get the estimated value of  $w$ .

In this way, the problem becomes an optimization problem with the log-likelihood function as the objective function. The methods commonly used in logistic regression learning are the gradient descent method and the Quasi-Newton method.

Assume that  $w$ 's estimated value of maximum likelihood is  $\hat{w}$ , then the learned logistic regression model is

$$\begin{aligned} P(Y = 1|x) &= \frac{\exp(\hat{w} \cdot x)}{1 + \exp(\hat{w} \cdot x)} \\ P(Y = 0|x) &= \frac{1}{1 + \exp(\hat{w} \cdot x)} \end{aligned}$$

### 6.1.4 Multi-nomial Logistic Regression

The logistic regression model introduced above is a binomial classification model for binary classification. It can be extended to a multi-nominal logistic regression model for multi-class classification. Assuming that the value set of the discrete random variable  $Y$  is  $\{1, 2, \dots, K\}$ , the multi-nominal logistic regression model is

$$P(Y = k|x) = \frac{\exp(w_k \cdot x)}{1 + \sum_{k=1}^{K-1} \exp(w_k \cdot x)}, \quad k = 1, 2, \dots, K-1 \quad (6.7)$$

$$P(Y = K|x) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(w_k \cdot x)} \quad (6.8)$$

where  $x \in \mathbb{R}^{n+1}$ ,  $w \in \mathbb{R}^{n+1}$ .

The parameter estimation method of binomial logistic regression can also be extended to multinomial logistic regression.

## 6.2 Maximum Entropy Model

The maximum entropy model is derived from the principle of maximum entropy. Here we first describe the general maximum entropy principle, then explain the derivation of the maximum entropy model, and finally present the form of maximum entropy model learning.

### 6.2.1 Maximum Entropy Principle

The of maximum entropy principle is a criterion for probabilistic model learning. It states that when learning probability models, the model with the maximum entropy of all possible probability models (distributions) is the best. Constraints are usually used to determine the set of probability models. Therefore, the maximum entropy principle can also be expressed as selecting the model with the maximum entropy from the set of models that satisfy the constraints.

Assuming that the probability distribution of the discrete random variable  $X$  is  $P(X)$ , its entropy (cf. Sect. 5.2.2) is

$$H(P) = - \sum_x P(x) \log P(x) \quad (6.9)$$

The entropy satisfies the following inequalities:

$$0 \leq H(P) \leq \log|X|$$

where  $|X|$  is the number of  $X$  values, and the equal sign on the right holds if and only if the distribution of  $X$  is uniform. It means that the entropy is maximum when  $X$  obeys a uniform distribution.

Intuitively, the maximum entropy principle believes that the probability model to be selected must first meet the existing facts, i.e., the constraint conditions. Without more information, those uncertain parts are “equiprobable”. The maximum entropy principle expresses equiprobability by maximizing the entropy. The “equiprobability” is not easy to operate, and the entropy is an optimizable numerical index.

First, the maximum entropy principle is introduced with a simple example.<sup>1</sup>

**Example 6.1** Supposing that the random variable  $X$  has five values  $\{A, B, C, D, E\}$ , the probability of each value  $P(A), P(B), P(C), P(D), P(E)$  is to be estimated.

**Solution** These probability values satisfy the following constraint:

$$P(A) + P(B) + P(C) + P(D) + P(E) = 1$$

There are infinite probability distributions that satisfy this constraint. If there is no other information, the probability distribution still needs to be estimated. One possible way is to assume that the probabilities of taking each value in this distribution are equal:

$$P(A) = P(B) = P(C) = P(D) = P(E) = \frac{1}{5}$$

Equal probabilities indicate ignorance of the facts. However, in the absence of more information, this judgment is reasonable.

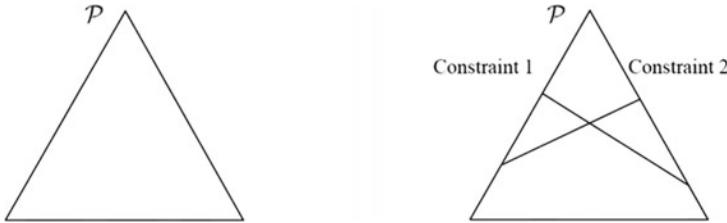
Sometimes, some constraints on the probability value can be obtained from some prior knowledge, for example:

$$\begin{aligned} P(A) + P(B) &= \frac{3}{10} \\ P(A) + P(B) + P(C) + P(D) + P(E) &= 1 \end{aligned}$$

There are still infinite probability distributions that satisfy these two constraints. In the absence of other information, it can be considered that  $A$  and  $B$  are equiprobable, and  $C, D$  and  $E$  are equiprobable. Then,

---

<sup>1</sup> This example comes from reference [1].



Probabilistic Model Space    A collection of models that meet the constraints

**Fig. 6.2** Probability model set

$$P(A) = P(B) = \frac{3}{20}$$

$$P(C) = P(D) = P(E) = \frac{7}{30}$$

If there is a third constraint:

$$P(A) + P(C) = \frac{1}{2}$$

$$P(A) + P(B) = \frac{3}{10}$$

$$P(A) + P(B) + P(C) + P(D) + P(E) = 1$$

we can continue to estimate the probability distribution by obtaining equal probability under the constraint conditions. We will not continue the discussion here. The above approach to probabilistic model learning follows precisely the maximum entropy principle.

Figure 6.2 provides a geometric explanation of the probability model selection using the maximum entropy principle. The set of probability models  $\mathcal{P}$  can be represented by a simplex<sup>2</sup> in Euclidean space, as the triangle (2-simplex) on the left. A point represents a model, and the entire simplex represents a collection of models. A straight line on the right-hand diagram corresponds to a constraint condition, and the intersection of the straight lines corresponds to a set of models that satisfy all constraint conditions. Generally, there are infinitely many such models. The purpose of learning is to select the optimal model from the set of possible models, and the principle of maximum entropy gives a criterion for optimal model selection.

---

<sup>2</sup> A simplex is the convex hull of the set of  $n+1$  affinely independent points in an  $n$ -dimensional Euclidean space.

### 6.2.2 Definition of Maximum Entropy Model

The maximum entropy principle is a general principle of machine learning, and it is applied to classification to obtain the maximum entropy model.

$X \in \mathcal{X} \subseteq \mathbb{R}^n$  Suppose the classification model is a conditional probability distribution  $P(Y|X)$ , as the input and  $Y \in \mathcal{Y}$  as the output.  $\mathcal{X}$  and  $\mathcal{Y}$  are the sets of inputs and outputs, respectively. This model represents the output  $Y$  with a conditional probability  $P(Y|X)$  for a given input  $X$ .

Given a training dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

The object of learning is to use the maximum entropy principle to select the best classification model.

Consider first the conditions that the model should satisfy. With the given training dataset, the empirical distribution of the joint distribution  $P(X, Y)$  and the empirical distribution of the marginal distribution  $P(X)$ , denoted as  $\tilde{P}(X, Y)$  and  $\tilde{P}(X)$  respectively, can be determined.

$$\begin{aligned}\tilde{P}(X = x, Y = y) &= \frac{\nu(X = x, Y = y)}{N} \\ \tilde{P}(X = x) &= \frac{\nu(X = x)}{N}\end{aligned}$$

where  $\nu(X = x, Y = y)$  represents the frequency of the sample  $(x, y)$  in the training data,  $\nu(X = x)$  represents the frequency of the input  $x$  in the training data, and  $N$  represents the training sample capacity.

The feature function  $f(x, y)$  is used to describe a certain fact between the input  $x$  and the output  $y$ . It is defined as

$$f(x, y) = \begin{cases} 1, & x \text{ and } y \text{ satisfy a certain fact} \\ 0, & \text{otherwise} \end{cases}$$

It is a binary function<sup>3</sup> whose value is 1 when  $x$  and  $y$  satisfy this fact and 0 otherwise.

The expected value of the feature function  $f(x, y)$  for the empirical distribution  $P(Y|X)$  is denoted as  $E_{\tilde{P}}(f)$  :

$$E_{\tilde{P}}(f) = \sum_{x,y} \tilde{P}(x, y) f(x, y)$$

---

<sup>3</sup> In general, the feature function can be any real-valued function.

The expected value of the feature function  $f(x, y)$  concerning the model  $P(Y|X)$  and the empirical distribution  $\tilde{P}(X)$  is expressed by  $E_P(f)$ :

$$E_P(f) = \sum_{x,y} \tilde{P}(x) P(y|x) f(x, y)$$

If the model has access to the information of the training data, then we can assume that the two expected values are equal, i.e.,

$$E_P(f) = E_{\tilde{P}}(f) \quad (6.10)$$

or

$$\sum_{x,y} \tilde{P}(x) P(y|x) f(x, y) = \sum_{x,y} \tilde{P}(x, y) f(x, y) \quad (6.11)$$

We use Eqs. (6.10) or (6.11) as the constraint on model learning. If there are  $n$  feature functions  $f_i(x, y), i = 1, 2, \dots, n$ , then there are  $n$  constraints.

**Definition 6.3 (Maximum Entropy Model)** Assume that the set of models satisfying all constraints is

$$C = \{P \in \mathcal{P} | E_P(f_i) = E_{\tilde{P}}(f_i), i = 1, 2, \dots, n\} \quad (6.12)$$

the conditional entropy defined on the conditional probability distribution  $P(Y|X)$  is

$$H(P) = - \sum_{x,y} \tilde{P}(x) P(y|x) \log P(y|x) \quad (6.13)$$

then the model with the largest conditional entropy  $H(P)$  in the model set  $C$  is called the maximum entropy model. The logarithm in the formula is the natural logarithm.

### 6.2.3 Learning of the Maximum Entropy Model

The learning process of the maximum entropy model is the process of solving the maximum entropy model, which can be formalized as a constrained optimization problem.

For a given training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$  and the feature function  $f_i(x, y), i = 1, 2, \dots, n$ , the learning of the maximum entropy model is equivalent to the constrained optimization problem.

$$\max_{P \in C} H(P) = - \sum_{x,y} \tilde{P}(x) P(y|x) \log P(y|x)$$

$$s.t. E_P(f_i) = E_{\tilde{P}}(f_i), i = 1, 2, \dots, n$$

$$\sum_y P(y|x) = 1$$

As is customary in the optimization problem, the maximum value problem is rewritten as an equivalent minimum value problem:

$$\min_{P \in C} -H(P) = \sum_{x,y} \tilde{P}(x) P(y|x) \log P(y|x) \quad (6.14)$$

$$s.t. E_P(f_i) - E_{\tilde{P}}(f_i) = 0, i = 1, 2, \dots, n \quad (6.15)$$

$$\sum_y P(y|x) = 1 \quad (6.16)$$

The solution to the constrained optimization problem by Eqs. (6.14–6.16) is the one to the maximum entropy model learning. The specific derivation is given below.

Here, the original problem of constrained optimization is transformed into a dual problem of unconstrained optimization.<sup>4</sup> The original problem is solved by solving the dual problem.

First, introduce the Lagrangian multipliers  $w_0, w_1, w_2, \dots, w_n$ , and define the Lagrangian function  $L(P, w)$ :

$$\begin{aligned} L(P, w) &\equiv -H(P) + w_0 \left( 1 - \sum_y P(y|x) \right) + \sum_{i=1}^n w_i (E_{\tilde{P}}(f_i) - E_P(f_i)) \\ &= \sum_{x,y} \tilde{P}(x) P(y|x) \log P(y|x) + w_0 \left( 1 - \sum_y P(y|x) \right) + \\ &\quad \sum_{i=1}^n w_i \left( \sum_{x,y} \tilde{P}(x, y) f_i(x, y) - \sum_{x,y} \tilde{P}(x) P(y|x) f_i(x, y) \right) \end{aligned} \quad (6.17)$$

The original problem of optimization is

$$\min_{P \in C} \max_w L(P, w) \quad (6.18)$$

The dual problem is

$$\max_w \min_{P \in C} L(P, w) \quad (6.19)$$

---

<sup>4</sup> See Appendix C.

Since the Lagrange function  $L(P, w)$  is a convex function of  $P$ , the solution to the original problem (6.18) and the one to the dual problem (6.19) are equivalent. In this way, the original problem (6.18) can be solved by solving the dual problem (6.19).

First, solve the minimization problem  $\min_{P \in C} L(P, w)$  inside the dual problem (6.19).  $\min_{P \in C} L(P, w)$  is a function of  $w$ , denote as

$$\Psi(w) = \min_{P \in C} L(P, w) = L(P_w, w) \quad (6.20)$$

$\Psi(w)$  is called the dual function and its solution is denote as

$$P_w = \arg \min_{P \in C} L(P, w) = P_w(y|x) \quad (6.21)$$

Specifically, find the partial derivative of  $L(P, w)$  with respect to  $P(y|x)P(y|x)$

$$\begin{aligned} \frac{\partial L(P, w)}{\partial P(y|x)} &= \sum_{x,y} \tilde{P}(x)(\log P(y|x) + 1) - \sum_y w_0 \\ &\quad - \sum_{x,y} \left( \tilde{P}(x) \sum_{i=1}^n w_i f_i(x, y) \right) \\ &= \sum_{x,y} \tilde{P}(x) \left( \log P(y|x) + 1 - w_0 - \sum_{i=1}^n w_i f_i(x, y) \right) \end{aligned}$$

Let the partial derivative be equal to 0. In the case of  $\tilde{P}(x) > 0$ , the solution is

$$P(y|x) = \exp \left( \sum_{i=1}^n w_i f_i(x, y) + w_0 - 1 \right) = \frac{\exp \left( \sum_{i=1}^n w_i f_i(x, y) \right)}{\exp(1 - w_0)}$$

since  $\sum_y P(y|x) = 1$ , we have

$$P_w(y|x) = \frac{1}{Z_w(x)} \exp \left( \sum_{i=1}^n w_i f_i(x, y) \right) \quad (6.22)$$

where,

$$Z_w(x) = \sum_y \exp \left( \sum_{i=1}^n w_i f_i(x, y) \right) \quad (6.23)$$

$Z_w(x)$  is called the normalization factor;  $f_i(x, y)$  is the feature function; and  $w_i$  is the weight of the feature. The model  $P_w = P_w(y|x)$  expressed by Eq. (6.22) and Eq. (6.23) is the maximum entropy model. Here,  $w$  is the parameter vector in the maximum entropy model.

After that, solve the maximization problem outside the dual problem

$$\max_w \Psi(w) \quad (6.24)$$

denote its solution as  $w^*$ , i.e.,

$$w^* = \arg \max_w \Psi(w) \quad (6.25)$$

It means that the optimization algorithm can be applied to maximize the dual function  $\Psi(w)$  to obtain  $w^*$ , which can be used to express  $P^* \in \mathcal{C}$ . Here,  $P^* = P_{w^*} = P_{w^*}(y|x)$  is the learned optimal model (maximum entropy model). In other words, the learning of the maximum entropy model boils down to the maximization of the dual function  $\Psi(w)$ .

**Example 6.2** Learning the maximum entropy model in Example 6.1.

**Solution** For convenience,  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $E$  are denoted by  $y_1$ ,  $y_2$ ,  $y_3$ ,  $y_4$ ,  $y_5$ , respectively, so that the optimization problem of maximum entropy model learning is

$$\begin{aligned} \min -H(P) &= \sum_{i=1}^5 P(y_i) \log P(y_i) \\ \text{s.t. } P(y_1) + P(y_2) &= \tilde{P}(y_1) + \tilde{P}(y_2) = \frac{3}{10} \\ \sum_{i=1}^5 P(y_i) &= \sum_{i=1}^5 \tilde{P}(y_i) = 1 \end{aligned}$$

Introduce the Lagrangian multiplier  $w_0$ ,  $w_1$ , and define the Lagrangian function

$$\begin{aligned} L(P, w) &= \sum_{i=1}^5 P(y_i) \log p(y_i) + w_1 \left( P(y_1) + P(y_2) - \frac{3}{10} \right) \\ &\quad + w_0 \left( \sum_{i=1}^5 P(y_i) - 1 \right) \end{aligned}$$

According to the Lagrange duality, the solution to the original optimization problem can be obtained by solving the dual optimization problem. Solve for

$$\max_w \min_P L(P, w)$$

First, solve the minimization problem of  $L(P, w)$  on  $P$ . To this end, fix  $w_0$  and  $w_1$ , and find the partial derivative

$$\begin{aligned}\frac{\partial L(P, w)}{\partial P(y_1)} &= 1 + \log P(y_1) + w_1 + w_0 \\ \frac{\partial L(P, w)}{\partial P(y_2)} &= 1 + \log P(y_2) + w_1 + w_0 \\ \frac{\partial L(P, w)}{\partial P(y_3)} &= 1 + \log P(y_3) + w_0 \\ \frac{\partial L(P, w)}{\partial P(y_4)} &= 1 + \log P(y_4) + w_0 \\ \frac{\partial L(P, w)}{\partial P(y_5)} &= 1 + \log P(y_5) + w_0\end{aligned}$$

Let each partial derivative be 0, and the solution is

$$\begin{aligned}P(y_1) = P(y_2) &= e^{-w_1-w_0-1} \\ P(y_3) = P(y_4) = P(y_5) &= e^{-w_0-1}\end{aligned}$$

thus,

$$\begin{aligned}\min_P L(P, w) &= L(P_w, w) = -2e^{-w_1-w_0-1} \\ &\quad - 3e^{-w_0-1} - \frac{3}{10}w_1 - w_0\end{aligned}$$

Then solve the maximum problem of  $L(P_w, w)$  on  $w$ :

$$\max_P L(P_w, w) = -2e^{-w_1-w_0-1} - 3e^{-w_0-1} - \frac{3}{10}w_1 - w_0$$

Find the partial derivatives of  $L(P_w, w)$  on  $w_0$  and  $w_1$ , respectively. Set them to 0, and obtain

$$\begin{aligned}e^{-w_1-w_0-1} &= \frac{3}{20} \\ e^{-w_0-1} &= \frac{7}{30}\end{aligned}$$

Then the required probability distribution is obtained

$$\begin{aligned}P(y_1) = P(y_2) &= \frac{3}{20} \\ P(y_3) = P(y_4) = P(y_5) &= \frac{7}{30}\end{aligned}$$

### 6.2.4 Maximum Likelihood Estimation

The maximum entropy model is a conditional probability distribution represented by Eqs. (6.22) and (6.23), as can be seen from the above maximum entropy model learning. It is shown below that the maximization of the dual function is equivalent to the maximum likelihood estimation of the maximum entropy model.

Given the empirical probability distribution of training data  $\tilde{P}(X, Y)$ , the log-likelihood function of the conditional probability distribution  $P(Y|X)$  is expressed as

$$L_{\tilde{P}}(P_w) = \log \prod_{x,y} P(y|x)^{\tilde{P}(x,y)} = \sum_{x,y} \tilde{P}(x,y) \log P(y|x)$$

When the conditional probability distribution  $P(Y|X)$  is the maximum entropy model (6.22) and (6.23), the log-likelihood function  $L_{\tilde{P}}(P_w)$  is.

$$\begin{aligned} L_{\tilde{P}}(P_w) &= \sum_{x,y} \tilde{P}(x,y) \log P(y|x) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n w_i f_i(x,y) - \sum_{x,y} \tilde{P}(x,y) \log Z_w(x) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n w_i f_i(x,y) - \sum_x \tilde{P}(x) \log Z_w(x) \end{aligned} \quad (6.26)$$

Look again at the dual function  $\Psi(w)$ , which can be obtained from Eq. (6.17) and Eq. (6.20).

$$\begin{aligned} \Psi(w) &= \sum_{x,y} \tilde{P}(x) P_w(y|x) \log P_w(y|x) \\ &\quad + \sum_{i=1}^n w_i \left( \sum_{x,y} \tilde{P}(x) f_i(x,y) - \sum_{x,y} \tilde{P}(x) P_w(y|x) f_i(x,y) \right) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n w_i f_i(x,y) + \sum_{x,y} \tilde{P}(x) P_w(y|x) \left( \log P_w(y|x) - \sum_{i=1}^n w_i f_i(x,y) \right) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n w_i f_i(x,y) - \sum_{x,y} \tilde{P}(x) P_w(y|x) \log Z_w(x) \\ &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n w_i f_i(x,y) - \sum_{x,y} \tilde{P}(x) \log Z_w(x) \end{aligned} \quad (6.27)$$

The last step uses  $\sum_y P(y|x) = 1$ .

Compare Eqs. (6.26) and (6.27), it can be concluded that

$$\Psi(w) = L_{\tilde{P}}(P_w)$$

Since the dual function  $\Psi(w)$  is equivalent to the log-likelihood function  $L_{\tilde{P}}(P_w)$ , the fact is proved that the dual function maximization in the maximum entropy model learning is equivalent to the maximum likelihood estimation of the maximum entropy model.

In this way, the learning problem of the maximum entropy model transforms into maximizing the log-likelihood function or the dual function.

The maximum entropy model can be written in a more general form.

$$Pw(y|x) = \frac{1}{Z_w(x)} \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right) \quad (6.28)$$

where

$$Z_w(x) = \sum_y \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right) \quad (6.29)$$

Here,  $x \in R^n$  is the input,  $y \in \{1, 2, \dots, K\}$  is the output,  $w \in R^n$  is the weight vector, and  $f_i(x, y)$ ,  $i = 1, 2, \dots, n$  is any real valued feature function.

The maximum entropy model and logistic regression model have similar forms, and they are also called log linear model. Model learning is the maximum likelihood estimation or regularized maximum likelihood estimation of the model under given training data conditions.

## 6.3 Optimization Algorithm of Model Learning

The learning of the logistic regression model and maximum entropy model is an optimization problem with the likelihood function as the objective function. The problem is usually solved by the iterative algorithm. From the perspective of optimization, the objective function in the case has excellent properties. It is a smooth convex function, so many optimization methods are applicable for finding the global optimal solution. Commonly used methods are the improved iterative scaling method, gradient descent method, Newton's method, or Quasi-Newton method. Newton's method or Quasi-Newton method generally converges faster.

The following introduces the maximum entropy model learning algorithm based on the improved iterative scaling method and Quasi-Newton method. See Appendix A for the gradient descent method.

### 6.3.1 Improved Iterative Scaling

Improved iterative scaling (IIS) is an optimization algorithm for maximum entropy model learning.

The maximum entropy model is known as

$$P_w(y|x) = \frac{1}{Z_w(x)} \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right)$$

where

$$Z_w(x) = \sum_y \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right)$$

The log-likelihood function is

$$L(w) = \sum_{x,y} \tilde{P}(x, y) \sum_{i=1}^n w_i f_i(x, y) - \sum_x \tilde{P}(x) \log Z_w(x)$$

The object is to learn the model parameters by maximum likelihood estimation, i.e., to find the maximum value of the log-likelihood function.

The idea of IIS is that, assuming that the current parameter vector of the maximum entropy model is  $w = (w_1, w_2, \dots, w_n)^T$ , we want to find a new parameter vector  $w + \delta = (w_1 + \delta_1, w_2 + \delta_2, \dots, w_n + \delta_n)^T$  to increase the value of the log-likelihood function of the model. If there is such a parameter vector updating method  $\tau : w \rightarrow w + \delta$ , then this method can be used repeatedly until the maximum value of the log-likelihood function is found.

For a given empirical distribution  $\tilde{P}(x, y)$ , with the model parameter varying from  $w$  to  $w + \delta$ , the change of the log-likelihood function is

$$\begin{aligned} L(w + \delta) - L(w) &= \sum_{x,y} \tilde{P}(x, y) \log P_{w+\delta}(y|x) - \sum_{x,y} \tilde{P}(x, y) \log P_w(y|x) \\ &= \sum_{x,y} \tilde{P}(x, y) \sum_{i=1}^n \delta_i f_i(x, y) - \sum_x \tilde{P}(x) \log \frac{Z_{w+\delta(x)}}{Z_w(x)} \end{aligned}$$

Using the inequality

$$-\log \alpha \geq 1 - \alpha, \quad \alpha > 0$$

to establish the lower bound of the change in log-likelihood function

$$\begin{aligned}
L(w + \delta) - L(w) &\geq \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n \delta_i f_i(x,y) \\
&\quad + 1 - \sum_x \tilde{P}(x) \frac{Z_{w+\delta}(x)}{Z_w(x)} \\
&= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n \delta_i f_i(x,y) + 1 \\
&\quad - \sum_x \tilde{P}(x) \sum_y P_w(y|x) \exp \sum_{i=1}^n \delta_i f_i(x,y)
\end{aligned}$$

Denote the right end as

$$\begin{aligned}
A(\delta|w) &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n \delta_i f_i(x,y) + 1 \\
&\quad - \sum_x \tilde{P}(x) \sum_y P_w(y|x) \exp \sum_{i=1}^n \delta_i f_i(x,y)
\end{aligned}$$

Then, we have

$$L(w + \delta) - L(w) \geq A(\delta|w)$$

That is,  $A(\delta|w)$  is a lower bound of the change in the log-likelihood function.

If an appropriate  $\delta$  can be found to increase the lower bound  $A(\delta|w)$ , then the log-likelihood function will also increase. However, the  $\delta$  in the function  $A(\delta|w)$  is a vector with multiple variables, which are not easy to be optimized simultaneously. IIS tries to optimize only one of the variables  $\delta_i$  at a time while fixing the other variables  $\delta_j$ ,  $i \neq j$ .

To achieve this goal, IIS further reduces the lower bound  $A(\delta|w)$ . Specifically, it introduces a quantity  $f^\#(x, y)$ ,

$$f^\#(x, y) = \sum_i f_i(x, y)$$

Since  $f_i$  is a binary function,  $f^\#(x, y)$  represents the number of times all features appear in  $(x, y)$ . In this way,  $A(\delta|w)$  can be rewritten as

$$\begin{aligned}
A(\delta|w) &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^n \delta_i f_i(x,y) + 1 \\
&\quad - \sum_x \tilde{P}(x) \sum_y P_w(y|x) \exp(f^\#(x,y) \sum_{i=1}^n \frac{\delta_i f_i(x,y)}{f^\#(x,y)}) \tag{6.30}
\end{aligned}$$

Using the convexity of the exponential function and the fact that for any  $i$ , there is  $\frac{f_i(x, y)}{f^\#(x, y)} \geq 0$  and  $\sum_{i=1}^n \frac{f_i(x, y)}{f^\#(x, y)} = 1$ , with Jensen's inequality, we obtain

$$\exp\left(\sum_{i=1}^n \frac{f_i(x, y)}{f^\#(x, y)} \delta_i f^\#(x, y)\right) \leq \sum_{i=1}^n \frac{f_i(x, y)}{f^\#(x, y)} \exp(\delta_i f^\#(x, y))$$

Then Formula (6.30) can be rewritten as

$$\begin{aligned} A(\delta|w) &\geq \sum_{x, y} \tilde{P}(x, y) \sum_{i=1}^n \delta_i f_i(x, y) + 1 - \\ &\quad \sum_x \tilde{P}(x) \sum_y P_w(y|x) \sum_{i=1}^n \left( \frac{f_i(x, y)}{f^\#(x, y)} \right) \exp(\delta_i f^\#(x, y)) \end{aligned} \quad (6.31)$$

Denote the right side of Inequality (6.31) as

$$\begin{aligned} B(\delta|w) &= \sum_{x, y} \tilde{P}(x, y) \sum_{i=1}^n \delta_i f_i(x, y) + 1 \\ &\quad - \sum_x \tilde{P}(x) \sum_y P_w(y|x) \sum_{i=1}^n \left( \frac{f_i(x, y)}{f^\#(x, y)} \right) \exp(\delta_i f^\#(x, y)) \end{aligned}$$

Then we get

$$L(w + \delta) - L(w) \geq B(\delta|w)$$

Here,  $B(\delta|w)$  is a new (relatively noncompact) lower bound for the change in the log-likelihood function.

Find the partial derivative of  $\delta_i$  to  $B(\delta|w)$ :

$$\frac{\partial B(\delta|w)}{\partial \delta_i} = \sum_{x, y} \tilde{P}(x, y) f_i(x, y) - \sum_x \tilde{P}(x) \sum_y P_w(y|x) f_i(x, y) \exp(\delta_i f^\#(x, y)) \quad (6.32)$$

In Eq. (6.32), there are no other variables except  $\delta_i$ . Let the partial derivative be 0 to get

$$\sum_{x, y} \tilde{P}(x) P_w(y|x) f_i(x, y) \exp(\delta_i f^\#(x, y)) = E_{\tilde{P}}(f_i) \quad (6.33)$$

Thus, solving Eq. (6.33) for  $\delta_i$  in turn yields  $\delta$ .

This gives an iterative algorithm for finding the optimal solution of  $w$ , the improved iterative scaling algorithm (IIS).

**Algorithm 6.1 (Improved Iterative Scaling Algorithm IIS)**

Input: feature function  $f_1, f_2, \dots, f_n$ ; the empirical distribution  $\tilde{P}(X, Y)$ , the model  $P_w(y|x)$ ;

Output: the optimal parameter value  $w_i^*$ ; the optimal model  $P_{w^*}$ .

- (1) For all  $i \in \{1, 2, \dots, n\}$ , take the initial value  $w_i = 0$ .
- (2) For each  $i \in \{1, 2, \dots, n\}$

(a) Let  $\delta_i$  be the solution of the equation

$$\sum_{x,y} \tilde{P}(x) P(y|x) f_i(x, y) \exp(\delta_i f^\#(x, y)) = E_{\tilde{P}}(f_i)$$

Here,

$$f^\#(x, y) = \sum_{i=1}^n f_i(x, y)$$

- (b) Update the value of  $w_i$ :  $w_i \leftarrow w_i + \delta_i$ .
- (3) (If not all  $w_i$  converge, repeat step (2)).

The key step in this algorithm is (a), which solves Eq. (6.33) for  $\delta_i$ . If.

$f^\#(x, y)$  is a constant, i.e., for any  $x, y$ ,  $f^\#(x, y) = M$ , then  $\delta_i$  can be expressed explicitly as

$$\delta_i = \frac{1}{M} \log \frac{E_{\tilde{P}}(f_i)}{E_P(f_i)} \quad (6.34)$$

If  $f^\#(x, y)$  is not a constant, then  $\delta_i$  must be found numerically. A simple and effective method is Newton's method. Equation (6.33) is expressed by  $g(\delta_i) = 0$ , and  $\delta_i^*$  is obtained by Newton's method through iteration so that  $g(\delta_i^*) = 0$ . The iterative formula is

$$\delta_i^{(k+1)} = \delta_i^{(k)} - \frac{g(\delta_i^{(k)})}{g'(\delta_i^{(k)})} \quad (6.35)$$

As long as the initial value  $\delta_i^{(0)}$  is properly selected, Newton's method converges constantly and rapidly since the Eq. (6.33) of  $\delta_i$  has a single solution.

### 6.3.2 Quasi-Newton Method

Newton's method or Quasi-Newton method can also be used to learn the maximum entropy model. See Appendix B.

For the maximum entropy model

$$P_w(y|x) = \frac{\exp(\sum_{i=1}^n w_i f_i(x, y))}{\sum_y \exp(\sum_{i=1}^n w_i f_i(x, y))}$$

with the objective function:

$$\min_{w \in R^n} f(w) = \sum_x \tilde{P}(x) \log \sum_y \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right) - \sum_{x, y} \tilde{P}(x, y) \sum_{i=1}^n w_i f_i(x, y)$$

Gradient:

$$g(w) = \left( \frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \dots, \frac{\partial f(w)}{\partial w_n} \right)^T$$

where

$$\frac{\partial f(w)}{\partial w_i} = \sum_{x, y} \tilde{P}(x) P_w(y|x) f_i(x, y) - E_{\tilde{P}}(f_i), \quad i = 1, 2, \dots, n$$

The corresponding Quasi-Newton BFGS algorithm is as follows.

#### Algorithm 6.2 (BFGS Algorithm for Maximum Entropy Model Learning)

Input: feature function  $f_1, f_2, \dots, f_n$ ; empirical distribution  $\tilde{P}(x, y)$ , objective function  $f(w)$ , gradient  $g(w) = \nabla f(w)$ , accuracy requirement  $\varepsilon$ ;

Output: the optimal parameter value  $w^*$ ; the optimal model  $P_{w^*}(y|x)$ .

- (1) Select the initial point  $w^{(0)}$ , take  $B_0$  as the positive definite symmetric matrix, and set  $K = 0$ ;
- (2) Compute  $g_k = g(w^{(k)})$ . If  $\|g_k\| < \varepsilon$ , then stop the computation and get  $w^* = w^{(k)}$ ; otherwise, turn to (3);
- (3)  $p_k$  is computed from  $B_k p_k = -g_k$ ;
- (4) One-dimensional search (line search): finding  $\lambda_k$  to obtain

$$f(w^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(w^{(k)} + \lambda p_k)$$

- (5) Set  $w^{(k+1)} = w^{(k)} + \lambda_k p_k$ ;

- (6) Compute  $g_{k+1} = g(w^{(k+1)})$ , if  $\|g_{k+1}\| < \varepsilon$ , stop the computation and get  $w^* = w^{(k+1)}$ ; otherwise, find  $B_{k+1}$  from the following equation:

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T \delta_k} - \frac{B_k \delta_k \delta_k^T B_k}{\delta_k^T B_k \delta_k}$$

where

$$y_k = g_{k+1} - g_k, \quad \delta_k = w^{(k+1)} - w^{(k)}$$

- (7) Set  $k = k + 1$ , turn to (3);

### Summary

1. Logistic regression model is a classification model represented by the following conditional probability distribution. It is applicable for binary or multi-class classification.

$$P(Y = k|x) = \frac{\exp(w_k \cdot x)}{1 + \sum_{k=1}^{K-1} \exp(w_k \cdot x)}, \quad k = 1, 2, \dots, K-1$$

$$P(Y = K|x) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(w_k \cdot x)}$$

here,  $x$  is the input feature and  $w$  is the weight of the feature.

The logistic regression model is derived from the logistic distribution, whose distribution function  $F(x)$  is an S-shaped function. The logistic regression model is a logarithmic probability model of output represented by the linear function of the input.

2. The maximum entropy model is a classification model denoted by the following conditional probability distribution. The maximum entropy model can also be applied to the binary or multi-class classification.

$$P_w(y|x) = \frac{1}{Z_w(x)} \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right)$$

$$Z(x) = \sum_y \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right)$$

where  $Z_w(x)$  is the normalization factor,  $f_i$  is the feature function, and  $w_i$  is the weight of the feature.

3. Maximum entropy models can be derived from the maximum entropy principle. The maximum entropy principle is a criterion for probabilistic model learning or estimation. It states that of the set of all possible probability models (distributions), the model with the highest entropy is the best.

The maximum entropy principle is applied to the learning of classification models with the following constrained optimization problems:

$$\begin{aligned} \min -H(P) &= \sum_{x,y} \tilde{P}(x)P(y|x)\log P(y|x) \\ \text{s.t. } P(f_i) - \tilde{P}(f_i) &= 0, \quad i = 1, 2, \dots, n \\ \sum_y P(y|x) &= 1 \end{aligned}$$

The maximum entropy model is obtained by solving the dual problem of the optimization problem.

4. The logistics regression and the maximum entropy models are both log-linear.
5. The learning of logistics regression and maximum entropy models generally uses the maximum likelihood estimation, or regularized maximum likelihood estimation. The logistics regression model and maximum entropy model learning can be formalized as an unconstrained optimization problem. The algorithms for solving this optimization problem include the improved iterative scaling method, the gradient descent method, and the Quasi-Newton method.

### Further Reading

See the literature [1] for the introduction to logistics regression and [2, 3] for the introduction to the maximum entropy model. Refer to [4] for the relationship between the logistics regression model and the naive Bayes model, [5] for the relationship between the logistics regression model and AdaBoost, and [6] for the relationship between the logistics regression model and the kernel function.

### Exercises

- 6.1 Confirm that the logistics distribution belongs to the family of exponential distributions.
- 6.2 Write a gradient descent algorithm for logistics regression model learning.
- 6.3 Propose the DFP algorithm for maximum entropy model learning. (See Appendix B for general DFP algorithms).

## References

1. Berger A, Della Pietra SD, Pietra VD. A maximum entropy approach to natural language processing. *Comput Linguist*. 1996;22(1):39–71.
2. Berger A. The improved iterative scaling algorithm: a gentle introduction. <http://www.cs.cmu.edu/afs/cs/user/aberger/www/ps/scaling.ps>.
3. Hastie T, Tibshirani R, Friedman J. The elements of statistical learning: data mining, inference, and prediction. Springer, 2001.
4. Mitchell TM. Machine learning. McGraw-Hill Companies, Inc. 1997.
5. Collins M, Schapire RE, Singer Y. Logistic regression, AdaBoost and Bregman distances. *Mach Learn*. 2002;48(1–3):253–85.
6. Canu S, Smola AJ. Kernel method and exponential family. *Neurocomputing*. 2005;69:714–20.

# Chapter 7

## Support Vector Machine

The Support Vector Machine(SVM) is a two-class classification model. The basic model of SVM is a linear classifier with the maximum margin defined on the feature space, and the maximum margin distinguishes it from a perceptron; The SVM also includes the kernel trick, which makes it a non-linear classifier. The learning strategy of SVM is margin maximization, which can be formalized as a problem of solving convex quadratic programming, which is also equivalent to the problem of minimizing a regularized hinge loss function. The learning algorithm for SVM is the optimization algorithm for solving convex quadratic programming.

The SVM learning methods include building models from simple to complex: the linear SVM in the linearly separable case, linear SVM, and non-linear SVM. The simple model is the basis and a special case of the complex model. When the training data is linearly separable, a linear classifier is learned by hard margin maximization, i.e., the linearly separable SVM, also known as hard margin SVM; when the training data is approximately linearly separable, a linear classifier, i.e., a linear SVM, also known as a soft margin SVM, is learned by soft margin maximization; when the training data is linearly inseparable, a non-linear SVM is learned by the kernel trick and soft margin maximization.

When the input space is a Euclidean space or a discrete set and the feature space is a Hilbert space, the kernel function represents the inner product between the feature vectors obtained by mapping the input from the input space to the feature space. A non-linear SVM can be learned by the kernel function, which is equivalent to implicitly learning a linear SVM in a high-dimensional feature space. Such a method is called the kernel method. The kernel method is a more general machine learning method than the SVM method.

Cortes and Vapnik proposed the linear Support Vector Machine, and Boser, Guyon, and Vapnik introduced the kernel trick to propose the non-linear SVM.

Based on the ideas above, this chapter will introduce three classes of SVMs, kernel functions, and a fast learning algorithm—the Sequential Minimum Optimization (SMO) algorithm.

## 7.1 Linear Support Vector Machine in the Linearly Separable Case and Hard Margin Maximization

### 7.1.1 Linear Support Vector Machine in the Linearly Separable Case

A two-class classification problem is being considered. Assume that the input and the feature spaces are different. The input space is a Euclidean space or discrete set, and the feature space is a Euclidean space or Hilbert space. The linear SVM in the linearly separable case and the linear SVM assume that the elements in the two spaces are in one-to-one correspondence and map the input from the input space to the feature vector in the feature space. Non-linear SVM maps the input to the feature vector by non-linear mapping from input space to feature space. Therefore, all inputs are transformed from the input space to the feature space, and the learning of the SVM is performed in the feature space.

Suppose a training dataset in a feature space is given

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i \in \mathcal{X} = \mathbb{R}^n$ ,  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ .  $x_i$  is the  $i$ -th feature vector, also called an instance, and  $y_i$  is the class label of  $x_i$ . When  $y_i = +1$ ,  $x_i$  is called a positive instance; when  $y_i = -1$ ,  $x_i$  is called a negative instance.  $(x_i, y_i)$  is called the sample point. Then suppose the training dataset is linearly separable (see Definition 2.2).

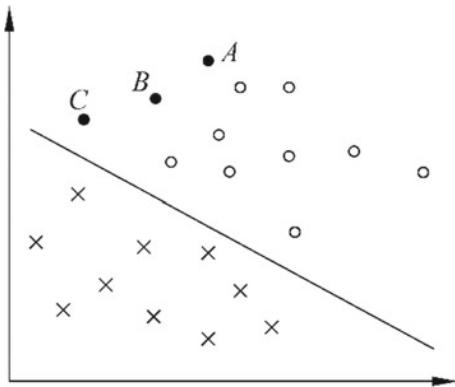
The learning goal is to find a separating hyperplane, which divides instances into different classes. The separating hyperplane corresponds to the equation  $w \bullet x + b = 0$ . It is determined by the normal vector  $w$  and the intercept  $b$  and can be represented by  $(w, b)$ . The separating hyperplane divides the feature space into two parts, including the positive class and the negative class. The one to which the normal vector points is positive class, the other is negative.

Generally, when the training dataset is linearly separable, there is an infinite number of separating hyperplanes that can correctly separate the two classes of data. The perceptron uses the strategy of minimum misclassification to find the separating hyperplane, but there are infinite solutions. The linear SVM in the linearly separable case uses margin maximization to find the optimal separating hyperplane, at which point the solution is unique.

**Definition 7.1 (Linear Support Vector Machine in the Linearly Separable Case)** Given a training dataset in the linearly separable case, the separating hyperplane obtained by margin maximization or solving the corresponding convex quadratic programming problem equivalently is

$$w^* \bullet x + b^* = 0 \tag{7.1}$$

**Fig. 7.1** The two-class classification problem



The corresponding classification decision function is

$$f(x) = \text{sign}(w^* \bullet x + b^*) \quad (7.2)$$

which is called the linear SVM in the linearly separable case.

Consider the classification problem in the 2-dimensional feature space shown in Fig. 7.1. “o” indicates positive instances, and “x” represents negative instances. The training dataset is linearly separable, and then many straight lines can correctly separate the two data classes. The linear SVM in the linearly separable case corresponds to the line that successfully separates the data classes with the maximum margin, as shown in Fig. 7.1.

The maximum margin and the corresponding constrained optimization problems will be described below. Here we first introduce the concepts of the function margin and the geometric margin.

### 7.1.2 Function Margin and Geometric Margin

In Fig. 7.1, there are three points, A, B and C, representing three instances, all on the positive class side of the separating hyperplane. Their classes are predicted. Point A is far from the separating hyperplane, and if it is predicted to be a positive class, the prediction is more accurate. Point C is closer to the separating hyperplane, and it is less confident if it is predicted to be a positive class. Point B is between points A and C, and the reliability of predicting it to be a positive class is also between A and C.

Generally, the distance between a point and the separating hyperplane can indicate the degree of certainty of classification prediction. When hyperplane  $w \bullet x + b = 0$  is determined,  $|w \bullet x + b|$  can relatively represent the distance between point  $x$  and the hyperplane. The consistency of the sign of  $w \bullet x + b$  with that of the class label  $y$  indicates whether the classification is correct. Therefore,  $y(w \bullet x + b)$  could

be used to indicate the correctness and the certainty of classification, which is the concept of functional margin.

**Definition 7.2 (Functional Margin)** With regard to the given training dataset  $T$  and the hyperplane  $(w, b)$ , define the functional margin of hyperplane  $(w, b)$  on sample point  $(x_i, y_i)$  as

$$\hat{\gamma}_i = y_i(w \bullet x_i + b) \quad (7.3)$$

Define the functional margin of hyperplane  $(w, b)$  on training dataset  $T$  as the minimum of the functional margin of hyperplane  $(w, b)$  on all sample points  $(x_i, y_i)$  in  $T$ , i.e.

$$\hat{\gamma} = \min_{i=1, \dots, N} \hat{\gamma}_i \quad (7.4)$$

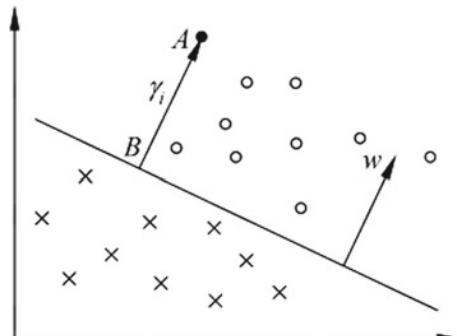
The functional margin indicates the correctness and the degree of confidence of classification prediction. But only the functional margin is not enough while selecting the separating hyperplane. Because by changing  $w$  and  $b$  proportionally, for instance, to  $2w$  and  $2b$ , the hyperplane does not change, but the functional margin becomes twice the original. This fact reveals that certain constraints, such as normalization,  $\|w\| = 1$ , can be added to the normal vector  $w$  of the separating hyperplane such that the margin is deterministic. The functional margin then becomes the geometric margin.

The hyperplane  $(w, b)$  and its normal vector  $w$  are given in Fig. 7.2. Point  $A$  represents an instance  $x_i$  whose class is labeled  $y_i = +1$ . The distance between point  $A$  and the hyperplane  $(w, b)$  is given by the segment AB, denoted as  $\gamma_i$ .

$$\gamma_i = \frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}$$

where  $\|w\|$  is the L2 parametrization of  $w$ . This is the case where point  $A$  is on the positive side of the hyperplane. If point  $A$  is on the negative side of the hyperplane,

**Fig. 7.2** Geometric margin



i.e.,  $y_i = -1$ , then the distance between the point and the hyperplane is

$$\gamma_i = -\left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right)$$

In general, when the sample points  $(x_i, y_i)$  are correctly classified by the hyperplane  $(w, b)$ , the distance between the point  $x_i$  and the hyperplane  $(w, b)$  is

$$\gamma_i = y_i \left( \frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|} \right)$$

The concept of geometric margin is derived from this fact.

**Definition 7.3 (Geometric Margin)** For a given training dataset  $T$  and hyperplane  $(w, b)$ , define the geometric margin of the hyperplane  $(w, b)$  about the sample points  $(x_i, y_i)$  as

$$\gamma_i = y_i \left( \frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|} \right) \quad (7.5)$$

Define the geometric margin of the hyperplane  $(w, b)$  with respect to the training dataset  $T$  as the minimum of the geometric margin of the hyperplane  $(w, b)$  with respect to all sample points  $(x_i, y_i)$  in  $T$ , i.e.

$$\gamma = \min_{i=1, \dots, N} \gamma_i \quad (7.6)$$

The geometric margin of the hyperplane  $(w, b)$  about the sample points  $(x_i, y_i)$  is generally the signed distance from the instance point to the hyperplane, which is the distance from the instance point to the hyperplane when the sample points are correctly classified by the hyperplane.

From the definition of the functional margin and the geometric margin (Eqs. (7.3)~(7.6)), it follows that the functional margin and geometric margin have the following relationship:

$$\gamma_i = \frac{\hat{\gamma}_i}{\|w\|} \quad (7.7)$$

$$\gamma = \frac{\hat{\gamma}}{\|w\|} \quad (7.8)$$

If  $\|w\| = 1$ , then the functional margin and the geometric margin are equal. If the hyperplane parameters  $w$  and  $b$  change proportionally (the hyperplane does not change), the functional margin also changes in the same proportion, while the geometric margin remains unchanged.

### 7.1.3 Maximum Margin

The basic idea of SVM learning is to find the separating hyperplane that correctly divides the training dataset and maximizes the geometric margin. For linearly separable training datasets, there are infinite linearly separable separating hyperplanes (equivalent to the perceptron), but the separating hyperplane with the maximum geometric margin is unique. The maximization here is also called the hard margin maximization (as opposed to the soft margin maximization when the training dataset is approximately linearly separable, which will be discussed).

The intuitive interpretation of margin maximization is that finding the hyperplane with the largest geometric margin for the training dataset means classifying the training data with sufficiently large confidence. That is, not only the positive and negative instance points are separated, but also the most difficult instance points (the points closest to the hyperplane) are separated with sufficient confidence. Such a hyperplane should have good classification prediction capability for unknown new instances.

#### 7.1.3.1 Maximum Margin Separating Hyperplane

In the following, we consider how to find a separating hyperplane with the largest geometric margin, i.e., the maximum margin separating hyperplane. Specifically, this problem can be expressed as the following constrained optimization problem:

$$\max_{w, b} \gamma \quad (7.9)$$

s.t.

$$y_i \left( \frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|} \right) \geq \gamma, \quad i = 1, 2, \dots, N \quad (7.10)$$

i.e., we expect to maximize the geometric margin  $\gamma$  of the hyperplane  $(w, b)$  with respect to the training dataset. The constraint indicates that the geometric margin of the hyperplane  $(w, b)$  concerning each training sample point is at least  $\gamma$ .

$$\max_{w, b} \frac{\hat{\gamma}}{\|w\|} \quad (7.11)$$

s.t.

$$y_i(w \cdot x_i + b) \geq \hat{\gamma}, \quad i = 1, 2, \dots, N \quad (7.12)$$

The value of the function margin  $\hat{\gamma}$  does not affect the solution of the optimization problem. Suppose that we  $w$  and  $b$  are changed proportionally to  $\lambda w$  and  $\lambda b$ , and the function margin becomes  $\lambda \hat{\gamma}$ . This change in the function margin does not affect the optimization problem above, nor does it affect the optimization of the objective

function, i.e., it produces an equivalent optimization problem. In this way,  $\hat{\gamma} = 1$  can be taken. Substituting  $\hat{\gamma} = 1$  into the optimization problem above and noting that maximizing  $\frac{1}{\|w\|}$  and minimizing  $\frac{1}{2}\|w\|^2$  is equivalent, the following optimal learning problem of the linear SVM in the linearly separable case is obtained:

$$\min_{w, b} \frac{1}{2}\|w\|^2 \quad (7.13)$$

$$\begin{aligned} & \text{s.t.} \\ & y_i(w \cdot x_i + b) - 1 \geq 0, \quad i = 1, 2, \dots, N \end{aligned} \quad (7.14)$$

This is a convex quadratic programming problem.

The convex optimization problem is a constrained optimization problem

$$\min_w f(w) \quad (7.15)$$

$$\begin{aligned} & \text{s.t.} \\ & g_i(w) \leq 0, \quad i = 1, 2, \dots, k \end{aligned} \quad (7.16)$$

$$h_i(w) = 0, \quad i = 1, 2, \dots, l \quad (7.17)$$

where the objective function  $f(w)$  and the constraint function  $g_i(w)$  are both continuous and differentiable convex functions on  $R^n$ , and the constraint function  $h_i(w)$  is an affine function on  $R^n$ .<sup>1</sup>

When the objective function  $f(w)$  is a quadratic function and the constraint function  $g_i(w)$  is an affine function, the above convex optimization problem becomes a convex quadratic programming problem.

If the solutions  $w^*, b^*$  of the constrained optimization problem (7.13)~(7.14) are obtained, then we have the maximum separating hyperplane  $w^* \cdot x + b^* = 0$  and the classification decision function  $f(x) = \text{sign}(w^* \cdot x + b^*)$ , i.e., the linearly separable SVM model.

In summary, the following learning algorithm for the linearly separable SVM, the maximum margin method, is available.

### Algorithm 7.1 (The Learning Algorithm of Linear SVM in the Linearly Separable Case—the Maximum Margin Method)

**Input:** linearly separable dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \mathcal{X} = R^n$ ,  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ ;

**Output:** the maximum margin separating hyperplane and classification decision function.

- (1) Construct and solve constrained the optimization problem:

---

<sup>1</sup>  $f(x)$  is called an affine function, if it satisfies  $f(x) = a \cdot x + b$ ,  $a \in R^n$ ,  $b \in R$ ,  $x \in R$

$$\begin{aligned} & \min_{w, b} \frac{1}{2} \|w\|^2 \\ & \text{s.t.} \\ & y_i(w \cdot x_i + b) - 1 \geq 0, \quad i = 1, 2, \dots, N \end{aligned}$$

Find the optimal solution  $w^*, b^*$ .

- (2) Obtain the separating hyperplane:

$$w^* \cdot x + b^* = 0$$

and the classification decision function

$$f(x) = \text{sign}(w^* \cdot x + b^*)$$

### 7.1.3.2 The Existence and Uniqueness of the Maximum Margin Separating Hyperplane

The maximum margin separating hyperplane of the linearly separable training dataset exists and is unique.

**Theorem 7.1 (The Existence and Uniqueness of the Maximum Margin Separating Hyperplane)** If the training dataset  $T$  is linearly separable, the maximum margin separating hyperplane that can completely separate the sample points in the training dataset exactly exists and is unique.

#### Proof

- (1) Existence

Since the training dataset is linearly separable, there must be feasible solutions for the optimization problem (7.13)~(7.14) in Algorithm 7.1. Since the objective function has a lower bound, the optimization problem (7.13)~(7.14) must have a solution denoted as  $(w^*, b^*)$ . Since there are both positive and negative class points in the training dataset,  $(w, b) = (0, b)$  is not an optimal feasible solution. The optimal solution  $(w^*, b^*)$  must satisfy  $w^* \neq 0$ . The existence of the separating hyperplane is thus known.

- (2) Uniqueness

First, prove the uniqueness of  $w^*$  in the solution of the optimization problem (7.13)~(7.14). Suppose there are two optimal solutions  $(w_1^*, b_1^*)$  and  $(w_2^*, b_2^*)$  for the problem (7.13)~(7.14). Obviously  $\|w_1^*\| = \|w_2^*\| = c$ , where  $c$  is a constant. Let  $w = \frac{w_1^* + w_2^*}{2}$ ,  $b = \frac{b_1^* + b_2^*}{2}$ , and it is easy to know that  $(w, b)$  is a feasible solution for the problem (7.13)~(7.14), so

$$c \leq \|w\| \leq \frac{1}{2} \|w_1^*\| + \frac{1}{2} \|w_2^*\| = c$$

The above equation shows that the inequality sign in the equation can be changed to an equal sign, i.e.,  $\|w\| = \frac{1}{2}\|w_1^*\| + \frac{1}{2}\|w_2^*\|$ , and thus we have  $w_1^* = \lambda w_2^*$ ,  $|\lambda|=1$ . If  $\lambda = -1$ , then  $w = 0$  and  $(w, b)$  is not a feasible solution for the problem (7.13)~(7.14), which is contradictory. So there must be  $\lambda = 1$ , i.e.

$$w_1^* = w_2^*$$

Therefore, the two optimal solutions  $(w_1^*, b_1^*)$  and  $(w_2^*, b_2^*)$  can be written as  $(w^*, b_1^*)$  and  $(w^*, b_2^*)$ . Then prove that  $b_1^* = b_2^*$ . Let  $x'_1$  and  $x'_2$  be the points in the set  $\{x_i | y_i = +1\}$  corresponding to  $(w^*, b_1^*)$  and  $(w^*, b_2^*)$ , respectively, at which the equality sign of the inequality holds, and  $x''_1$  and  $x''_2$  be the points in the set  $\{x_i | y_i = -1\}$  corresponding to  $(w^*, b_1^*)$  and  $(w^*, b_2^*)$ , respectively, which makes the equality sign of the inequality hold. Then, from  $b_1^* = -\frac{1}{2}(w^* \cdot x'_1 + w^* \cdot x''_1)$  and  $b_2^* = -\frac{1}{2}(w^* \cdot x'_2 + w^* \cdot x''_2)$ , we have

$$b_1^* - b_2^* = -\frac{1}{2}[w^* \cdot (x'_1 - x'_2) + w^* \cdot (x''_1 - x''_2)]$$

Since

$$\begin{aligned} w^* \cdot x'_2 + b_1^* &\geq 1 = w^* \cdot x'_1 + b_1^* \\ w^* \cdot x'_1 + b_2^* &\geq 1 = w^* \cdot x'_2 + b_2^* \end{aligned}$$

$w^* \cdot (x'_1 - x'_2) = 0$  holds. The same is true for  $w^* \cdot (x''_1 - x''_2) = 0$ . Thus,

$$b_1^* - b_2^* = 0$$

Known from  $w_1^* = w_2^*$  and  $b_1^* = b_2^*$ , the two optimal solutions  $(w_1^*, b_1^*)$  and  $(w_2^*, b_2^*)$  are the same, thus the uniqueness of the solution is proved.

The separating hyperplane is proved to be unique according to the uniqueness of the solution to the problem (7.13)~(7.14).

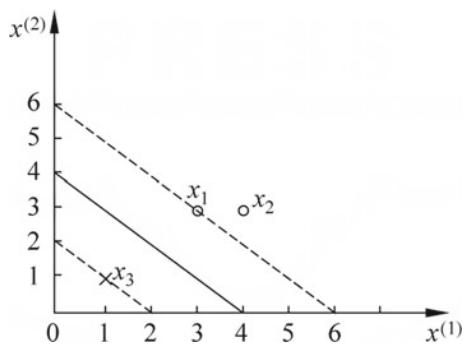
- (3) The separating hyperplane can separate the two classes of points in the training dataset completely and correctly.

It can be known from the fact that the solution satisfies the problem's constraints.

### 7.1.3.3 Support Vector and MarginBoundary

In the linearly separable case, the instance of the sample points in the training dataset closest to the separating hyperplane is called a support vector. The support vector is the point at which the equal sign of the constraint Eq. (7.14) holds, i.e.

$$y_i(w \cdot x_i + b) - 1 = 0$$

**Fig. 7.3** Support vector

For the positive instance point of  $y_i = +1$ , the support vector is on the hyperplane

$$H_1 : w \cdot x + b = 1$$

For the negative instance point of  $y_i = -1$ , the support vector is on the hyperplane

$$H_2 : w \cdot x + b = -1$$

As shown in Fig. 7.3, the points on  $H_1$  and  $H_2$  are the support vectors.

Note that  $H_1$  and  $H_2$  are parallel with no instance point falling between them. A long band is formed between  $H_1$  and  $H_2$ , with the separating hyperplane parallel to them and in the center of them. The width of the long belt, i.e., the distance between  $H_1$  and  $H_2$ , is called the margin. The margin depends on the normal vector  $w$  of the separating hyperplane, which is equal to  $\frac{2}{\|w\|}$ .  $H_1$  and  $H_2$  are called margin boundaries.

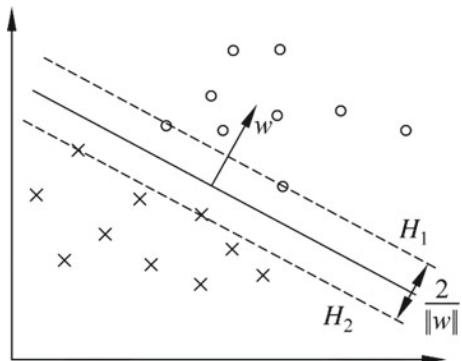
When deciding the separating hyperplane, only the support vector plays a role, while other instance points have no contribution. If the support vector is moved, the solution will change; however, if other instance points are moved outside the margin boundaries or even removed, the solution will not change. Since support vectors play a decisive role in determining the separating hyperplane, this classification model is called a Support Vector Machine. The number of support vectors is generally small, so the Support Vector Machine is determined by a few “important” training samples.

**Example 7.1** The data is the same as in Example 2.1. Give a training dataset shown in Fig. 7.4 with positive instance points  $x_1 = (3, 3)^T$ ,  $x_2 = (4, 3)^T$  and the negative instance point  $x_3 = (1, 1)^T$ . Try to find the maximum marginseparating hyperplane.

**Solution** Construct a constrained optimization problem based on the training dataset according to Algorithm 7.1:

$$\min_{w, b} \frac{1}{2} (w_1^2 + w_2^2)$$

**Fig. 7.4** Example of maximum separating hyperplane



$$\begin{aligned} \text{s.t. } & 3w_1 + w_2 + b \geq 1 \\ & 4w_1 + 3w_2 + b \geq 1 \\ & -w_1 - w_2 - b \geq 1 \end{aligned}$$

Find the solution of this optimization problem  $w_1 = w_2 = \frac{1}{2}$ ,  $b = -2$ . Then the maximum marginseparating hyperplane is

$$\frac{1}{2}x^{(1)} + \frac{1}{2}x^{(2)} - 2 = 0$$

where  $x_1 = (3, 3)^T$  and  $x_3 = (1, 1)^T$  are support vectors.

#### 7.1.4 Dual Algorithm of Learning

To solve the optimization problem (7.13)~(7.14) of the linear SVM in the linearly separable case, the optimal solution of the primal problem is obtained by solving the dual problem using Lagrange duality (see Appendix C), which is the dual algorithmof the linear SVM in the linearly separable case. The advantages of this method are: firstly, dual problems are often easier to solve; secondly, the kernel function is introduced naturally and then extended to the non-linear classification problem.

First construct the Lagrange function. In order to do this, the Lagrange multiplier  $\alpha_i \geq 0$ ,  $i = 1, 2, \dots, N$  is introduced for each inequality constraint (7.14), and the Lagrange function is defined as:

$$L = (w, b, a) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^N \alpha_i y_i (w \bullet x_i + b) + \sum_{i=1}^N \alpha_i \quad (7.18)$$

where  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_N)^T$  is the Lagrange multiplier vector.

According to Lagrange duality, the dual problem of the primal problem is a min–max problem:

$$\max_{\alpha} \min_{w, b} L(w, b, \alpha)$$

Therefore, in order to obtain the solution of the dual problem, it is necessary to find the minimum of  $L(w, b, \alpha)$  for  $w, b$  first, and then find the maximum for  $\alpha$ .

(1) Find  $\min_{w, b} L(w, b, \alpha)$ .

Take the partial derivatives of the Lagrange function  $L(w, b, \alpha)$  with respect to  $w$  and  $b$  and set them equal to 0.

$$\begin{aligned}\nabla_w L(w, b, \alpha) &= w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \\ \nabla_b L(w, b, \alpha) &= - \sum_{i=1}^N \alpha_i y_i = 0\end{aligned}$$

Then we get

$$w = \sum_{i=1}^N \alpha_i y_i x_i \quad (7.19)$$

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (7.20)$$

Substituting Eq. (7.19) into the Lagrange function (7.18) and using Eq. (7.20), we get

$$\begin{aligned}L(w, b, \alpha) &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) \\ &\quad - \sum_{i=1}^N \alpha_i y_i \left( \left( \sum_{j=1}^N \alpha_j y_j x_j \right) \bullet x_i + b \right) + \sum_{i=1}^N \alpha_i \\ &= - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) + \sum_{i=1}^N \alpha_i\end{aligned}$$

i.e.,

$$\min_{w, b} L(w, b, \alpha) = - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) + \sum_{i=1}^N \alpha_i$$

- (2) Find the maximum of  $\min_{w, b} L(w, b, \alpha)$  to  $\alpha$ , i.e., the dual problem

$$\begin{aligned} \max_{\alpha} & -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) + \sum_{i=1}^N \alpha_i \\ \text{s.t.} & \sum_{i=1}^N \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, 2, \dots, N \end{aligned} \tag{7.21}$$

Converting the objective function of Eq. (7.21) from seeking maximum to seeking minimum, the following equivalent dual optimization problem is obtained:

$$\begin{aligned} \min_{\alpha} & \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) - \sum_{i=1}^N \alpha_i \\ \text{s.t.} & \sum_{i=1}^N \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, 2, \dots, N \end{aligned} \tag{7.23}$$

Consider the primal optimization problem (7.13)~(7.14) and the dual optimization problem (7.22)~(7.24). The primal problem satisfies the condition of Theorem C.2, so there exist  $w^*$ ,  $\alpha^*$ ,  $\beta^*$  where  $w^*$  is the solution of the primal problem,  $\alpha^*$ ,  $\beta^*$  are the solutions of the dual problem. It means that solving the primal problem (7.13)~(7.14) can be converted to solving the dual problem (7.22)~(7.24).

For the linearly separable training dataset, suppose that the dual optimization problem (7.22)~(7.24) has a solution to  $\alpha$  as  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$ . The solution  $w^*$ ,  $\beta^*$  of the primal optimization problem (7.13)~(7.14) to  $(w, \beta)$  can be obtained from  $\alpha^*$ . There are theorems as follows.

**Theorem 7.2** Suppose  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_l^*)^T$  is the solution of the dual optimization problem (7.22)~(7.24), then there is a subscript  $j$ , making  $\alpha_j^* > 0$ , and the solution  $w^*$ ,  $b^*$  of the primal optimization problem (7.13)~(7.14) can be obtained as follows:

$$w^* = \sum_{i=1}^N \alpha_i^* y_i x_i \tag{7.25}$$

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i (x_i \cdot x_j) \quad (7.26)$$

**Proof** According to Theorem C.3, the KKT condition holds, so we get

$$\begin{aligned} \nabla_w L(w^*, b^*, \alpha^*) &= w^* - \sum_{i=1}^N \alpha_i^* y_i x_i = 0 \\ \nabla_b L(w^*, b^*, \alpha^*) &= - \sum_{i=1}^N \alpha_i^* y_i = 0 \\ \alpha_i^*(y_i(w^* \bullet x_i + b^*) - 1) &= 0, \quad i = 1, 2, \dots, N \\ y_i(w^* \bullet x_i + b^*) - 1 &\geq 0, \quad i = 1, 2, \dots, N \\ \alpha_i^* &\geq 0, \quad i = 1, 2, \dots, N \end{aligned} \quad (7.27)$$

From this we get

$$w^* = \sum_i \alpha_i^* y_i x_i$$

where at least one  $\alpha_j^*$  makes  $\alpha_j^* > 0$  (The proof by contradiction is used. Assuming  $\alpha^* = 0$ , from Eq. (7.27) we know that  $w^* = 0$ . But  $w^* = 0$  is not a solution to the primal optimization problem (7.13)~(7.14) and leads to contradiction). For  $j$ , we have

$$y_j(w^* \bullet x_j + b^*) - 1 = 0 \quad (7.28)$$

Substituting Eq. (7.25) into Eq. (7.28) and noticing that  $y_j^2 = 1$ , we obtain

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i (x_i \bullet x_j)$$

From this theorem, the separating hyperplane can be written as

$$\sum_{i=1}^N \alpha_i^* y_i (x \bullet x_i) + b^* = 0 \quad (7.29)$$

The classification decision function can be written as

$$f(x) = \text{sign}\left(\sum_{i=1}^N \alpha_i^* y_i (x \bullet x_i) + b^*\right) \quad (7.30)$$

It indicates that the classification decision function depends only on the inner product of the input  $x$  and the training sample input. Equation (7.30) is called the dual form of the linear SVM in the linearly separable case.

In summary, for a given linearly separable training dataset, we can first find the solution  $\alpha^*$  of the dual problem (7.22)~(7.24); then use Eqs. (7.25) and (7.26) to find the solution  $w^*, b^*$  of the primal problem, thus obtaining the separating hyperplane and classification decision function. This algorithm is called the dual learning algorithm of the linear SVM in the linearly separable case, which is the basic algorithm for learning linearly separable SVM.

**Algorithm 7.2 (The Learning Algorithm of the Linear Support Vector Machine in the Linearly Separable Case)**

Input: Linearly separable training set  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \chi = R^n, y_i \in \mathcal{Y} = \{-1, +1\}, i = 1, 2, \dots, N$ ;

Output: Separating hyperplane and classification decision function.

- (1) Construct and solve the constrained optimization problems

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) - \sum_{i=1}^N \alpha_i$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 1, 2, \dots, N$$

Find the optimal solution  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$ .

- (2) Calculate

$$w^* = \sum_{i=1}^N \alpha_i^* y_i x_i$$

and select a positive component  $\alpha_j^*$  of  $\alpha^*$ ,  $\alpha_j^* > 0$ , then calculate

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i (x_i \bullet x_j)$$

- (3) Find the separating hyperplane

$$w^* \bullet x + b^* = 0$$

and the classification decision function:

$$f(x) = \text{sign}(w^* \bullet x + b^*)$$

In the linearly separable SVM, it can be seen from Eqs. (7.25) and (7.26) that  $w^*$  and  $b^*$  depend only on the sample points  $(x_i, y_i)$  in the training data corresponding to  $\alpha_i^* > 0$ , while other sample points do not affect  $w^*$  and  $b^*$ . We refer to the instance points  $x_i \in R^n$  corresponding to  $\alpha_i^* > 0$  in the training data as the support vectors.

**Definition 7.4 (Support Vector)** Considering the primal optimization problem (7.13)~(7.14) and the dual optimization problem (7.22)~(7.24), the instances  $x_i \in R^n$  in the training dataset corresponding to the sample points  $(x_i, y_i)$  where  $\alpha_i^* > 0$  are called the support vectors.

By this definition, the support vector must be on the margin boundary. From the complementary conditions of KKT, we have

$$\alpha_i^*(y_i(w^* \bullet x_i + b^*) - 1) = 0, \quad i = 1, 2, \dots, N$$

Corresponding to the instance  $x_i$  where  $\alpha_i^* > 0$ , there is

$$y_i(w^* \bullet x_i + b^*) - 1 = 0$$

or

$$w^* \bullet x_i + b^* = \pm 1$$

i.e.,  $x_i$  must be on the margin boundary. The definition of support vector here is consistent with the version given earlier.

**Example 7.2** The training data is the same as in Example 7.1. As shown in Fig. 7.4, the positive instance points are  $x_1 = (3, 3)^T$ ,  $x_2 = (4, 3)^T$ , and the negative instance point is  $x_3 = (1, 1)^T$ . Try Algorithm 7.2 to find the linear Support Vector Machine in the linearly separable case.

**Solution** According to the given data, the dual problem is

$$\begin{aligned}
& \min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) - \sum_{i=1}^N \alpha_i \\
& = \frac{1}{2} (18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 \\
& \quad + 42\alpha_1\alpha_2 - 12\alpha_1\alpha_3 - 14\alpha_2\alpha_3) \\
& \quad - \alpha_1 - \alpha_2 - \alpha_3 \\
& \text{s.t.} \\
& \alpha_1 + \alpha_2 - \alpha_3 = 0 \\
& \alpha_i \geq 0, \quad i = 1, 2, 3
\end{aligned}$$

Solve this optimization problem. Substitute  $\alpha_3 = \alpha_1 + \alpha_2$  into the objective function and denote it as

$$s(\alpha_1, \alpha_2) = 4\alpha_1^2 + \frac{13}{2}\alpha_2^2 + 10\alpha_1\alpha_2 - 2\alpha_1 - 2\alpha_2$$

Take the partial derivative of  $\alpha_1$  and  $\alpha_2$  and set it to 0. It is easy to see that  $s(\alpha_1, \alpha_2)$  takes the extreme value at point  $(\frac{3}{2}, -1)^T$ , but the constraint  $\alpha_2 \geq 0$  is not satisfied at that point, so the minimum value should be reached on the boundary.

When  $\alpha_1 = 0$ , the minimum value is  $s(0, \frac{2}{13}) = -\frac{2}{13}$ ; when  $\alpha_2 = 0$ , the minimum value is  $s(\frac{1}{4}, 0) = -\frac{1}{4}$ . Thus  $s(\alpha_1, \alpha_2)$  reaches the minimum at  $\alpha_1 = \frac{1}{4}$ ,  $\alpha_2 = 0$ , at which time,  $\alpha_3 = \alpha_1 + \alpha_2 = \frac{1}{4}$ .

Thus, the instance points  $x_1, x_3$  corresponding to  $\alpha_1^* = \alpha_2^* = \frac{1}{4}$  are support vectors. According to Eqs. (7.25) and (7.26), we have

$$\begin{aligned}
w_1^* &= w_2^* = \frac{1}{2} \\
b^* &= -2
\end{aligned}$$

The separating hyperplane is

$$\frac{1}{2}x^{(1)} + \frac{1}{2}x^{(2)} - 2 = 0$$

The classification decision function is

$$f(x) = \text{sign}\left(\frac{1}{2}x^{(1)} + \frac{1}{2}x^{(2)} - 2\right)$$

For linearly separable problems, the above learning algorithm (hard margin maximization) of the linearly separable SVM is perfect. However, the linear separability of the training dataset is the ideal case. In practical problems, the training dataset is

often linearly inseparable, i.e., there is noise or peculiarities in the sample. At this time, there are more general learning algorithms.

## 7.2 Linear Support Vector Machine and Soft Margin Maximization

### 7.2.1 Linear Support Vector Machine

The SVM learning method for linearly separable problems does not work for linearly inseparable training data because the inequality constraints in the above methods do not all hold. How can it be extended to linearly inseparable problems? This requires modifying the hard margin to maximize the soft margin.

Suppose that a training dataset on the feature space is given

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where  $x_i \in \mathcal{X} = R^n$ ,  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ .  $x_i$  is the  $i$ -th feature vector, and  $y_i$  is the class label of  $x_i$ . Assume again that the training dataset is not linearly separable. Usually, there are some outliers in the training data. After removing these outliers, most of the remaining sample points sets are linearly separable.

Linear inseparability means that some sample points  $(x_i, y_i)$  cannot satisfy the constraint condition (7.14) that the function margin is greater than or equal to 1. For solving this problem, we can introduce a slack variable  $\xi_i \geq 0$  for each sample point  $(x_i, y_i)$  so that the function margin plus the slack variable is greater than or equal to 1. In this way, the constraint becomes

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

Also, for each slack variable  $\xi_i$ , a cost  $\xi_i$  is paid. The objective function changes from the original  $\frac{1}{2}\|w\|^2$  to

$$\frac{1}{2}\|w\|^2 + C \sum_{i=1}^N \xi_i \quad (7.31)$$

Here,  $C > 0$  is called the penalty parameter, which is generally determined by the application problem. When the  $C$  is large, the penalty for misclassification increases for large  $C$  values and decreases for small  $C$  values. Minimizing the objective function (7.31) has two levels: to make  $\frac{1}{2}\|w\|^2$  as small as possible, i.e., the margin is as large as possible, and to keep the number of misclassified points as small as possible, and  $C$  is the coefficient that reconciles the two.

With the above idea, the linear SVM learning problem when the training dataset is linearly inseparable can be considered in the same way as when the training dataset is linearly separable. Corresponding to the hard margin maximization, it is called the soft margin maximization.

The learning problem of the linear SVM in the linearly inseparable case becomes the following convex quadratic programming problem (the primal problem):

$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \quad (7.32)$$

$$\text{s.t.} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N \quad (7.33)$$

$$\xi_i \geq 0, \quad i = 1, 2, \dots, N \quad (7.34)$$

The primal problem (7.32)~(7.34) is a convex quadratic programming problem, and thus solutions to  $(w, b, \xi)$  exist. It can be proved that the solution of  $w$  is unique, but the solution of  $b$  may exist in an interval [1], instead of being unique.

Suppose the solution of the problem (7.32)~(7.34) is  $w^*, b^*$ , then we obtain the separating hyperplane  $w^* \cdot x + b^* = 0$  and the classification decision function  $f(x) = \text{sign}(w^* \cdot x + b^*)$ . Such a model is called a linear SVM when the training samples are linearly inseparable, or a linear SVM for short. Obviously, the linear SVM includes the linear SVM in the linearly separable case. Since the training dataset is often linearly inseparable in practice, the linear SVM has wider applicability.

The definition of linear SVM is given below.

**Definition 7.5 (Linear Support Vector Machine)** For a given linear inseparable training dataset, by solving the convex quadratic programming problem, i.e., the soft margin maximization problem (7.32)~(7.34), the separating hyperplane obtained is

$$w^* \cdot x + b^* = 0 \quad (7.35)$$

and the corresponding classification decision function is

$$f(x) = \text{sign}(w^* \cdot x + b^*) \quad (7.36)$$

And it is called a linear Support Vector Machine.

### 7.2.2 Dual Learning Algorithm

The dual problem of the primal problem (7.32)~(7.34) is

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^N \alpha_i \quad (7.37)$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (7.38)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N \quad (7.39)$$

The Lagrange function of the primal optimization problem (7.32)~(7.34) is

$$\begin{aligned} L(w, b, \xi, \alpha, \mu) = & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ & - \sum_{i=1}^N \alpha_i (y_i (w \cdot x_i + b) - 1 + \xi_i) - \sum_{i=1}^N \mu_i \xi_i \end{aligned} \quad (7.40)$$

where  $\alpha_i \geq 0, \mu_i \geq 0$ .

The dual problem is the min-max problem of the Lagrange function. First, find the minimum of  $L(w, b, \xi, \alpha, \mu)$  for  $w, b, \xi$  by

$$\nabla_w L(w, b, \xi, \alpha, \mu) = w - \sum_{i=1}^N \alpha_i y_i x_i = 0$$

$$\nabla_b L(w, b, \xi, \alpha, \mu) = - \sum_{i=1}^N \alpha_i y_i = 0$$

$$\nabla_{\xi_i} L(w, b, \xi, \alpha, \mu) = C - \alpha_i - \mu_i = 0$$

and get

$$w = \sum_{i=1}^N \alpha_i y_i x_i \quad (7.41)$$

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (7.42)$$

$$C - \alpha_i - \mu_i = 0 \quad (7.43)$$

Substituting Eqs. (7.41)~(7.43) into Eq. (7.40), we get

$$\min_{w, b, \xi} L(w, b, \xi, \alpha, \mu) = - \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_{i=1}^N \alpha_i$$

For  $\min_{w, b, \xi} L(w, b, \xi, \alpha, \mu)$  to find the maximum of  $\alpha$ , the dual problem is obtained:

$$\max_{\alpha} - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_{i=1}^N \alpha_i \quad (7.44)$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (7.45)$$

$$C - \alpha_i - \mu_i = 0 \quad (7.46)$$

$$\alpha_i \geq 0 \quad (7.47)$$

$$\mu_i \geq 0, \quad i = 1, 2, \dots, N \quad (7.48)$$

Transform the dual optimization problem (7.44)~(7.48): use the equality constraint (7.46) to eliminate  $\mu_i$ , leaving only the variable  $\alpha_i$ , and write the constraints (7.46)~(7.48) as

$$0 \leq \alpha_i \leq C \quad (7.49)$$

Then the maximization of the objective function is converted to the minimum, and the dual problem (7.37)~(7.39) is obtained.

The solution to the primal problem can be obtained by solving the dual problem, and then the separating hyperplane and decision function can be determined. For this reason, the relationship between the optimal solution of the primal problem and the optimal solution of the dual problem can be described in the form of a theorem.

**Theorem 7.3** Let  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$  be a solution of the dual problem (7.37)~(7.39). If there is a component of  $\alpha^*$  called  $\alpha_j^*$ ,  $0 < \alpha_j^* < C$ , then  $w^*$ ,  $b^*$ , the solution of the primal problem (7.32)~(7.34) can be obtained as follows:

$$w^* = \sum_{i=1}^N \alpha_i^* y_i x_i \quad (7.50)$$

$$b^* = y_j - \sum_{i=1}^N y_i \alpha_i^* (x_i \cdot x_j) \quad (7.51)$$

**Proof** The primal problem is a convex quadratic programming problem, and the solution satisfies the KKT condition, i.e.,

$$\nabla_w L(w^*, b^*, \xi^*, \alpha^*, \mu^*) = w^* - \sum_{i=1}^N \alpha_i^* y_i x_i = 0 \quad (7.52)$$

$$\begin{aligned} \nabla_b L(w^*, b^*, \xi^*, \alpha^*, \mu^*) &= -\sum_{i=1}^N \alpha_i^* y_i = 0 \\ \nabla_\xi L(w^*, b^*, \xi^*, \alpha^*, \mu^*) &= C - \alpha^* - \mu^* = 0 \\ \alpha_i^*(y_i(w^* \bullet x_i + b^*) - 1 + \xi_i^*) &= 0 \end{aligned} \quad (7.53)$$

$$\begin{aligned} \mu_i^* \xi_i^* &= 0 \\ y_i(w^* \bullet x_i + b^*) - 1 + \xi_i^* &\geq 0 \\ \xi_i^* &\geq 0 \\ \alpha_i^* &\geq 0 \\ \mu_i^* &\geq 0, \quad i = 1, 2, \dots, N \end{aligned} \quad (7.54)$$

It is easy to see Eq. (7.50) holds from Eq. (7.52). Equations (7.53) and (7.54) show that if there exists  $\alpha_j^*$ ,  $0 < \alpha_j^* < C$ , then we have  $y_i(w^* \bullet x_i + b^*) - 1 = 0$ , which yields (7.51).

From this theorem, the separating hyperplane can be written as

$$\sum_{i=1}^N \alpha_i^* y_i (x \bullet x_i) + b^* = 0 \quad (7.55)$$

The classification decision function can be written as

$$f(x) = \text{sign}\left(\sum_{i=1}^N \alpha_i^* y_i (x \bullet x_i) + b^*\right) \quad (7.56)$$

Equation (7.56) is the dual form of the linear SVM.

Based on the previous results, there is the following algorithm.

### Algorithm 7.3 (The Linear Support Vector Machine Learning Algorithm)

Input: training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \mathcal{X} = \mathbb{R}^n$ ,  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ .

Output: Separating hyperplane and classification decision function.

- (1) Select the penalty parameter  $C > 0$ , construct and solve the convex quadratic programming problem

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) - \sum_{i=1}^N \alpha_i$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N$$

Find the optimal solution  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$ .

$$(2) \text{ Calculate } w^* = \sum_{i=1}^N \alpha_i^* y_i x_i.$$

Choose  $\alpha_j^*$ , a component of  $\alpha^*$  to fit the condition  $0 < \alpha_j^* < C$ , and calculate

$$b^* = y_j - \sum_{i=1}^N y_i \alpha_i^* (x_i \bullet x_j)$$

(3) Find the separating hyperplane

$$w^* \bullet x + b^* = 0$$

and the classification decision function:

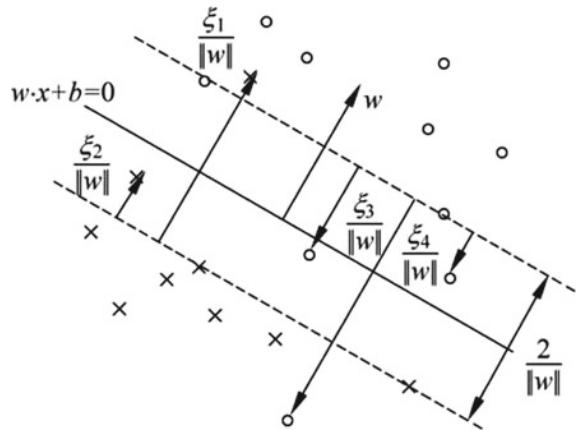
$$f(x) = \text{sign}(w^* \bullet x + b^*)$$

In step (2), for any  $\alpha_j^*$  satisfying the condition  $0 < \alpha_j^* < C$ ,  $b^*$  can be obtained according to Eq. (7.51). In theory, the solutions to the primal problem (7.32)~(7.34) may not be unique [2], but in practical applications, there is often only the case of algorithm description.

### 7.2.3 Support Vector

In the case of inseparable linearity, the instance  $x_i$  corresponding to the sample point  $(x_i, y_i)$  with  $\alpha_i^* > 0$  in the solution  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$  of the dual problem (7.37)~(7.39) is called the support vector (support vector of soft margin). As shown in Fig. 7.5, the support vector at this time is more complicated than the linearly separable case. In the figure, the separating hyperplane is represented by a solid line, the separating boundary is represented by a dotted line, the positive instance point is represented by “○”, and the negative instance point is represented by “×”. The distance  $\frac{\xi_i}{\|w\|}$  from the instance  $x_i$  to the separating boundary is also marked in the figure.

**Fig. 7.5** Support vector of soft margin



The support vector of the soft margin  $x_i$  is either on the margin boundary, between the margin boundary and the separating hyperplane, or on the misclassified side of the separating hyperplane. If  $\alpha_i^* < C$ , then  $\xi_i = 0$ , and the support vector  $x_i$  just falls on the margin boundary; if  $\alpha_i^* = C$ ,  $0 < \xi_i < 1$ , then the classification is correct, and  $x_i$  is between the margin boundary and the separating hyperplane; if  $\alpha_i^* = C$ ,  $\xi_i = 1$ , then  $x_i$  is on the separating hyperplane; if  $\alpha_i^* = C$ ,  $\xi_i > 1$ , then  $x_i$  lies on the misclassified side of the separating hyperplane.

### 7.2.4 Hinge Loss Function

For linear SVM learning, its model is separating hyperplane  $w^* \bullet x + b^* = 0$  and the decision function  $f(x) = \text{sign}(w^* \bullet x + b^*)$ . The learning strategy is soft margin maximization, and the learning algorithm is convex quadratic programming.

Another interpretation of linear SVM learning is to minimize the following objective function:

$$\sum_{i=1}^N [1 - y_i(w \bullet x_i + b)]_+ + \lambda \|w\|^2 \quad (7.57)$$

The first term of the objective function is empirical loss or empirical risk. The function

$$L(y(w \bullet x + b)) = [1 - y(w \bullet x + b)]_+ \quad (7.58)$$

is called hinge loss function. The subscript “+” denotes the following function with positive values.

$$[z]_+ \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases} \quad (7.59)$$

That is to say, when the sample points  $(x_i, y_i)$  are correctly classified and the function margin (certainty)  $y_i(w \bullet x_i + b)$  is greater than 1, the loss is 0. Otherwise, the loss is  $1 - y_i(w \bullet x_i + b)$ . Note that the instance point  $x_4$  in Fig. 7.5 is correctly classified, but the loss is not 0. The second term of the objective function is the  $L_2$  norm of  $w$  with coefficient  $\lambda$ , which is the regularization term.

**Theorem 7.4** The primal optimization problem of linear Support Vector Machine:

$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \quad (7.60)$$

$$\text{s.t.} \quad y_i(w \bullet x_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N \quad (7.61)$$

$$\xi_i \geq 0, \quad i = 1, 2, \dots, N \quad (7.62)$$

is equivalent to the optimization problem

$$\min_{w, b} \sum_{i=1}^N [1 - y_i(w \bullet x_i + b)]_+ + \lambda \|w\|^2 \quad (7.63)$$

*Proof* (7.63) can be written as the problem (7.60)~(7.62). Let

$$[1 - y_i(w \bullet x_i + b)]_+ = \xi_i \quad (7.64)$$

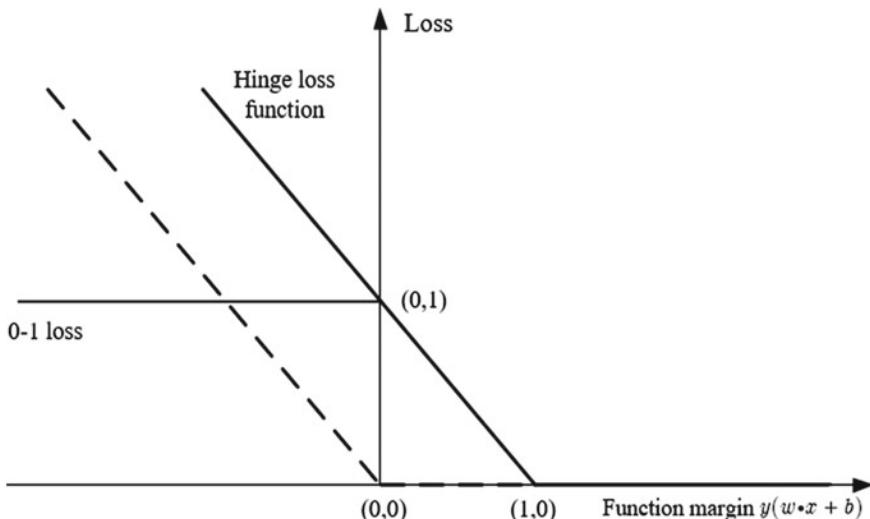
Then  $\xi_i \geq 0$  and Eq. (7.62) holds. From Eq. (7.64), we have  $y_i(w \bullet x_i + b) = 1 - \xi_i$  when  $1 - y_i(w \bullet x_i + b) > 0$ ; When  $1 - y_i(w \bullet x_i + b) \leq 0$ , we have  $\xi_i = 0$ ,  $y_i(w \bullet x_i + b) \geq 1 - \xi_i$ . Therefore, Eq. (7.61) holds. Then  $w$ ,  $b$ ,  $\xi_i$  satisfy the constraints (7.61)~(7.62). The optimization problem (7.63) can be written as

$$\min_{w, b} \sum_{i=1}^N \xi_i + \lambda \|w\|^2$$

If  $\lambda = \frac{1}{2C}$ , then

$$\min_{w, b} \frac{1}{C} \left( \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \right)$$

which is equivalent to Eq. (7.60).



**Fig. 7.6** Hinge loss function

On the contrary, the optimization problem (7.60)~(7.62) can be expressed as the problem (7.63).

The graph of hinge loss function is shown in Fig. 7.6. The horizontal axis is function margin  $y(w \cdot x + b)$ , and the vertical axis is the loss. Since the function is shaped like a hinge, it is called a hinge loss function.

Also shown in the figure is the 0–1loss function, which can be considered the real loss function of the two-class classification problem. The hinge loss function is the upper bound of the 0–1 loss function. Since the 0–1 loss function is not continuously differentiable, it is difficult to optimize the objective function composed of it directly. The linear SVM can be considered to optimize the objective function composed of the upper bound of the 0–1 loss function (hinge loss function). The upper bound loss function is also called the surrogate loss function.

The dotted line in Fig. 7.6 shows the loss function  $[-y_i(w \cdot x_i + b)]_+$  of the perceptron. At this time, the loss is 0 when the sample points  $(x_i, y_i)$  are correctly classified. Otherwise, the loss is  $-y_i(w \cdot x_i + b)$ . In contrast, the loss in the hinge loss function is 0 only when the classification is correct with sufficient certainty. In other words, the hinge loss function has higher requirements for learning.

## 7.3 Non-Linear Support Vector Machine and Kernel Functions

For solving linear classification problems, a linear classification support vector machine is a very effective method. However, sometimes the classification problem is non-linear, in which case a non-linear SVM can be used. This section describes the non-linear SVM, whose main feature is the use of kernel tricks. To this end, the kernel trick is first introduced. The kernel trick is not only applied to the Support Vector Machine but also to other machine learning problems.

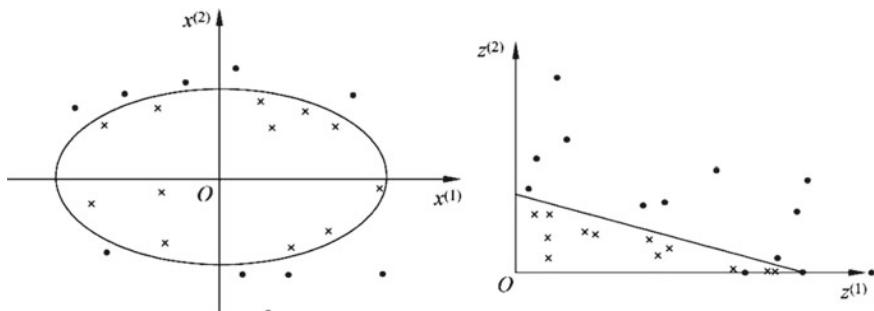
### 7.3.1 Kernel Trick

#### 7.3.1.1 Non-Linear Classification Problem

The non-linear classification problem is the problem that can be classified well only by using non-linear models. Let's look at an example: Fig. 7.7, left, shows a classification problem in which “.” represents positive instance points, and “ $\times$ ” represents a negative instance points. It can be seen from the figure that the positive and negative instances cannot be separated correctly by a straight line (the linear model), but they can be separated correctly by an elliptic curve (the non-linear model).

Generally speaking, for a given training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where the instance  $x_i$  belongs to the input space,  $x_i \in \mathcal{X} = \mathbb{R}^n$ , there are two classes of corresponding marks  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ . If a hypersurface in  $\mathbb{R}^n$  can be used to correctly separate positive and negative instances, then this problem is called a non-linear separable problem.

The non-linear problem is often not easy to solve, so it is desirable to solve this problem by solving linear classification problems. The approach taken is to perform a non-linear transformation, converting the non-linear problem into a linear problem,



**Fig. 7.7** Examples of the non-linear classification problem and kernel tricks

and then solve the original non-linear problem by solving the transformed linear problem. For the example shown in Fig. 7.7, the ellipse in the left figure is transformed into the straight line in the right figure, and the non-linear classification problem is transformed into a linear classification problem.

Suppose the original space is  $\mathcal{X} \subset R^2$ ,  $x = (x^{(1)}, x^{(2)})^T \in \mathcal{X}$ , and the new space is  $\mathcal{Z} \subset R^2$ ,  $z = (z^{(1)}, z^{(2)})^T \in \mathcal{Z}$ . Define the transformation (mapping) from the original space to new one as follows.

$$z = \phi(x) = \left( (x^{(1)})^2, (x^{(2)})^2 \right)^T$$

After the transformation  $z = \phi(x)$ , the original space  $\mathcal{X} \subset R^2$  is transformed into the new space  $\mathcal{Z} \subset R^2$ , and the points in the original space are transformed into those in the new space accordingly. The ellipse in the original space, i.e.,

$$w_1(x^{(1)})^2 + w_2(x^{(2)})^2 + b = 0$$

transforms into a straight line in the new space

$$w_1z^{(1)} + w_2z^{(2)} + b = 0$$

In the transformed new space, the straight line  $w_1z^{(1)} + w_2z^{(2)} + b = 0$  correctly separates the transformed positive and negative instance points. Thus, the non-linear separable problem in the original space becomes a linearly separable problem in the new space.

The above example illustrates that solving a non-linear classification problem by a linear classification method is a two-step process. First, use a transformation to map the data in the original space to the new space. Then the linear classification learning method is used to learn the classification model from the training data in the new space. The kernel trick is such an approach.

The basic idea of the kernel trick applied to the SVM is to correspond the input space (Euclidean space  $R^n$  or discrete set) to a feature space (Hilbert space  $\mathcal{H}$ ) by a non-linear transformation so that the hypersurface model in the input space  $R^n$  corresponds to the hyperplane model in the feature space  $\mathcal{H}$  (SVM). In this way, the learning task of the classification problem can be completed by solving the linear SVM in the feature space.

### 7.3.1.2 Definition of the Kernel Function

**Definition 7.6 (Kernel Function)** Let  $\mathcal{X}$  be the input space (a subset or discrete set of Euclidean space  $R^n$ ) and let  $\mathcal{H}$  be the feature space (Hilbert space). If there exists a mapping from  $\mathcal{X}$  to  $\mathcal{H}$ , i.e.,

$$\phi(x) : \mathcal{X} \rightarrow \mathcal{H} \quad (7.65)$$

such that for all  $x, z \in \mathcal{X}$ , the function  $K(x, z)$  satisfies the condition

$$K(x, z) = \phi(x) \cdot \phi(z) \quad (7.66)$$

Then  $K(x, z)$  is called the kernel function, and  $\phi(x)$  is the mapping function, where  $\phi(x) \cdot \phi(z)$  is the inner product of  $\phi(x)$  and  $\phi(z)$ .

The idea of the kernel trick is to define only the kernel function  $K(x, z)$  in learning and prediction without explicitly defining the mapping function  $\phi$ . Generally, it is easier to compute  $K(x, z)$  directly, while it is not easy to compute  $K(x, z)$  by  $\phi(x)$  and  $\phi(z)$ . Note that  $\phi$  is the mapping from the input space  $R^n$  to the feature space  $\mathcal{H}$ . The feature space  $\mathcal{H}$  is generally high-dimensional or even infinite. It can be seen that for a given kernel  $K(x, z)$ , the feature space  $\mathcal{H}$  and the mapping function  $\phi$  are not unique, and different feature spaces can be taken even in the same feature space.

Here is a simple example to illustrate the relationship between the kernel function and the mapping function.

**Example 7.3** Suppose the input space is  $R^2$  and the kernel function is  $K(x, z) = (x \cdot z)^2$ . Find the related feature space  $\mathcal{H}$  and the mapping  $\phi(x) : R^2 \rightarrow \mathcal{H}$ .

**Solution** Take the feature space  $\mathcal{H} \ominus = R^3$  and denote  $x = (x^{(1)}, x^{(2)})^T$ ,  $z = (z^{(1)}, z^{(2)})^T$ . Since

$$\begin{aligned} (x \cdot z)^2 &= (x^{(1)}z^{(1)} + x^{(2)}z^{(2)})^2 \\ &= (x^{(1)}z^{(1)})^2 + 2x^{(1)}z^{(1)}x^{(2)}z^{(2)} + (x^{(2)}z^{(2)})^2 \end{aligned}$$

we can take the mapping

$$\phi(x) = \left( (x^{(1)})^2, \sqrt{2}x^{(1)}x^{(2)}, (x^{(2)})^2 \right)^T$$

It is easy to verify that  $\phi(x) \cdot \phi(z) = (x \cdot z)^2 = K(x, z)$ .

We still take  $\mathcal{H} = R^3$  and

$$\phi(x) = \frac{1}{\sqrt{2}} \left( (x^{(1)})^2 - (x^{(2)})^2, 2x^{(1)}x^{(2)}, (x^{(1)})^2 + (x^{(2)})^2 \right)^T$$

Similarly we have  $\phi(x) \cdot \phi(z) = (x \cdot z)^2 = K(x, z)$ .

It is also possible to take  $\mathcal{H} \ominus = R^4$  and

$$\phi(x) = \left( (x^{(1)})^2, x^{(1)}x^{(2)}, x^{(1)}x^{(2)}, (x^{(2)})^2 \right)^T$$

### 7.3.1.3 The Application of Kernel Tricks in the Support Vector Machine

We noticed that in the dual problem of the linear SVM, both the objective function and the decision function (separating hyperplane) involve only the inner product between the input instance and the instance. The inner product  $x_i \cdot x_j$  in the objective function of the dual problem (7.37) can be replaced by the kernel function

$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ . At this time, the objective function of the dual problem becomes

$$W(\alpha) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i \quad (7.67)$$

Similarly, the inner product in the classification decision function can also be replaced by the kernel function, and the classification decision function becomes

$$\begin{aligned} f(x) &= \text{sign}\left(\sum_{i=1}^{N_S} a_i^* y_i \phi(x_i) \cdot \phi(x) + b^*\right) \\ &= \text{sign}\left(\sum_{i=1}^{N_S} a_i^* y_i K(x_i, x) + b^*\right) \end{aligned} \quad (7.68)$$

This is equivalent to transforming the original input space to a new feature space through the mapping function  $\phi$ , and transforming the inner product  $x_i \cdot x_j$  in the input space to the inner product  $\phi(x_i) \cdot \phi(x_j)$  in the feature space. The linear SVM is then learned from the training samples in the new feature space. When the mapping function is a non-linear function, the learned SVM with a kernel function is a non-linear classification model.

That is, under the condition that the Kernel function  $K(x, z)$  is given, the SVM for the non-linear classification problem can be solved by the method for solving the linear classification problem. The learning carries on implicitly in the feature space without explicitly defining the feature space and mapping functions. Such a trick, known as the kernel trick, is a technique that cleverly solves non-linear problems using linear classification learning methods and kernel functions. In practical applications, the selection of kernel functions often depends on domain knowledge, and the validity of kernel function selection needs to be verified by experiments.

### 7.3.2 Positive Definite Kernel

Given the mapping function  $\phi$ , the kernel function  $K(x, z)$  can be obtained by the inner product of  $\phi(x)$  and  $\phi(z)$ . Is it possible to determine whether a given function

$K(x, z)$  is a kernel function without constructing a mapping  $\phi(x)$ ? In other words, what conditions does the function  $K(x, z)$  satisfy to be a kernel function?

This section describes the necessary and sufficient conditions of a positive definite kernel. Generally speaking, the kernel function is a positive definite kernel function. To prove this theorem, we first introduce the relevant preparatory knowledge.

Suppose  $K(x, z)$  is a symmetric function defined on  $\mathcal{X} \times \mathcal{X}$ , and the Gram matrix of  $K(x, z)$  concerning  $x_1, x_2, \dots, x_m$  is semi-positive definite for any  $x_1, x_2, \dots, x_m \in \mathcal{X}$ . A Hilbert space can be constructed based on the function  $K(x, z)$ . The steps are as follows: firstly, define the mapping  $\phi$  and construct the vector space  $\mathcal{S}$ ; secondly, define the inner product on  $\mathcal{S}$  to construct the inner product space; and finally, complete  $\mathcal{S}$  to construct the Hilbert space.

### 7.3.2.1 Define the Mapping and Form the Vector Space $\mathcal{S}$

First, define the mapping

$$\phi : x \rightarrow K(\bullet, x) \quad (7.69)$$

According to this mapping, for any  $x_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, i = 1, 2, \dots, m$ , define the linear combination

$$f(\bullet) = \sum_{i=1}^m \alpha_i K(\bullet, x_i) \quad (7.70)$$

Consider a set  $\mathcal{S}$  consisting of linear combinations as elements. Since the set  $\mathcal{S}$  is closed for addition and number multiplication operations,  $\mathcal{S}$  constitutes a vector space.

### 7.3.2.2 Define the Inner Product on $\mathcal{S}$ to Make it an Inner Product Space

Define an operation  $*$  on  $\mathcal{S}$ : for any  $f, g \in \mathcal{S}$ ,

$$f(\bullet) = \sum_{i=1}^m \alpha_i K(\bullet, x_i) \quad (7.71)$$

$$g(\bullet) = \sum_{j=1}^l \beta_j K(\bullet, z_j) \quad (7.72)$$

Define operation  $**$

$$f * g = \sum_{i=1}^m \sum_{j=1}^l \alpha_i \beta_j K(x_i, z_j) \quad (7.73)$$

Prove that the operation  $*$  is the inner product of the space  $\mathcal{S}$ . To this end, it is necessary to prove that:

(1)

$$(cf) * g = c(f * g), c \in \mathbf{R} \quad (7.74)$$

(2)

$$(f + g) * h = f * h + g * h, h \in \mathcal{S} \quad (7.75)$$

(3)

$$f * g = g * f \quad (7.76)$$

(4)

$$f * f \geq 0 \quad (7.77)$$

$$f * f = 0 \Leftrightarrow f = 0 \quad (7.78)$$

The proof of (1), (2) and (3) can be easily obtained from Eqs. (7.70–7.73) and the symmetry of  $K(x, z)$ . Now prove the Eq. (7.77) of (4). According to Eq. (7.70) and Eq. (7.73), it can be concluded that:

$$f * f = \sum_{i,j=1}^m \alpha_i \alpha_j K(x_i, x_j)$$

From the semi positive definiteness of the Gram matrix, we know that the right-hand side of the above equation is not negative, that is,  $f * f \geq 0$ .

Then prove the Eq. (7.78) of (4). The sufficiency is obvious. To prove the necessity, we first prove the inequality:

$$|f * g|^2 \leq (f * f)(g * g) \quad (7.79)$$

Let  $f, g \in \mathcal{S}$  and  $\lambda \in \mathbf{R}$ , then  $f + \lambda g \in \mathcal{S}$ . Thus,

$$\begin{aligned} (f + \lambda g) * (f + \lambda g) &\geq 0 \\ f * f + 2\lambda(f * g) + \lambda^2(g * g) &\geq 0 \end{aligned}$$

Its left end is the quadratic trinomial of  $\lambda$ , which is non-negative, and its discriminant is less than or equal to 0, i.e.,

$$(f * g)^2 - (f * f)(g * g) \leq 0$$

So the Eq. (7.79) is proved. Now prove that if  $f * f = 0$ , then  $f = 0$ . In fact, if

$$f(\bullet) = \sum_{i=1}^m \alpha_i K(\bullet, x_i)$$

Then by the definition Eq. (7.73) of operation  $*$ , for any  $x \in \mathcal{X}$ , there is

$$K(\bullet, x) * f = \sum_{i=1}^m \alpha_i K(x, x_i) = f(x)$$

Thus,

$$|f(x)|^2 = |K(\bullet, x) * f|^2 \quad (7.80)$$

From Eqs. (7.79) and (7.77) we have

$$\begin{aligned} |K(\bullet, x) * f|^2 &\leq (K(\bullet, x) * K(\bullet, x))(f * f) \\ &= K(x, x)(f * f) \end{aligned}$$

From Eq. (7.80) we have

$$|f(x)|^2 \leq K(x, x)(f * f)$$

This equation shows that when  $f * f = 0$ , we have  $|f(x)| = 0$  for any  $x$ .

So far, it proves that  $*$  is the inner product of vector space  $\mathcal{S}$ . The vector space that gives the inner product is an inner product space. Therefore  $\mathcal{S}$  is an inner product space. Since  $*$  is the inner product operation of  $\mathcal{S}$ , it is still expressed by  $\bullet$ , i.e., if

$$f(\bullet) = \sum_{i=1}^m \alpha_i K(\bullet, x_i), \quad g(\bullet) = \sum_{j=1}^l \beta_j K(\bullet, z_j)$$

then

$$f \bullet g = \sum_{i=1}^m \sum_{j=1}^l \alpha_i \beta_j K(x_i, z_j) \quad (7.81)$$

### 7.3.2.3 Complete the Inner Product Space $\mathcal{S}$ into a Hilbert Space

Now we complete the inner product space  $\mathcal{S}$ . The inner product defined by Eq. (7.81) yields the norm

$$\|f\| = \sqrt{f \bullet f} \quad (7.82)$$

Thus,  $\mathcal{S}$  is a normed vector space. According to the theory of Functional Analysis, the incomplete normed vector space  $\mathcal{S}$  can be completed to obtain the normed vector space  $\mathcal{H}$ . An inner product space is a Hilbert space when it is a complete normed vector space. Therefore, we get the Hilbert space  $\mathcal{H}$ .

This Hilbert space  $\mathcal{H}$  is called the reproducing kernel Hilbert space (RKHS). This is due to the reproducing nature of the kernel  $K$ , i.e., when it satisfies

$$K(\bullet, x) \bullet f = f(x) \quad (7.83)$$

and

$$K(\bullet, x) \bullet K(\bullet, z) = K(x, z) \quad (7.84)$$

it is called the reproducing kernel.

### 7.3.2.4 Necessary and Sufficient Conditions for Positive Definite Kernels

**Theorem 7.5 (Necessary and Sufficient Conditions for Positive Definite Kernels)** Let  $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  be a symmetric function, then a sufficient condition for  $K(x, z)$  to be a positive definite kernel function is that for any  $x_i \in \mathcal{X}$ ,  $i = 1, 2, \dots, m$ , the Gram matrix corresponding to  $K(x, z)$

$$K = [K(x_i, x_j)]_{m \times m} \quad (7.85)$$

is a semi-positive definite matrix.

**Proof** Necessity. Since  $K(x, z)$  is a positive definite kernel on  $\mathcal{X} \times \mathcal{X}$ , there exists a mapping  $\phi$  from  $\mathcal{X}$  to the Hilbert space  $\mathcal{H}$  such that

$$K(x, z) = \phi(x) \bullet \phi(z)$$

Thus, the Gram matrix of  $K(x, z)$  with respect to  $x_1, x_2, \dots, x_m$  is constructed for any  $x_1, x_2, \dots, x_m$

$$[K_{ij}]_{m \times m} = [K(x_i, x_j)]_{m \times m}$$

For any  $c_1, c_2, \dots, c_m \in R$ , there is

$$\begin{aligned} \sum_{i,j=1}^m c_i c_j K(x_i, x_j) &= \sum_{i,j=1}^m c_i c_j (\phi(x_i) \bullet \phi(x_j)) \\ &= \left( \sum_i c_i \phi(x_i) \right) \bullet \left( \sum_j c_j \phi(x_j) \right) \\ &= \left\| \sum_i c_i \phi(x_i) \right\|^2 \geq 0 \end{aligned}$$

showing that the Gram matrix of  $K(x, z)$  with respect to  $x_1, x_2, \dots, x_m$  is semi-positive definite.

Sufficiency. It is known that the Gram matrix of the symmetric function  $K(x, z)$  with respect to  $x_1, x_2, \dots, x_m$  is semi-positive definite for any  $x_1, x_2, \dots, x_m \in \mathcal{X}$ . Based on the previous results, for a given  $K(x, z)$ , a mapping from  $\mathcal{X}$  to a certain Hilbert space  $\mathcal{H}$  can be constructed as follows:

$$\phi : x \rightarrow K(\bullet, x) \quad (7.86)$$

From Eq. (7.83), it can be seen that

$$K(\bullet, x) \bullet f = f(x)$$

And

$$K(\bullet, x) \bullet K(\bullet, z) = K(x, z)$$

From Eq. (7.86), we get

$$K(x, z) = \phi(x) \bullet \phi(z)$$

showing that  $K(x, z)$  is a kernel function on  $\mathcal{X} \times \mathcal{X}$ .

The theorem gives sufficient conditions for a positive definite kernel and therefore can be used as an alternative definition of a positive definite kernel, i.e., a kernel function.

**Theorem 7.7 (Equivalent Definition of the Positive Definite Kernel)** Set  $K(x, z)$  be a symmetric function defined on  $\mathcal{X} \times \mathcal{X}$ . If for any  $x_i \in \mathcal{X}, i = 1, 2, \dots, m$ , the Gram matrix corresponding to  $K(x, z)$ , i.e.,

$$K = [K(x_i, x_j)]_{m \times m} \quad (7.87)$$

is a semi-positive definite matrix, then  $K(x, z)$  is called a positive definite kernel.

This definition is helpful when constructing kernel functions. However, for a specific function  $K(x, z)$ , it is not easy to check whether it is a positive definite kernel function because it is required to verify that the Gram matrix corresponding to  $K$  is semi-positive definite for any finite set of inputs  $\{x_1, x_2, \dots, x_m\}$ . In practical problems, existing kernel functions are often applied. In addition, the Mercer kernel can be obtained from Mercer's theorem [1], and the positive definite kernel is more general than the Mercer kernel. Some common kernel functions are described below.

### 7.3.3 Commonly Used Kernel Functions

#### 7.3.3.1 Polynomial Kernel Function

$$K(x, z) = (x \bullet z + 1)^p \quad (7.88)$$

The corresponding SVM is a polynomial classifier of  $p$  times. In this case, the classification decision function becomes

$$f(x) = \text{sign}\left(\sum_{i=1}^{N_s} a_i * y_i (x_i \bullet x + 1)^p + b^*\right) \quad (7.89)$$

#### 7.3.3.2 Gaussian Kernel Function

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right) \quad (7.90)$$

The corresponding SVM is a Gaussian radial basis function classifier. In this case, the classification decision function becomes

$$f(x) = \text{sign}\left(\sum_{i=1}^{N_s} a_i * y_i \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right) + b^*\right) \quad (7.91)$$

#### 7.3.3.3 String Kernel Function

Kernel functions can be defined not only on Euclidean spaces but also on discrete data sets. For example, a string kernel is a kernel function defined on a collection of strings. String kernel functions have applications in text classification, information retrieval, bioinformatics, etc.

Consider a finite character table  $\Sigma$ . The string  $s$  is a sequence of finite characters taken from  $\Sigma$ , including the empty string. The length of the string  $s$  is denoted by

$|s|$ , and its elements are denoted as  $s(1) s(2) \dots s(|s|)$ . The concatenation of two strings,  $s$  and  $t$ , is denoted as  $st$ . The set of all strings of length  $n$  is denoted as  $\sum^n$ , and the set of all strings is denoted as  $\sum^* = \bigcup_{n=0}^{\infty} \sum^n$ .

Consider a substring  $u$  of the string  $s$ . Given a sequence of indicators  $i = (i_1, i_2, \dots, i_{|u|})$ ,  $1 \leq i_1 < i_2 < \dots < i_{|u|} \leq |s|$ , the substring of  $s$  is defined as  $u = s(i) = s(i_1)s(i_2)\dots s(i_{|u|})$ . Its length is denoted as  $l(i) = i_{|u|} - i_1 + 1$ . If  $i$  is continuous, then  $l(i) = |u|$ ; otherwise,  $l(i) > |u|$ .

Suppose  $\mathcal{S}$  is a set of strings of length greater than or equal to  $n$ , and  $s$  is an element of  $\mathcal{S}$ . Now build a mapping  $\phi_n(s)$  from the set of strings  $\mathcal{S}$  to the feature space  $\mathcal{H}_n = \mathcal{R}^{\sum^n}$ .  $\mathcal{R}^{\sum^n}$  denotes the space of real numbers defined on  $\sum^n$ , each dimension of which corresponds to a string  $u \in \sum^n$ . The mapping  $\phi_n(s)$  corresponds the string  $s$  to a vector of the space  $\mathcal{R}^{\sum^n}$ , whose values in  $u$  dimensions are

$$[\phi_n(s)]_u = \sum_{i : s(i)=u} \lambda^{l(i)} \quad (7.92)$$

Here,  $0 < \lambda \leq 1$  is a decay parameter,  $l(i)$  denotes the length of string  $i$ , and the summation is performed over all substrings in  $s$  that are identical to  $u$ .

For example, suppose  $\Sigma$  is the set of English characters,  $n$  is 3, and  $\mathcal{S}$  is the set of strings with lengths greater than or equal to 3. Consider the mapping from the character set  $\mathcal{S}$  to the feature space  $H_3$ . One dimension of  $H_3$  corresponds to the string asd. In this case, the values of the strings “Nasdaq” and “lassdas” in this dimension are  $[\phi_3(\text{Nasdaq})]_{\text{asd}} = \lambda^3$  and  $[\phi_3(\text{lass das})]_{\text{asd}} = 2\lambda^5$  ( $\square$  is a space). In the first string, asd is a continuous substring. In the 2nd string, asd is a discontinuous substring of length 5, occurring 2 times.

The string kernel functions on strings  $s$  and  $t$  are based on the inner product of the mapping  $\phi_n$  in the feature space:

$$\begin{aligned} k_n(s, t) &= \sum_{u \in \sum^n} [\phi_n(s)]_u [\phi_n(t)]_u \\ &= \sum_{u \in \sum^n} \sum_{(i, j) : s(i)=t(j)=u} \lambda^{l(i)} \lambda^{l(j)} \end{aligned} \quad (7.93)$$

The string kernel function  $k_n(s, t)$  gives the cosine similarity of the feature vector consisting of all substrings of length equal to  $n$  in strings  $s$  and  $t$ . Intuitively, the more identical substrings two strings have the more similar they are, and the larger the value of the string kernel function. The string kernel function can be computed quickly by dynamic programming.

### 7.3.4 Nonlinear Support Vector Classifier

As mentioned above, using the kernel trick, the learning method of linear classification can be applied to non-linear classification problems. Extending a linear SVM to a non-linear SVM simply requires replacing the dual inner product in the linear SVM with a kernel function.

**Definition 7.8 (Non-Linear Support Vector Machine)** The classification decision function obtained from the non-linear classification training set and learned by kernel function and soft margin maximization, or convex quadratic programming (7.95)~(7.97):

$$f(x) = \text{sign} \left( \sum_{i=1}^N \alpha_i^* y_i K(x, x_i) + b^* \right) \quad (7.94)$$

is called the non-linear Support Vector Machine, and  $K(x, x_i)$  is a positive definite kernel function.

The following describes the non-linear SVM learning algorithm.

#### Algorithm 7.4 (Non-Linear Support Vector Machine Learning Algorithm)

Input: the training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in X = \mathbb{R}^n$ ,  $y_i \in Y = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ ;

Output: the classification decision function.

- (1) Select the proper kernel function  $K(x, z)$  and the proper parameter  $C$ , construct and solve an optimization problem

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i \quad (7.95)$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (7.96)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N \quad (7.97)$$

- The optimal solution is  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$ .
- (2) Select a positive component  $\alpha^*$ ,  $0 < \alpha_j^* < C$ , and compute

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i K(x_i, x_j)$$

(3) Construct the decision function:

$$f(x) = \text{sign} \left( \sum_{i=1}^N \alpha_i^* y_i K(x, x_i) + b^* \right)$$

When  $K(x, z)$  is a positive definite kernel function, the problem (7.95)~(7.97) is a convex quadratic programming problem with an existing solution.

## 7.4 Sequential Minimal Optimization Algorithm

This section discusses the implementation problem of SVM learning. As we know, the learning problem of SVM can be formalized as a convex quadratic programming problem. Such a convex quadratic programming problem has a globally optimal solution, and there are many optimization algorithms available for solving the problem. However, when the training sample size is large, these algorithms often become so inefficient that they can't be used. Therefore, realizing SVM learning efficiently becomes an important issue. At present, many fast algorithms have been proposed. This section describes the sequential minimal optimization (SMO) algorithm proposed by Platt in 1998.

The SMO algorithm is to solve the dual problem of convex quadratic programming as follows:

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i \quad (7.98)$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (7.99)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N \quad (7.100)$$

Here, the variables is Lagrange multipliers, and one single variable  $\alpha_i$  corresponds to one sample point  $(x_i, x_j)$ . The total of variables equals the training sample size  $N$ .

The SMO algorithm is a heuristic algorithm based on the idea that the solution to the optimization problem is obtained if the solutions of all variables satisfy the Karush–Kuhn–Tucker (KKT) conditions of this optimization problem. It is because KKT conditions are necessary and sufficient conditions for this optimization problem. Otherwise, choose two variables, fix other variables, and construct a quadratic programming problem for these two variables. The solution of the quadratic programming problem concerning these two variables should be closer to the solution

of the original quadratic programming problem because it will make the objective function value of the original quadratic programming problem smaller. The important thing is that the subproblem can then be solved by analytical methods, which can greatly improve the computational speed of the entire algorithm. The subproblem has two variables, one is the one that violates the KKT condition the most, and the other is determined automatically by the constraints. In this way, the SMO algorithm continuously decomposes the original problem into subproblems and solves them to solve the original problem.

Note that only one of the two variables of the subproblem is a free variable. Assume that  $\alpha_1, \alpha_2$  are two variables and  $\alpha_3, \alpha_4, \dots, \alpha_N$  are fixed, then we know from the equality constraint (7.99) that

$$\alpha_1 = -y_1 \sum_{i=2}^N \alpha_i y_i$$

If  $\alpha_2$  is determined, then  $\alpha_1$  is also determined. So the two variables are updated simultaneously in the subproblem.

The entire SMO algorithm consists of two parts: the analytic method to solve two-variable quadratic programming and the heuristic method for selecting variables.

#### 7.4.1 The Method of Solving Two-Variable Quadratic Programming

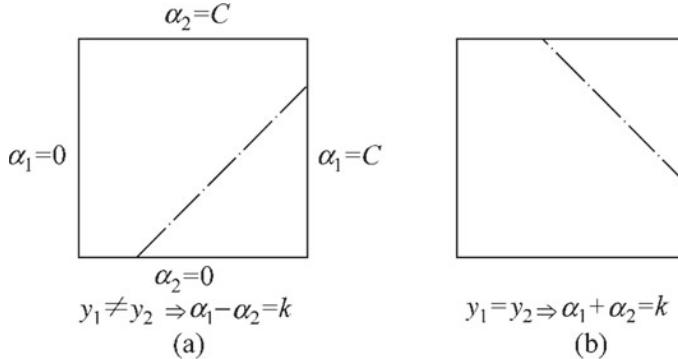
Without loss of generality, assume that the two variables selected are  $\alpha_1, \alpha_2$ , and other variables  $\alpha_i (i = 3, 4, \dots, N)$  are fixed. Then the subproblem of the SMO optimization problem (7.98)~(7.100) can be written as:

$$\begin{aligned} \min_{\alpha_1, \alpha_2} W(\alpha_1, \alpha_2) &= \frac{1}{2} K_{11} \alpha_1^2 + \frac{1}{2} K_{22} \alpha_2^2 + y_1 y_2 K_{12} \alpha_1 \alpha_2 \\ &\quad - (\alpha_1 + \alpha_2) \\ &\quad + y_1 \alpha_1 \sum_{i=3}^N y_i \alpha_i K_{i1} + y_2 \alpha_2 \sum_{i=3}^N y_i \alpha_i K_{i2} \end{aligned} \quad (7.101)$$

s.t.

$$\alpha_1 y_1 + \alpha_2 y_2 = - \sum_{i=3}^N y_i \alpha_i = \varsigma \quad (7.102)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2 \quad (7.103)$$



**Fig. 7.8** The two-variable optimization problem

where  $K_{ij} = K(x_i, x_j)$ ,  $i, j = 1, 2, \dots, N$ ,  $\varsigma$  is a constant, and the constant terms without  $\alpha_1, \alpha_2$  are omitted from the object function (7.101).

To solve the quadratic programming problem (7.101)~(7.103) with two variables, the constraint conditions are first analyzed and then minimized under the constraints.

Since there are only two variables ( $\alpha_1, \alpha_2$ ), the constraints could be indicated graphically in the two-dimensional space (as shown in Fig. 7.8).

The inequality constraint (7.103) makes  $(\alpha_1, \alpha_2)$  be in the box  $[0, C] \times [0, C]$ , and the equality constraint (7.102) makes  $(\alpha_1, \alpha_2)$  be on a line parallel to the diagonal of the box  $[0, C] \times [0, C]$ . Thus we should find the optimal value of the object function on a line segment parallel to the diagonal, which makes the two-variable optimization problem a univariate optimization problem. Then it could be considered as the optimization problem of variable  $\alpha_2$ .

Assume that the initial feasible solution of problems (7.101)~(7.103) are  $\alpha_1^{\text{old}}, \alpha_2^{\text{old}}$ , and the optimal solutions are  $\alpha_1^{\text{new}}, \alpha_2^{\text{new}}$ . Suppose that the optimal solution of  $\alpha_2$  when it is uncut along the constraint direction is  $\alpha_1^{\text{new}, \text{unc}}$ .

Since  $\alpha_2^{\text{new}}$  needs to satisfy the inequality constraint (7.103), the value range of the optimal value  $\alpha_2^{\text{new}}$  must satisfy the condition:

$$L \leq \alpha_2^{\text{new}} \leq H$$

$L$  and  $H$  are the bounds of the endpoints of the diagonal where  $\alpha_2^{\text{new}}$  is located. If  $y_1 \neq y_2$  (as shown in Fig. 7.8a), then

$$L = \max(0, \alpha_2^{\text{old}} - \alpha_1^{\text{old}}), \quad H = \min(C, C + \alpha_2^{\text{old}} - \alpha_1^{\text{old}})$$

If  $y_1 = y_2$  (as shown in Fig. 7.8b), then

$$L = \max(0, \alpha_2^{\text{old}} + \alpha_1^{\text{old}} - C), \quad H = \min(C, \alpha_2^{\text{old}} + \alpha_1^{\text{old}})$$

In the following, we first find the optimal solution  $\alpha_2^{\text{new, unc}}$  for  $\alpha_2$  along the constraint direction when it is uncut, i.e., without considering the inequality constraint (7.103). Then we obtain the solution  $\alpha_2^{\text{new}}$  for  $\alpha_2$  after editing. We describe this result in terms of a theorem. For the sake of simplicity, let

$$g(x) = \sum_{i=1}^N \alpha_i y_i K(x_i, x) + b \quad (7.104)$$

Let

$$E_i = g(x_i) - y_i = \left( \sum_{j=1}^N \alpha_j y_j K(x_j, x_i) + b \right) - y_i, \quad i = 1, 2 \quad (7.105)$$

When  $i = 1, 2$ ,  $E_i$  is the difference between the predicted value of the function  $g(x)$  for input  $x_i$  and the true output  $y_i$ .

**Theorem 7.6** Along the constraint direction, the solution of the optimization problem (7.101)~(7.103) without editing is

$$\alpha_2^{\text{new, unc}} = \alpha_2^{\text{old}} + \frac{y_2(E_1 - E_2)}{\eta} \quad (7.106)$$

where

$$\eta = K_{11} + K_{22} - 2K_{12} = \|\Phi(x_1) - \Phi(x_2)\|^2 \quad (7.107)$$

$\Phi(x)$  is the mapping from the input space to the feature space.  $E_i$ ,  $i = 1, 2$ , is given by Eq. (7.105).

The solution of  $\alpha_2$  after editing is

$$\alpha_2^{\text{new}} = \begin{cases} H, & \alpha_2^{\text{new, unc}} > H \\ \alpha_2^{\text{new, unc}}, & L \leq \alpha_2^{\text{new, unc}} \leq H \\ L, & \alpha_2^{\text{new, unc}} < L \end{cases} \quad (7.108)$$

From  $\alpha_2^{\text{new}}$ , we find that  $\alpha_1^{\text{new}}$  is

$$\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + y_1 y_2 (\alpha_2^{\text{old}} - \alpha_2^{\text{new}}) \quad (7.109)$$

**Proof** Introduce the notation

$$v_i = \sum_{j=3}^N \alpha_j y_j K(x_i, x_j) = g(x_i) - \sum_{j=1}^2 \alpha_j y_j K(x_i, x_j) - b, \quad i = 1, 2$$

The objective function can be written as

$$\begin{aligned} W(\alpha_1, \alpha_2) = & \frac{1}{2}K_{11}\alpha_1^2 + \frac{1}{2}K_{22}\alpha_2^2 + y_1y_2K_{12}\alpha_1\alpha_2 \\ & - (\alpha_1 + \alpha_2) + y_1v_1\alpha_1 + y_2v_2\alpha_2 \end{aligned} \quad (7.110)$$

From  $\alpha_1y_1 = \varsigma - \alpha_2y_2$  and  $y_i^2 = 1$ ,  $\alpha_1$  can be expressed as

$$\alpha_1 = (\varsigma - y_2\alpha_2)y_1$$

Substituting  $\alpha_1$  into Eq. (7.110), we obtain a function only with regard to  $\alpha_2$  as the objective function:

$$\begin{aligned} W(\alpha_2) = & \frac{1}{2}K_{11}(\varsigma - \alpha_2y_2)^2 + \frac{1}{2}K_{22}\alpha_2^2 + y_2K_{12}(\varsigma - \alpha_2y_2)\alpha_2 \\ & - (\varsigma - \alpha_2y_2)y_1 - \alpha_2 + v_1(\varsigma - \alpha_2y_2) + y_2v_2\alpha_2 \end{aligned}$$

Find the derivative of  $\alpha_2$ :

$$\begin{aligned} \frac{\partial W}{\partial \alpha_2} = & K_{11}\alpha_2 + K_{22}\alpha_2 - 2K_{12}\alpha_2 \\ & - K_{11}\varsigma y_2 + K_{12}\varsigma y_2 + y_1y_2 - 1 - v_1y_2 + y_2v_2 \end{aligned}$$

Let it be 0 to get

$$\begin{aligned} (K_{11} + K_{22} - 2K_{12})\alpha_2 &= y_2(y_2 - y_1 + \varsigma K_{11} - \varsigma K_{12} + v_1 - v_2) \\ &= y_2 \left[ \begin{array}{l} y_2 - y_1 + \varsigma K_{11} - \varsigma K_{12} \\ + \left( g(x_1) - \sum_{j=1}^2 y_j \alpha_j K_{1j} - b \right) \\ - \left( g(x_2) - \sum_{j=1}^2 y_j \alpha_j K_{2j} - b \right) \end{array} \right] \end{aligned}$$

Substitute  $\varsigma = \alpha_1^{\text{old}}y_1 + \alpha_2^{\text{old}}y_2$  into the above equation to get

$$\begin{aligned} (K_{11} + K_{22} - 2K_{12})\alpha_2^{\text{new, unc}} &= y_2 \left( \begin{array}{l} (K_{11} + K_{22} - 2K_{12})\alpha_2^{\text{old}}y_2 \\ + y_2 - y_1 + g(x_1) \\ - g(x_2) \end{array} \right) \\ &= (K_{11} + K_{22} - 2K_{12})\alpha_2^{\text{old}} \\ &\quad + y_2(E_1 - E_2) \end{aligned}$$

Substitute  $\eta = K_{11} + K_{22} - 2K_{12}$  into the above equation to get

$$\alpha_2^{\text{new, unc}} = \alpha_2^{\text{old}} + \frac{y_2(E_1 - E_2)}{\eta}$$

To satisfy the inequality constraint it must be restricted to the interval  $[L, H]$ , which gives the expression for  $\alpha_2^{\text{new}}$  (7.108). From the equation constraint (7.102), the expression (7.109) for  $\alpha_1^{\text{new}}$  is obtained. Thus, the solutions  $(\alpha_1^{\text{new}}, \alpha_2^{\text{new}})$  of the optimization problem (7.101)~(7.103) are obtained.

## 7.4.2 Selection Methods of Variables

The SMO algorithm selects two variables to optimize in each subproblem, at least one of which is a violation of the KKT condition.

### 7.4.2.1 Selection of the First Variable

SMO calls the process of selecting the first variable the outer loop. The outer loop selects the sample point in the training sample that violates the KKT condition most seriously and uses its corresponding variable as the first variable. Specifically, the training sample points  $(x_i, y_i)$  are checked to see if they satisfy the KKT condition, i.e.

$$\alpha_i = 0 \Leftrightarrow y_i g(x_i) \geq 1 \quad (7.111)$$

$$0 < \alpha_i < C \Leftrightarrow y_i g(x_i) = 1 \quad (7.112)$$

$$\alpha_i = C \Leftrightarrow y_i g(x_i) \leq 1 \quad (7.113)$$

$$\text{where } g(x_i) = \sum_{j=1}^N \alpha_j y_j K(x_i, x_j) + b.$$

The test performs in the range of  $\varepsilon$ . In the test process, the outer loop first iterates through all sample points that satisfy the condition  $0 < \alpha_i < C$ , i.e., the support vector points on the margin boundary, and tests whether they satisfy the KKT condition. If all these sample points meet the KKT condition, the entire training set is traversed to check whether they satisfy the KKT condition.

### 7.4.2.2 Selection of the Second Variable

SMO calls the process of selecting the second variable the inner loop. Suppose you have already found the first variable  $\alpha_1$  in the outer loop and now you want to find the second variable  $\alpha_2$  in the inner loop. The second variable is chosen based on the desire to cause a sufficient change in  $\alpha_2$ .

From Eqs. (7.106) and (7.108),  $\alpha_2^{\text{new}}$  is dependent on  $|E_1 - E_2|$ . For accelerating the computation, a simple approach is to choose  $\alpha_2$  so that its corresponding  $|E_1 - E_2|$  is maximum. Since  $\alpha_1$  is determined,  $E_1$  is also determined. If  $E_1$  is positive, the smallest  $E_i$  is selected as  $E_2$ ; if  $E_1$  is negative, the largest  $E_i$  is chosen as  $E_2$ . All  $E_i$  values are saved in a list to save computation time.

In special cases, if the  $\alpha_2$  chosen by the inner loop with the above method does not result in a sufficient descent in the objective function, the following heuristic rule is used to continue the selection of  $\alpha_2$ . Traverse the support vector points on the margin boundary are traversed and try their corresponding variables as  $\alpha_2$  in turn until the objective function drops enough. If no suitable  $\alpha_2$  is found, the training dataset is traversed; if still no suitable  $\alpha_2$  is found, discard the first  $\alpha_1$  and seek another  $\alpha_1$  through the outer loop.

### 7.4.2.3 Calculate the Threshold $b$ and the Difference $E_i$

After each optimization of the two variables, the threshold  $b$  must be recalculated. When  $0 < \alpha_1^{\text{new}} < C$ , it follows from the KKT condition (7.112) that:

$$\sum_{i=1}^N \alpha_i y_i K_{i1} + b = y_1$$

Then,

$$b_1^{\text{new}} = y_1 - \sum_{i=3}^N \alpha_i y_i K_{i1} - \alpha_1^{\text{new}} y_1 K_{11} - \alpha_2^{\text{new}} y_2 K_{21} \quad (7.114)$$

From the definition Eq. (7.105) of  $E_1$ , we have

$$E_1 = \sum_{i=3}^N \alpha_i y_i K_{i1} + \alpha_1^{\text{old}} y_1 K_{11} + \alpha_2^{\text{old}} y_2 K_{21} + b^{\text{old}} - y_1$$

The first two items of Eq. (7.114) can be written as:

$$y_1 - \sum_{i=3}^N \alpha_i y_i K_{i1} = -E_1 + \alpha_1^{\text{old}} y_1 K_{11} + \alpha_2^{\text{old}} y_2 K_{21} + b^{\text{old}}$$

Substitute it into (7.114), and we get

$$b_1^{\text{new}} = -E_1 - y_1 K_{11}(\alpha_1^{\text{new}} - \alpha_1^{\text{old}}) - y_2 K_{21}(\alpha_2^{\text{new}} - \alpha_2^{\text{old}}) + b^{\text{old}} \quad (7.115)$$

Similarly, if  $0 < \alpha_2^{\text{new}} < C$ , then,

$$b_2^{\text{new}} = -E_2 - y_1 K_{12}(\alpha_1^{\text{new}} - \alpha_1^{\text{old}}) - y_2 K_{22}(\alpha_2^{\text{new}} - \alpha_2^{\text{old}}) + b^{\text{old}} \quad (7.116)$$

If  $\alpha_1^{\text{new}}$  and  $\alpha_2^{\text{new}}$  both satisfy the condition  $0 < \alpha_i^{\text{new}} < C$ ,  $i = 1, 2$ , then  $b_1^{\text{new}} = b_2^{\text{new}}$ . If  $\alpha_1^{\text{new}}$  and  $\alpha_2^{\text{new}}$  are 0 or  $C$ , then  $b_1^{\text{new}}$  and  $b_2^{\text{new}}$  and the numbers between them are the threshold values satisfying KKT conditions, at which point their midpoint is selected as  $b^{\text{new}}$ .

After each optimization of the two variables, the corresponding  $E_i$  values must be updated and saved in the list. The update of  $E_i$  value needs the value of  $b^{\text{new}}$  and the corresponding  $\alpha_j$  of all support vectors:

$$E_i^{\text{new}} = \sum_S y_j \alpha_j K(x_i, x_j) + b^{\text{new}} - y_i \quad (7.117)$$

where  $S$  is the set of all support vectors  $x_j$ .

### 7.4.3 SMO Algorithm

#### Algorithm 7.5 (SMO Algorithm)

Input: training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{+1, -1\}, i = 1, 2, \dots, N$ , the precision  $\varepsilon$ ;

Output: the approximate solution  $\hat{\alpha}$ .

- (1) Take the initial value  $\alpha^{(0)} = 0$ , and let  $k = 0$ ;
- (2) Select optimization variables  $\alpha_1^{(k)}, \alpha_2^{(k)}$ , analytically solve the optimization problem of the two variables (7.101)~(7.103), find the optimal solution  $\alpha_1^{(k+1)}, \alpha_2^{(k+2)}$ , and update  $\alpha$  to  $\alpha^{(k+1)}$ ;
- (3) If the stop condition is met within the precision  $\varepsilon$

$$\sum_{i=1}^N \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, 2, \dots, N$$

$$y_i \cdot g(x_i) \begin{cases} \geq 1, \{x_i | \alpha_i = 0\} \\ = 1, \{x_i | 0 < \alpha_i < C\} \\ \leq 1, \{x_i | \alpha_i = C\} \end{cases}$$

among them,

$$g(x_i) = \sum_{j=1}^N \alpha_j y_j K(x_j, x_i) + b$$

Then go to (4); otherwise, let  $k = k + 1$ , and go to (2);  
(4) Take  $\hat{\alpha} = \alpha^{(k+1)}$ .

## Summary

1. The simplest case of a Support Vector Machine is the linear SVM in linearly separable case or a hard-margin SVM. The condition for constructing it is that the training data is linearly separable. The learning strategy is the maximum margin method. It can be expressed as a convex quadratic programming problem whose primal optimization problem is

$$\begin{aligned} & \min_{w, b} \frac{1}{2} \|w\|^2 \\ & \text{s.t.} \\ & y_i(w \cdot x_i + b) - 1 \geq 0, \quad i = 1, 2, \dots, N \end{aligned}$$

The solution of the optimization problem is  $w^*$ ,  $b^*$ , and the linearly separable SVM is obtained. The separating hyperplane is:

$$w^* \cdot x + b^* = 0$$

The classification decision function is:

$$f(x) = \text{sign}(w^* \cdot x + b^*)$$

The function margin and geometric margin are important concepts in the maximum margin method.

The optimal solution of linear SVM in the linearly separable case exists and is unique. The instance points located on the margin boundary are support vectors. The optimal separating hyperplane is completely determined by support vectors.

The dual problem of the quadratic programming problem is:

$$\begin{aligned} & \min \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^N \alpha_i \\ & \text{s.t.} \\ & \sum_{i=1}^N \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, 2, \dots, N \end{aligned}$$

Generally, the linear SVM in the linearly separable case is learned by solving dual problems, i.e., the optimal value  $\alpha^*$  of the dual problem is first solved, and then the optimal values  $w^*$  and  $b^*$  are computed to obtain the separating hyperplane and the classification decision function.

2. In reality, there are few cases where training data are linearly separable, and the training data are often approximately linearly separable. In this case, the linear SVM or the soft marginSVM is used. The linear SVM is the most basic Support Vector Machine.

For noise or exceptions, the convex quadratic programming problem for linear SVM learning is obtained by introducing the slack variable  $\xi_i$  to make it “separable”. The primal optimization problem is:

$$\begin{aligned} \min_{w, b, \xi} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N \\ & \xi_i \geq 0, \quad i = 1, 2, \dots, N \end{aligned}$$

Find the solutions  $w^*$  and  $b^*$  of the primal optimization problem to obtain a linear SVM, whose separating hyperplane is

$$w^* \cdot x + b^* = 0$$

The classificationdecision function is:

$$f(x) = \text{sign}(w^* \cdot x + b^*)$$

The solution  $w^*$  of a linear SVM is unique, but  $b^*$  is not necessarily unique.

The dual problem is:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^N \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N \end{aligned}$$

The dual learning algorithmfor the linear SVM first solves the dual problem to obtain the optimal solution  $\alpha^*$ , then obtains the optimal solutions  $w^*$  and  $b^*$  of the original problem to get the separating hyperplane and the classification decision function.

In the solution  $a^*$  of the dual problem, the instance points  $x_i$  satisfying  $\alpha_i^* > 0$  are called support vectors. The support vectors can be on the separating boundary, between the separating boundary and the separating hyperplane, or on the misclassified side of the separating hyperplane. The optimal separating hyperplane is completely determined by the support vector.

The linear SVM learning is equivalent to minimizing the hinge function of the second-order norm regularization

$$\sum_{i=1}^N [1 - y_i(w \bullet x + b)]_+ + \lambda \|w\|^2$$

### 3. Non-Linear Support Vector Machine

The non-linear classification problem in the input space can be transformed into a linear classification problem in a certain high-dimensional feature space through the non-linear transformation. The linear SVM is learned in the high-dimensional feature space. Since in the dual problem of linear SVM learning, both the objective function and the classification decision function only involve the inner product between the instances, there is no need to explicitly specify the non-linear transformation but to replace the inner product among them with the kernel function. The kernel function represents the inner product between two instances after a non-linear transformation. Specifically,  $K(x, z)$  is a kernel function, or a positive definite kernel, implying that there exists a mapping  $\varphi(x) : \chi \rightarrow H$  from the input space  $\chi$  to the feature space  $H$ . For any  $x, z \in \chi$ , we have

$$K(x, z) = \phi(x) \bullet \phi(z)$$

The necessary and sufficient conditions for the symmetric function  $K(x, z)$  to be a positive definite kernel are as follows: for any  $x_i \in \chi$ ,  $i = 1, 2, \dots, m$  and any positive integer  $m$ , the Gram matrix corresponding to the symmetric function  $K(x, z)$  is semi-positive definite.

Therefore, in the dual problem of linear SVM learning, the kernel function  $K(x, z)$  is used to replace the inner product, and the solution obtained is a non-linear Support Vector Machine

$$f(x) = \text{sign}\left(\sum_{i=1}^N \alpha_i^* y_i K(x, x_i) + b^*\right)$$

### 4. SMO algorithm

The SMO algorithm is a fast algorithm for SVM learning, which is characterized by continuously decomposing the primal quadratic programming problem into quadratic programming sub-problems with only two variables and solving the sub-problems analytically until all variables satisfy the KKT condition. In

this way, the optimal solution of the primal quadratic programming problem is obtained by the heuristic method. Since the sub-problems have analytical solutions, they are quickly computed every time. Although there are many computations of the sub-problems, it is generally efficient.

## Further Reading

The linear Support Vector Machine (soft margin) was proposed by Cortes and Vapnik [3]. At the same time, Boser, Guyon and Vapnik introduced kernel tricks and proposed non-linear SVMs [4]. Drucker et al. extended it to support vector regression [5]. Vapnik Vladimir discussed the generalization capability of SVMs in his book *Statistical Learning Theory* [6].

SMO, a fast learning algorithm for SVMs was proposed by Platt [7], and the SVM Light implemented by Joachims and LIBSVM software packages implemented by Chang and Lin are widely used.<sup>2</sup>

The primal SVM is a two-class classification model, which has been extended to multi-class classification SVMs [8, 9], and structural SVMs for structure prediction [10].

The literature on the Support Vector Machine is extensive. An introduction to SVM can be found in references [1, 2, 11, 12]. Kernel methods are considered more general machine learning methods than the SVM method. The introduction of kernel methods can be found in references [13–15].

## Exercises

- 7.1. Compare the dual form of the perceptron with that of the linear separable support vector machines in linearly separable cases.
- 7.2. Given the positive instance points  $x_1 = (1, 2)^T$ ,  $x_2 = (2, 3)^T$ ,  $x_3 = (3, 3)^T$  and the negative instance points  $x_4 = (2, 1)^T$ ,  $x_5 = (3, 2)^T$ , try to find the maximum margin separating hyperplane and the classification decision function, and draw the separating hyperplane, margin boundary and support vectors on the graph.
- 7.3. The Linear Support Vector Machine can also be defined in the following form:

$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i^2$$

s.t.

$$y_i(w \bullet x_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, N$$

$$\xi_i \geq 0, \quad i = 1, 2, \dots, N$$

Try to find its dual form.

- 7.4. Prove the positive definite kernel function of the inner product:

$$K(x, z) = (x \bullet z)^p$$

---

<sup>2</sup> SVM Light: <http://svmlight.joachims.org/>. LIBSVM: <http://www.csie.ntu.edu.tw/cjlin/libsvm/>.

is a positive definite kernel function, where  $p$  is a positive integer,  $x, z \in \mathbb{R}^n$ .

## References

1. Deng N-Y, Tian Y-J. A new approach in data mining—support vector machines. Beijing: Science Press; 2004.
2. Cristianini N, Shawe-Taylor J. An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press; 2000.
3. Cortes C, Vapnik V. Support-vector networks. *Mach Learn*. 1995;20(3):273–97.
4. Boser BE, Guyon IM, Vapnik VN. A training algorithm for optimal margin classifiers. In: Haussler D, editor. Proc of the 5th annual ACM workshop on COLT. Pittsburgh, PA; 1992, 144–152.
5. Drucker H, Burges CJC, Kaufman L, et al. Support vector regression machines. In: Advances in neural information processing systems, vol. 9, NIPS. The MIT Press; 1996, 155–161.
6. Vapnik V. The nature of statistical learning theory. Berlin: Springer-Verlag; 1995.
7. Platt JC. Fast training of support vector machines using sequential minimal optimization. Microsoft Research, <http://research.microsoft.com/apps/pubs/?id=68391>.
8. Weston JAE, Watkins C. Support vector machines for multi-class pattern recognition. In: Proceedings of the 7th European symposium on articial neural networks; 1999.
9. Crammer K, Singer Y. On the algorithmic implementation of multiclass kernel-based machines. *J Mach Learn Res*; 2001, 2(Dec): 265–292.
10. Tsochantaridis I, Joachims T, Hofmann T, et al. Large margin methods for structured and interdependent output variables. *JMLR*; 2005, 6: 1453–1484.
11. Burges CJC. A tutorial on support vector machines for pattern recognition. Bell Laboratories, Lucent Technologies; 1997.
12. Deng N-Y, Tian Y-J. Support vector machines—theory, algorithms and extensions. Beijing: Science Press; 2009.
13. Schölkopf B, Smola AJ, Bach F. Learning with kernels: support vector machines, regularization, optimization, and beyond. The MIT Press; 2002.
14. Herbrich R. Learning kernel classifiers: theory and algorithms. The MIT Press; 2002.
15. Hofmann T, Schölkopf B, Smola AJ. Kernel methods in machine learning. *Ann Stat*. 2008;36(3):1171–220.

# Chapter 8

## Boosting

The boosting method is a frequently-used machine learning method that is widely applied and effective. In the classification problem, it learns multiple classifiers by changing the weight of the training data and linearly combines these classifiers to improve the classification performance.

This chapter first introduces the idea of the boosting method and the representative boosting algorithm AdaBoost; then discusses why AdaBoost can improve learning accuracy through training error analysis; then interprets AdaBoost from the perspective of the forward stepwise addition model; finally describes more specific examples of the lifting method—Boosting tree. Freund and Schapire proposed the AdaBoost algorithm in 1995, and the boosting tree was proposed by Friedman et al. in 2000.

### 8.1 AdaBoost Algorithm

#### 8.1.1 *The Basic Idea of Boosting*

The boosting method is based on the idea that for a complex task, an appropriate combination of several experts' judgments is better than the individual judgment of any one of them. It is also known as two heads are better than one.

Historically, Kearns and Valiant first proposed the concepts of “strongly learnable” and “weakly learnable”. They pointed out that in the learning framework of probably approximately correct (PAC), a concept (a class) is said to be strongly learnable if there is a polynomial learning algorithm that can learn it with high accuracy; a concept is weakly learnable if there exists a polynomial learning algorithm that can learn it with a correct rate only slightly better than random guesses. It is very interesting to note that Schapire later proved that being strongly learnable and being weakly learnable are equivalent, i.e., in the framework of PAC learning, a sufficient and necessary condition for a concept to be strongly learnable is that the concept is weakly learnable.

The question then becomes, if a “weak learning algorithm” has been discovered in learning, can it be boosted to a “strong learning algorithm”? It is well known that it’s usually much easier to find weak learning algorithms than strong ones. Then, how to implement boosting specifically is the problem to be solved when developing boosting methods. There is a lot of research on boosting methods, and many algorithms have been proposed. The most representative one is the AdaBoost algorithm.

For classification problems, given a training dataset, it is much easier to find rough classification rules (weak classifiers) than accurate classification rules (strong classifiers). The boosting methods start from the weak learning algorithm, learn repeatedly to obtain a series of weak classifiers (also called basic classifiers), and then combine these weak classifiers to form a strong classifier. Most of the boosting methods change the probability distribution of the training data (the weight distribution of the training data) and use the weak learning algorithm to learn a series of weak classifiers for different training data distributions.

Thus, there are two questions for boosting methods: first, how to change the weight or probability distribution of the training data in each round; and second, how to combine weak classifiers into a strong classifier. Regarding the first question, AdaBoost increases the weight of samples misclassified by the previous round of weak classifiers and decreases the weight of correctly classified samples. Hence, the data that has not been correctly classified will receive greater attention from the weak classifiers in the next round due to the increase in their weight. Therefore, the classification problem is divided and solved by a series of weak classifiers. As for the second question, the combination of weak classifiers, AdaBoost adopts a weighted majority voting method. Specifically, the weaker classifiers with smaller classification error rates are weighted more to play a more important role in voting, and weaker classifiers with higher classification error rates are weighted less to play a less vital role in the vote.

The ingenuity of AdaBoost lies in its natural and effective implementation of these ideas in one algorithm.

### 8.1.2 AdaBoost Algorithm

The AdaBoost algorithm is now described. Supposing a training dataset for two-class classification is given

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

where each sample point is composed of instances and labels. Here, the instances  $x_i \in \mathcal{X} \subseteq R^n$  and the labels  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $\mathcal{X}$  is the instance space, and  $\mathcal{Y}$  is the label set. AdaBoost uses the following algorithm to learn a series of weak

classifiers or basic classifiers from training data and linearly combine these weak classifiers into a strong classifier.

**Algorithm 8.1 (AdaBoost)**

Input: training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , where  $x_i \in \mathcal{X} \subseteq \mathbf{R}^n$ ,  $y_i \in \mathcal{Y} = \{-1, +1\}$ ; weak learning algorithm;

Output: final classifier  $G(x)$ .

- (1) Initialize the weight distribution of the training data

$$D_1 = (w_{11}, \dots, w_{1i}, \dots, w_{1N}), \quad w_{1i} = \frac{1}{N}, \quad i = 1, 2, \dots, N$$

- (2) For  $m = 1, 2, \dots, M$

- (a) Use the training dataset with the weight distribution  $D_m$  to learn to get the basic classifier

$$G_m(x) : \mathcal{X} \rightarrow \{-1, +1\}$$

- (b) Calculate the classification error rate of  $G_m(x)$  on the training dataset

$$e_m = \sum_{i=1}^N P(G_m(x_i) \neq y_i) = \sum_{i=1}^N w_{mi} I(G_m(x_i) \neq y_i) \quad (8.1)$$

- (c) Calculate the coefficient of  $G_m(x)$

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \quad (8.2)$$

The logarithm here is a natural logarithm.

- (d) Update the weight distribution of the training dataset

$$D_{m+1} = (w_{m+1,1}, \dots, w_{m+1,i}, \dots, w_{m+1,N}) \quad (8.3)$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N \quad (8.4)$$

Here,  $Z_m$  is the normalization factor

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i)) \quad (8.5)$$

which makes  $D_{m+1}$  a probability distribution.

(3) Construct a linear combination of basic classifiers

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x) \quad (8.6)$$

get the final classifier

$$\begin{aligned} G(x) &= \text{sign}(f(x)) \\ &= \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right) \end{aligned} \quad (8.7)$$

The AdaBoost algorithm is explained as follows:

- Step (1) Assume that the training dataset has a uniform weight distribution, i.e., each training sample plays the same role in learning the basic classifier. This assumption ensures that the first step can learn the basic classifier  $G_1(x)$  on the original data.
- Step (2) AdaBoost learns the basic classifier repeatedly, and in each round  $m = 1, 2, \dots, M$  performs the following operations in sequence:
  - (a) Use the training dataset weighted by the current distribution  $D_m$  to learn the basic classifier  $G_m(x)$ .
  - (b) Calculate the classification error rate of the basic classifier  $G_m(x)$  on the weighted training dataset:

$$\begin{aligned} e_m &= \sum_{i=1}^N P(G_m(x_i) \neq y_i) \\ &= \sum_{G_m(x_i) \neq y_i} w_{mi} \end{aligned} \quad (8.8)$$

Here,  $w_{mi}$  represents the weight of the  $i$ -th instance in the  $m$ -th round,  $\sum_{i=1}^N w_{mi} = 1$ . This shows that the classification error rate of  $G_m(x)$  on the weighted training dataset is the sum of the weights of the samples misclassified by  $G_m(x)$ , indicating the relationship between the data weight distribution  $D_m$  and the classification error rate.

- (c) Calculate the coefficient  $\alpha_m$  of the basic classifier  $G_m(x)$ .  $\alpha_m$  represents the importance of  $G_m(x)$  in the final classifier. It can be seen from Eq. (8.2) that when  $e_m \leq \frac{1}{2}$ ,  $\alpha_m \geq 0$ , and  $\alpha_m$  increases with the decrease of  $e_m$ , therefore, the smaller the classification error rate, the greater the role of the basic classifier in the final classifier.
- (d) Update the weight distribution of the training data to prepare for the next round. Equation (8.4) can be written as:

**Table 8.1** Training data table ( $x, y$ )

Number	1	2	3	4	5	6	7	8	9	10
$x$	0	1	2	3	4	5	6	7	8	9
$y$	1	1	1	-1	-1	-1	1	1	1	-1

$$w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases}$$

It can be seen that the weight of the sample misclassified by the basic classifier  $G_m(x)$  can be expanded, while the weight of the correctly classified sample can be reduced. Comparing the two, from Eq. (8.2), the weight of the misclassified sample is amplified  $e^{2\alpha_m} = \frac{1-e_m}{e_m}$  times. Therefore, misclassified samples play a greater role in the next round of learning. It does not change the training data given, and constantly changes the weight distribution of the training data, so that the training data plays a different role in the learning of the basic classifier, which is a feature of AdaBoost.

Step (3) The linear combination  $f(x)$  realizes the weighted voting of  $M$  basic classifiers. The coefficient  $\alpha_m$  indicates the importance of the basic classifier  $G_m(x)$ , where the sum of all  $\alpha_m$  is not 1. The sign of  $f(x)$  determines the class of instance  $x$ , and the absolute value of  $f(x)$  represents the certainty of classification. The use of linear combinations of basic classifiers to construct the final classifier is another feature of AdaBoost.

### 8.1.3 AdaBoost Example<sup>1</sup>

**Example 8.1** Given the training data shown in Table 8.1. Assuming that the weak classifier is generated by  $x < v$  or  $x > v$ , its threshold  $v$  makes the classification error rate of the classifier on the training dataset the lowest. Try the AdaBoost algorithm to learn a strong classifier.

**Solution** Initialize data weight distribution

$$D_1 = (w_{11}, w_{12}, \dots, w_{110}) \\ w_{1i} = 0.1, i = 1, 2, \dots, 10$$

For  $m = 1$ ,

- (a) On the training data with weight distribution  $D_1$ , the classification error rate is the lowest when the threshold  $v$  is set to 2.5, so the basic classifier is

---

<sup>1</sup> The example is cited from <http://www.csie.edu.tw>.

$$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$

- (b) The error rate of  $G_1(x)$  on the training dataset  $e_1 = P(G_1(x_i) \neq y_i) = 0.3$ .
- (c) Calculate the coefficient of  $G_1(x)$ :  $\alpha_1 = \frac{1}{2} \log \frac{1-e_1}{e_1} = 0.4236$ .
- (d) Update the weight distribution of the training data:

$$\begin{aligned} D_2 &= (w_{21}, \dots, w_{2i}, \dots, w_{210}) \\ w_{2i} &= \frac{w_{1i}}{Z_1} \exp(-\alpha_1 y_i G_1(x_i)), \quad i = 1, 2, \dots, 10 \\ D_2 &= (0.07143, 0.07143, 0.07143, 0.07143, 0.07143, \\ &\quad 0.07143, 0.16667, 0.16667, 0.16667, 0.07143) \\ f_1(x) &= 0.4236 G_1(x) \end{aligned}$$

The classifier  $\text{sign}[f_1(x)]$  has 3 misclassified points on the training dataset.  
For  $m = 2$ ,

- (a) On the training data with weight distribution  $D_2$ , when the classification error rate is the lowest when the threshold  $v$  is set to 8.5, so the basic classifier is

$$G_2(x) = \begin{cases} 1 & x < 8.5 \\ -1 & x > 8.5 \end{cases}$$

- (b) The error rate of  $G_2(x)$  on training dataset  $e_2 = 0.2143$ .
- (c) Calculate the coefficient of  $G_2(x)$ :  $\alpha_2 = 0.6496$ .
- (d) Update the weight distribution of the training data:

$$\begin{aligned} D_3 &= (0.455, 0.455, 0.455, 0.1667, 0.1667, 0.1667, \\ &\quad 0.1060, 0.1060, 0.1060, 0.0455) \\ f_2(x) &= 0.4236 G_1(x) + 0.6496 G_2(x) \end{aligned}$$

The classifier  $\text{sign}[f_2(x)]$  has 3 misclassified points on the training dataset.  
For  $m = 3$ ,

- (a) On the training data with weight distribution  $D_3$ , when the classification error rate is the lowest when the threshold  $v$  is set to 2.5, so the basic classifier is

$$G_3(x) = \begin{cases} 1, & x < 5.5 \\ -1, & x > 5.5 \end{cases}$$

- (b) The error rate of  $G_3(x)$  on training dataset  $e_3 = 0.1820$ .
- (c) Calculate the coefficient of  $G_3(x)$ :  $\alpha_3 = 0.7514$ .
- (d) Update the weight distribution of the training data:

$$D_4 = (0.125, 0.125, 0.125, 0.102, 0.102, 0.102, 0.065, 0.065, 0.065, 0.125)$$

So get:

$$f_3(x) = 0.4236G_1(x) + 0.6496G_2(x) + 0.7514G_3(x)$$

The classifier  $\text{sign}[f_3(x)]$  has 0 misclassified point on training dataset.  
So the final classifier is

$$\begin{aligned} G(x) &= \text{sign}[f_3(x)] \\ &= \text{sign}[0.4236G_1(x) + 0.6496G_2(x) + 0.7514G_3(x)] \end{aligned}$$

## 8.2 Training Error Analysis of AdaBoost Algorithm

The most basic property of AdaBoost is that it can continuously reduce the training error during the learning process, that is, the classification error rate on the training dataset. There are the following theorems on this problem.

**Theorem 8.1 (AdaBoost Training Error Bound)** The training error bound of the final classifier of the AdaBoost algorithm by

$$\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \frac{1}{N} \sum_i \exp(-y_i f(x_i)) = \prod_m Z_m \quad (8.9)$$

Here,  $G(x)$ ,  $f(x)$ , and  $Z_m$  are given by Eqs. (8.7), (8.6) and (8.5) respectively.

**Proof** When  $G(x_i) \neq y_i$ ,  $y_i f(x_i) < 0$ , so  $\exp(-y_i f(x_i)) \geq 1$ . From this, the first half is directly derived.

The derivation of the latter part will use the Definition Eq. (8.5) of  $Z_m$  and the transformation of (8.4):

$$w_{mi} \exp(-\alpha_m y_i G_m(x_i)) = Z_m w_{m+1,i}$$

It is derived as follows:

$$\begin{aligned} \frac{1}{N} \sum_i \exp(-y_i f(x_i)) &= \frac{1}{N} \sum_i \exp\left(-\sum_{m=1}^M \alpha_m y_i G_m(x_i)\right) \\ &= \sum_i w_{1i} \prod_{m=1}^M \exp(-\alpha_m y_i G_m(x_i)) \end{aligned}$$

$$\begin{aligned}
&= Z_1 \sum_i w_{2i} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \\
&= Z_1 Z_2 \sum_i w_{3i} \prod_{m=3}^M \exp(-\alpha_m y_i G_m(x_i)) \\
&= \dots \\
&= Z_1 Z_2 \cdots Z_{M-1} \sum_i w_{Mi} \exp(-\alpha_M y_i G_M(x_i)) \\
&= \prod_{m=1}^M Z_m
\end{aligned}$$

This theorem shows that an appropriate  $G_m$  can be selected in each round to minimize  $Z_m$ , so that the training error drops the fastest. For the two-class classification problem, the following results are obtained.

**Theorem 8.2 (AdaBoost Training Error Bound of the Two-Class Classification Problem)**

$$\begin{aligned}
\prod_{m=1}^M Z_m &= \prod_{m=1}^M \left[ 2\sqrt{e_m(1 - e_m)} \right] \\
&= \prod_{m=1}^M \sqrt{(1 - 4\gamma_m^2)} \\
&\leq \exp\left(-2 \sum_{m=1}^M \gamma_m^2\right)
\end{aligned} \tag{8.10}$$

where  $\gamma_m = \frac{1}{2} - e_m$ .

**Proof** From the Definition Equation (8.5) of  $Z_m$  and the Definition Eq. (8.8), we can get

$$\begin{aligned}
Z_m &= \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i)) \\
&= \sum_{y_i = G_m(x_i)} w_{mi} e^{-\alpha_m} + \sum_{y_i \neq G_m(x_i)} w_{mi} e^{\alpha_m} \\
&= (1 - e_m) e^{-\alpha_m} + e_m e^{\alpha_m} \\
&= 2\sqrt{e_m(1 - e_m)} \\
&= \sqrt{1 - 4\gamma_m^2}
\end{aligned} \tag{8.11}$$

as for the inequality

$$\prod_{m=1}^M \sqrt{(1 - 4\gamma_m^2)} \leq \exp\left(-2 \sum_{m=1}^M \gamma_m^2\right)$$

the inequality  $\sqrt{(1 - 4\gamma_m^2)} \leq \exp(-2\gamma_m^2)$  can be derived from the Taylor expansion of  $e^x$  and  $\sqrt{1 - x}$  at the point  $x = 0$ .

**Corollary 8.1** If there exists  $\gamma > 0$  with  $\gamma_m \geq \gamma$  for all  $m$ , then

$$\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \exp(-2M\gamma^2) \quad (8.12)$$

This shows that under this condition, the training error of AdaBoost decreases at an exponential rate, which is deservedly very attractive.

Note that the AdaBoost algorithm does not need to know the lower bound  $\gamma$ , which is what Freund and Schapire considered when designing the AdaBoost. Unlike some early boosting methods, AdaBoost is adaptable, i.e. it can adapt to the individual training error rate of weak classifiers. It is also the origin of the name (adaptive promotion), Ada being short for Adaptive.

## 8.3 Explanation of AdaBoost Algorithm

Another interpretation of the AdaBoost algorithm is that it can be considered as a two-class classification learning method when the model is additive, the loss function is an exponential function, and the learning algorithm is forward stepwise.

### 8.3.1 Forward Stepwise Algorithm

Consider the additive model

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m) \quad (8.13)$$

where  $b(x; \gamma_m)$  is the basis function,  $\gamma_m$  is the parameter of the basis function, and  $\beta_m$  is the coefficient of the basis function. Obviously, Equation (8.6) is an additive model.

Given the training data and the loss function  $L(y, f(x))$ , learning the additive model  $f(x)$  becomes a problem of minimizing the experience risk or the minimization of the loss function:

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^N L\left(y_i, \sum_{m=1}^M \beta_m b(x_i; \gamma_m)\right) \quad (8.14)$$

Generally, this is a complex optimization problem. The idea of the forward stepwise algorithm to solve this optimization problem is: due to an additive model is being learned, if learning from front to back, only one basis function and its coefficients are learned at each step, and gradually approximate the optimization objective function formula (8.14), then the complexity of optimization can be simplified. Specifically, each step only needs to optimize the following loss function:

$$\min_{\beta, \gamma} \sum_{i=1}^N L(y_i, \beta b(x_i; \gamma)) \quad (8.15)$$

Given a training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ,  $x_i \in \mathcal{X} \subseteq \mathbf{R}^n$ ,  $y_i \in \mathcal{Y} = \{-1, +1\}$ . The loss function  $L(y, f(x))$  and the set of basis functions  $\{b(x; \gamma)\}$ , the forward stepwise algorithm for learning the addition model  $f(x)$  is as follows.

### Algorithm 8.2 (Forward Stepwise Algorithm)

**Input:** training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ; loss function  $L(y, f(x))$ ; basis function set  $\{b(x; \gamma)\}$ ;

**Output:** the additive model  $f(x)$ .

- (1) Initialize  $f_0(x) = 0$ ;
- (2) For  $m = 1, 2, \dots, M$

(a) Minimize the loss function

$$(\beta_m, \gamma_m) = \operatorname{argmin}_{\beta, \gamma} \sum_{i=1}^N L(y_i f_{m-1}(x_i) + \beta b(x_i; \gamma_m)) \quad (8.16)$$

to get parameters  $\beta_m, \gamma_m$ .

(b) Update

$$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \quad (8.17)$$

- (3) Get the additive model

$$f(x) = f_M(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m) \quad (8.18)$$

In this way, the forward stepwise algorithm will simultaneously solve the optimization problem of all parameters  $\beta_m$  and  $\gamma_m$  from  $m = 1$  to  $M$  and simplify the optimization problem of solving each  $\beta_m$  and  $\gamma_m$  successively.

### 8.3.2 Forward Stepwise Algorithm and AdaBoost

AdaBoost can be derived from the forward stepwise algorithm. This relationship is described by the following theorem.

**Theorem 8.3** The AdaBoost algorithm is a special case of the forward stepwise addition algorithm. In this case, the model is an additive model composed of basic classifiers, and the loss function is an exponential function.

**Proof** The forward stepwise algorithm learns the additive model. When the basis function is the basic classifier, this additive model is equivalent to the final classifier of AdaBoost

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x) \quad (8.19)$$

which is composed of the basic classifier  $G_m(x)$  and its coefficient  $\alpha_m$ ,  $m = 1, 2, \dots, M$ . The forward stepwise algorithm learns the basis function one by one, consistent with the AdaBoost algorithm learning basic classifiers one by one. Now, prove that when the loss function of the forward stepwise algorithm is an exponential loss function

$$L(y, f(x)) = \exp[-yf(x)]$$

the specific operation of its learning is equivalent to that of the AdaBoost algorithm.

Suppose that  $f_{m-1}(x)$  has been obtained through  $m - 1$  iterations

$$\begin{aligned} f_{m-1}(x) &= f_{m-2}(x) + \alpha_{m-1} G_{m-1}(x) \\ &= \alpha_1 G_1(x) + \dots + \alpha_{m-1} G_{m-1}(x) \end{aligned}$$

Then in the  $m$ -th round iteration,  $\alpha_m$ ,  $G_m(x)$  and  $f_m(x)$  are obtained.

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

The goal is to minimize the exponential loss of  $f_m(x)$  on the training dataset  $T$  by the  $\alpha_m$  and  $G_m(x)$  obtained by the forward stepwise algorithm,

$$(\alpha_m, G_m(x)) = \arg \min_{\alpha, G} \sum_{i=1}^N \exp[-y_i(f_{m-1}(x_i) + \alpha G(x_i))] \quad (8.20)$$

Equation (8.20) can be expressed as

$$(\alpha_m, G_m(x)) = \arg \min_{\alpha, G} \sum_{i=1}^N \bar{w}_i \exp[-y_i \alpha G(x_i)] \quad (8.21)$$

Among them,  $\bar{w}_{mi} = \exp[-y_i f_{m-1}(x_i)]$ . Since  $\bar{w}_{mi}$  does not depend on either the  $\alpha$  or  $G$ , it is not related to minimization. However, it depends on  $f_{m-1}(x)$  and changes with each iteration.

Now prove that  $\alpha_m^*$  and  $G_m^*(x)$  that minimize Eq. (8.21) are the  $\alpha_m$  and  $G_m(x)$  obtained by the AdaBoost algorithm. Solving Eq. (8.21) can be done in two steps:

First, find  $G_m^*(x)$ . The  $G(x)$  that minimizes Eq. (8.21) for any  $\alpha > 0$  is given by the following equation.

$$G_m^*(x) = \arg \min_G \sum_{i=1}^N \bar{w}_{mi} I(y_i \neq G(x_i))$$

Among them,  $\bar{w}_{mi} = \exp[-y_i f_{m-1}(x_i)]$ .

This classifier  $G_m^*(x)$  is the basic classifier  $G_m(x)$  of the AdaBoost algorithm because it is the basic classifier that minimizes the classification error rate of the weighted training data in round  $m$ .

After that, find the value of  $\alpha_m^*$ . Refer to Eq. (8.11) and Eq. (8.21) and get

$$\begin{aligned} & \sum_{i=1}^N \bar{w}_{mi} \exp[-y_i \alpha G(x_i)] \\ &= \sum_{y_i = G_m(x_i)} \bar{w}_{mi} e^{-\alpha} + \sum_{y_i \neq G_m(x_i)} \bar{w}_{mi} e^\alpha \\ &= (e^\alpha - e^{-\alpha}) \sum_{i=1}^N \bar{w}_{mi} I(y_i \neq G(x_i)) + e^{-\alpha} \sum_{i=1}^N \bar{w}_{mi} \end{aligned} \quad (8.22)$$

Substituting the obtained  $G_m^*(x)$  into Eq. (8.22), taking the derivative of  $\alpha$ , and making it 0, we obtain  $\alpha$  that minimizes Eq. (8.21).

$$\alpha_m^* = \frac{1}{2} \log \frac{1 - e_m}{e_m}$$

where  $e_m$  is the classification error rate:

$$\begin{aligned} e_m &= \frac{\sum_{i=1}^N \bar{w}_{mi} I(y_i \neq G_m(x_i))}{\sum_{i=1}^N \bar{w}_{mi}} \\ &= \sum_{i=1}^N w_{mi} I(y_i \neq G_m(x_i)) \end{aligned} \quad (8.23)$$

Here  $\alpha_m^*$  is the same as the  $\alpha_m$  in step 2 (c) of the AdaBoost algorithm.

Finally, let's look at the update of sample weights in each round. From

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

and  $\bar{w}_{mi} = \exp[-y_i f_{m-1}(x_i)]$ , we get

$$\bar{w}_{m+1,i} = \bar{w}_{m,i} \exp[-y_i \alpha_m G_m(x)]$$

This is equivalent to the update of sample weights in step 2 (d) of the AdaBoost algorithm, which only differs in the normalization factor.

## 8.4 Boosting Tree

The boosting tree is a boosting method that uses classification trees or regression trees as the basic classifier. It is considered to be one of the best-performing methods in machine learning.

### 8.4.1 Boosting Tree Model

The boosting method uses the additive model (i.e., the linear combination of basis functions) and the forward stepwise algorithm. A boosting method using the decision tree as the basis function is called a boosting tree. The decision tree is a binary classification tree for classification problems and a binary regression tree for regression problems. The basic classifier  $x < v$  or  $x > v$  seen in Example 8.1 can be regarded as a simple decision tree with one root node directly connecting two leaf nodes, namely the so-called decision stump. The boosting tree model can be expressed as an additive model of the decision tree.

$$f_M(x) = \sum_{m=1}^M T(x; \Theta_m) \quad (8.24)$$

where  $T(x; \Theta_m)$  is the decision tree,  $\Theta_m$  is the parameter of the decision tree, and  $M$  is the number of trees.

### 8.4.2 Boosting Tree Algorithm

The boosting tree algorithm adopts the forward stepwise algorithm. Firstly, the initial boosting tree  $f_0(x) = 0$  is determined, and the model of step  $m$  is

$$f_m(x) = f_{m-1}(x) + T(x; \Theta_m) \quad (8.25)$$

where  $f_{m-1}(x)$  is the current model, and the parameters  $\Theta_m$  of the next decision tree are determined by empirical risk minimization:

$$\hat{\Theta}_m = \operatorname{argmin}_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \quad (8.26)$$

The linear combination of trees can fit the training data well, even if the relationship between the input and output in the data is very complex. Thus the boosting tree is a high function learning algorithm.

Next, we discuss the boosting tree learning algorithms for different problems. The main difference is that the loss function is different. It includes the regression problem with square error loss function, the classification problem with exponential loss function, and the general decision problem with general loss function.

For the Two-class classification problem, the boosting tree algorithm only needs to limit the basic classifier in AdaBoost algorithm 8.1 to the Two-class classification tree. The boosting tree algorithm at this time can be regarded as a special case of AdaBoost algorithm, which will not be described in detail here. The following is the boosting tree of the regression problem.

Given a training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ,  $x_i \in \mathcal{X} \subseteq \mathbf{R}^n$ ,  $\mathcal{X}$  is the input space,  $y_i \in \mathcal{Y} \subseteq \mathbf{R}$ ,  $\mathcal{Y}$  is the output space. The problem of regression trees has been discussed in Sect. 5.5. If the input space  $\mathcal{X}$  is divided into  $J$  disjoint regions  $R_1, R_2, \dots, R_J$ , and the output constant  $c_j$  is determined on each region, then the tree can be expressed as

$$T(x; \Theta) = \sum_{j=1}^J c_j I(x \in R_j) \quad (8.27)$$

Among them, the parameter  $\Theta = \{(R_1, c_1), (R_2, c_2), \dots, (R_J, c_J)\}$  represents the region division of the tree and the constants on each region.  $J$  is the complexity of the regression tree, that is, the number of leaf nodes.

The boosting tree for regression problems uses the following forward stepwise algorithm:

$$\begin{aligned} f_0(x) &= 0 \\ f_m(x) &= f_{m-1}(x) + T(x; \Theta), \quad m = 1, 2, \dots, M \\ f_M(x) &= \sum_{m=1}^M T(x; \Theta) \end{aligned}$$

At the  $m$ -th step of the forward stepwise algorithm, given the current model  $f_{m-1}(x)$ , it is necessary to solve

$$\hat{\Theta}_m = \arg \min_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

Get  $\hat{\Theta}_m$ , the parameter of the  $m$ -th tree.  
When using the square error loss function,

$$L(y, f(x)) = (y - f(x))^2$$

the loss becomes

$$\begin{aligned} L(y, f_{m-1}(x) + T(x; \Theta_m)) &= [y - f_{m-1}(x) - T(x; \Theta_m)]^2 \\ &= [r - T(x; \Theta_m)]^2 \end{aligned}$$

Here,

$$r = y - f_{m-1}(x) \quad (8.28)$$

is the residual of the current model fitting data. Therefore, for the boosting tree algorithm for regression problems, it is only necessary to simply fit the residuals of the current model. In this way, the algorithm is quite simple. The boosting tree algorithm of the regression problem is described as follows.

### Algorithm 8.3 (Boosting Tree Algorithm for Regression Problems)

Input: training dataset  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ,  $x_i \in \mathcal{X} \subseteq \mathbf{R}^n$ ,  $y_i \in \mathcal{Y} \subseteq \mathbf{R}$ ;

Output: boosting tree  $f_M(x)$ .

- (1) Initialize  $f_0(x) = 0$
- (2) For  $m = 1, 2, \dots, M$ 
  - (a) Calculate the residual according to Formula (8.27):

$$r_{mi} = y_i - f_{m-1}(x_i), \quad i = 1, 2, \dots, N$$

- (b) The fitting residual  $r_{mi}$  learns a regression tree to get  $T(x; \Theta_m)$ .
- (c) Update  $f_m(x) = f_{m-1}(x) + T(x; \Theta_m)$ .
- (3) Get regression problem boosting tree

$$f_M(x) = \sum_{m=1}^M T(x; \Theta_m)$$

**Example 8.2** Given the training data shown in Table 8.2, the value range of  $x$  is the interval [0.5, 10.5], and the value range of  $y$  is the interval [5.0, 10.0]. Learn the

boosting tree model for this regression problem, consider only use tree stumps as basis functions.

**Solution** According to Algorithm 8.3, the first step is to find  $f_1(x)$  which is the regression tree  $T_1(x)$ .

Step 1 First go through the following optimization problems:

$$\min_S \left[ \min_{c_1} \sum_{x_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2 \right]$$

Solve the cut point  $s$  of the training data:

$$R_1 = \{x | x \leq s\}, \quad R_2 = \{x | x > s\}$$

It is easy to find  $c_1$  and  $c_2$  that make the squared loss error reach the minimum within  $R_1$  and  $R_2$  as

$$c_1 = \frac{1}{N_1} \sum_{x_i \in R_1} y_i, \quad c_2 = \frac{1}{N_2} \sum_{x_i \in R_2} y_i$$

Here  $N_1, N_2$  are the sample points of  $R_1, R_2$ .

Find the cut point of the training data. According to the given data, consider the following cut points:

$$1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5$$

For each cut point, it is not difficult to find the corresponding  $R_1, R_2, c_1, c_2$  and

$$m(s) = \min_{c_1} \sum_{x_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2$$

For example, when  $s = 1.5, R_1 = \{1\}, R_2 = \{2, 3, \dots, 10\}, c_1 = 5.56, c_2 = 7.50,$

$$m(s) = \min_{c_1} \sum_{x_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2 = 0 + 15.72 = 15.72$$

The calculation results of  $s$  and  $m(s)$  are listed as follows (see Table 8.3).

**Table 8.2** Training data table ( $x_i, y_i$ )

$x_i$	1	2	3	4	5	6	7	8	9	10
$y_i$	5.56	5.70	5.91	6.40	6.80	7.05	8.90	8.70	9.00	9.05

**Table 8.3** Calculation data table

$s$	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5
$m(s)$	15.72	12.07	8.36	5.78	3.91	1.93	8.01	11.73	15.74

It can be seen from Table 8.3 that when  $s = 6.5$ ,  $m(s)$  reaches the minimum value, at this time  $R_1 = \{1, 2, \dots, 6\}$ ,  $R_2 = \{7, 8, 9, 10\}$ ,  $c_1 = 6.24$ ,  $c_2 = 8.91$ , so the regression tree  $T_1(x)$  is

$$T_1(x) = \begin{cases} 6.24, & x < 6.5 \\ 8.91, & x \geq 6.5 \end{cases}$$

$$f_1(x) = T_1(x)$$

The residuals of using  $f_1(x)$  to fit the training data are shown in Table 8.4, where  $r_{2i} = y_i - f_1(x_i)$ ,  $i = 1, 2, \dots, 10$ .

Use  $f_1(x)$  to fit the squared loss error of the training data:

$$L(y, f_1(x)) = \sum_{i=1}^{10} (y_i - f_1(x_i))^2 = 1.93$$

Step 2 Step 2 Find  $T_2(x)$ . The method is the same as that of finding  $T_1(x)$ , except that the fitted data are the residuals in Table 8.4. You can get:

$$T_2(x) = \begin{cases} -0.52, & x < 3.5 \\ 0.22, & x \geq 3.5 \end{cases}$$

$$f_2(x) = f_1(x) + T_2(x) \begin{cases} 5.72, & x < 3.5 \\ 6.46, & 3.5 \leq x < 6.5 \\ 9.13, & x \geq 6.5 \end{cases}$$

The square loss error of fitting training data with  $f_2(x)$  is

$$L(y, f_2(x)) = \sum_{i=1}^{10} (y_i - f_2(x_i))^2 = 0.79$$

continue to solve to get

**Table 8.4** Residual error table

$x_i$	1	2	3	4	5	6	7	8	9	10
$r_{2i}$	-0.68	-0.54	-0.33	0.16	0.56	0.81	-0.01	-0.21	0.09	0.14

$$\begin{aligned}
T_3(x) &= \begin{cases} 0.15, & x < 6.5 \\ -0.22, & x \geq 6.5 \end{cases} \quad L(y, f_3(x)) = 0.47, \\
T_4(x) &= \begin{cases} -0.15, & x < 4.5 \\ 0.11, & x \geq 4.5 \end{cases} \quad L(y, f_4(x)) = 0.30, \\
T_5(x) &= \begin{cases} 0.07, & x < 6.5 \\ -0.11, & x \geq 6.5 \end{cases} \quad L(y, f_5(x)) = 0.23, \\
T_6(x) &= \begin{cases} -0.15, & x < 2.5 \\ 0.04, & x \geq 2.5 \end{cases} \\
f_6(x) &= f_5(x) + T_6(x) = T_1(x) + \dots + T_5(x) + T_6(x) \\
&= \begin{cases} 5.63, & x < 2.5 \\ 5.82, & 2.5 \leq x < 3.5 \\ 6.56, & 3.5 \leq x < 4.5 \\ 6.83, & 4.5 \leq x < 6.5 \\ 8.95, & x \geq 6.5 \end{cases}
\end{aligned}$$

The square loss error of fitting training data with  $f_6(x)$  is

$$L(y, f_6(x)) = \sum_{i=1}^{10} (y_i - f_6(x_i))^2 = 0.17$$

If the error requirement is satisfied, then  $f(x) = f_6(x)$  is the boosting tree.

### 8.4.3 Gradient Boosting

The boosting tree uses the additive model and the forward stepwise algorithm to realize the optimization process of learning. When the loss function is a square loss function and an exponential loss function, each step of optimization is very simple. But for the general loss function, it is not easy to optimize every step. To solve this problem, Friedman proposed a gradient boosting algorithm. This is an approximation method using the steepest descent method. The key is to fit a regression tree using the negative gradient of the loss function

$$-\left[ \frac{\partial L(y, f(x_i))}{\partial f(x_i)} \right]_{f(x) = f_{m-1}(x)}$$

as an approximation of the residuals in the regression problem boosting tree algorithm.

**Algorithm 8.4 (Gradient Boosting Algorithm)**

Input: training dataset.

$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ,  $x_i \in \mathcal{X} \subseteq \mathbf{R}^n$ ,  $y_i \in \mathcal{Y} \subseteq \mathbf{R}$ ; loss function  $L(y, f(x))$ ;

Output: Regression Tree  $\hat{f}(x)$ .

(1) Initialize

$$f_0(x) = \arg \min_c \sum_{i=1}^N L(y_i, c)$$

(2) For  $m = 1, 2, \dots, M$

(a) For  $i = 1, 2, \dots, N$ , calculate

$$r_{mi} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x) = f_{m-1}(x)}$$

- (b) Fitting a regression tree to  $r_{mi}$  in order to get  $\mathbf{R}_{mj}$  of the leaf node area of the + tree, where  $j = 1, 2, \dots, J$ .  
(c) For  $j = 1, 2, \dots, J$ , calculate

$$c_{mj} = \arg \min_c \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)$$

$$(d) \text{ Update } f_m(x) = f_{m-1}(x) + \sum_{j=1}^J c_{mj} I(x \in R_{mj})$$

(3) Get the regression tree

$$\hat{f}(x) = f_M(x) = \sum_{m=1}^M \sum_{j=1}^J c_{mj} I(x \in R_{mj})$$

The algorithm initializes in step 1, estimating the constant value that minimizes the loss function, which is a tree with only one root node. Step 2 (a) calculates the value of the negative gradient of the loss function in the current model as an estimate of the residual. For the square loss function, it is commonly referred to as the residual; for the general loss function, it is the approximation of the residual. Step 2 (b) estimates the regressed leaf node regions to fit the approximation of the residuals. Step 2 (c) minimizes the loss function by estimating the value of the leaf node area using a linear search. Step 2 (d) updates the regression tree. Step 3 obtains  $\hat{f}(x)$ , the final model of the output.

## Summary

- The boosting method is to upgrade the weak learning algorithm to the strong learning algorithm. In classification learning, the boosting method constructs a series of basic classifiers (weak classifiers) by repeatedly modifying the weight distribution of training data and combines these basic classifiers linearly to form a strong classifier. A representative boosting method is the AdaBoost algorithm.

The AdaBoost model is a linear combination of weak classifiers:

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x)$$

- AdaBoost algorithm is characterized by learning one basic classifier at a time through iteration. In each iteration, the weight of the data that was misclassified by the previous classifier is increased, while the weight of the data that was correctly classified is decreased. Finally, AdaBoost uses the linear combination of the basic classifiers as a strong classifier and gives a large weight to the basic classifier with a small classification error rate and a small weight to the basic classifier with a large classification error rate.
- The training error analysis of AdaBoost shows that each iteration of AdaBoost can reduce its classification error rate on the training dataset, which demonstrates its effectiveness as a boosting method.
- One explanation of the AdaBoost algorithm is that it is actually an implementation of a forward stepwise algorithm. In this method, the model is additive, the loss function is an exponential loss, and the algorithm is forward stepwise.

Minimize the loss function in each step

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

to obtain the parameters  $\beta_m, \gamma_m$ .

- The boosting tree is a method of boosting based on a classification tree or a regression tree. The boosting tree is considered one of the most effective methods in machine learning.

## Further Reading

An introduction to the boosting method can be found in the literature [1, 2]. PAC learning can be found in the literature [3]. The relationship between strongly learnable and weakly learnable can be found in [4]. The earliest publication on AdaBoost is the literature [5]. Refer to [6] for the explanation of AdaBoost's forward stepwise additive model and [6, 7] for the boosting tree and gradient boosting algorithm. AdaBoost was originally used for the binary classification, but Schapire and Singer extended it

**Table 8.5** Candidate profile data

	1	2	3	4	5	6	7	8	9	10
Physical condition	0	0	1	1	1	0	1	1	1	0
Business ability	1	3	2	1	2	1	1	1	3	2
Development potential	3	1	2	3	3	2	2	1	1	1
Classification	-1	-1	-1	-1	-1	-1	1	1	-1	-1

to multi-class classification problems [8]. The relationship between AdaBoost and Logistic Regression has also been studied [9].

## Exercises

- 8.1 A company tests the candidates' physical condition, business ability, development potential when recruiting staff. The physical condition is divided into two levels: PASS 1 and FAIL 0. Business ability and development potential are divided into three levels: UPPER 1, MIDDLE 2, and LOWER 3. The classification includes PASS 1 and FAIL -1. The data of 10 individuals are known, as shown in Table 8.5. Assume that the weak classifier is a decision tree stump. Try to learn a strong classifier using the AdaBoost algorithm.
- 8.2 Compare learning strategies and algorithms for the Support Vector Machine, AdaBoost, and logistic regression models.

## References

1. Freund Y, Schapire RE. A short introduction to boosting. *J Japanese Soc Artif Intell*. 1999;14(5):771–80.
2. Hastie T, Tibshirani R, Friedman JH. The elements of statistical learning: data mining, inference, and prediction. NY: Springer; 2001.
3. Valiant LG. A theory of the learnable. *Commun ACM*. 1984;27(11):1134–42.
4. Schapire R. The strength of weak learnability. *Mach Learn*. 1990;5(2):197–227.
5. Freund Y, Schapire RE. A decision-theoretic generalization of on-line learning and an application to boosting. Computational learning theory. EuroCOLT 1995. Lecture notes in computer science, vol. 904. Berlin, Heidelberg: Springer; 1995. p. 23–37.
6. Friedman JH, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann Stat*. 2000;28(2):337–407.
7. Friedman JH. Greedy function approximation: a gradient boosting machine. *Ann Stat*. 2001;29(5):1189–232.
8. Schapire RE, Singer Y. Improved boosting algorithms using confidence-rated predictions. *Mach Learn*. 1999;37(3):297–336.
9. Collins M, Schapire RE, Singer Y. Logistic regression, AdaBoost and Bregman distances. *Mach Learn*. 2002;48(1):253–85.

# Chapter 9

## EM Algorithm and Its Extensions

The EM algorithm is an iterative algorithm, summarized by Dempster et al. in [1], for the maximum likelihood estimation, or maximum a posteriori estimation of the parameters of a probabilistic model with hidden variables. Each iteration of the EM algorithm consists of two steps: E-step, find the expectation, and M-step, find the maximization. Therefore, this algorithm is called the expectation maximization algorithm, or EM algorithm for short. This chapter first describes the EM algorithm, then discusses the convergence of the EM algorithm; as an application of the EM algorithm, introduces the learning of the Gaussian mixture model; and finally describes the extension of the EM algorithm, the GEM algorithm.

### 9.1 Introduction of the EM Algorithm

Probabilistic models sometimes contain both observable variables and hidden or latent variables. If the variables of a probabilistic model are all observable variables, then given the data, the model parameters can be estimated directly by the method of maximum likelihood estimation or Bayesian estimation. However, these estimation methods cannot be simply used when the model contains hidden variables. The EM algorithm is the maximum likelihood estimation of the parameters of a probabilistic model with hidden variables or the maximum posterior probability estimation method. We only discuss the maximum likelihood estimation, which is similar to the maximum a posteriori estimation.

#### 9.1.1 EM Algorithm

An example of using the EM algorithm is presented first.

**Example 9.1** (Three-Coin Model) Suppose there are three coins, denoted A, B, and C. The probabilities of these coins coming up heads are  $\pi$ ,  $p$  and  $q$ . Conduct the following coin toss test. First, toss coin A and select coin B or coin C according to its result, with coin B being selected for heads and coin C for tails. Then toss the selected coin, with the result recorded as 1 for heads and 0 for tails. Repeat the test  $n$  times independently (here,  $n = 10$ ). The observation results are as follows:

$$1, 1, 0, 1, 0, 0, 1, 0, 1, 1$$

Suppose that only the result of the coin toss can be observed, but not the process of tossing. The question is how to estimate the probability that all three coins will come up heads, i.e., the parameters of the three-coin model.

**Solution** The three-coin model can be written as

$$\begin{aligned} P(y|\theta) &= \sum_z P(y, z|\theta) = \sum_z P(z|\theta)P(y, z|\theta) \\ &= \pi p^y(1-p)^{1-y} + (1-\pi)q^y(1-q)^{1-y} \end{aligned} \quad (9.1)$$

Here, the random variable  $y$  is the observed variable, indicating that the outcome of a trial observation is 1 or 0; the random variable  $z$  is the hidden variable, indicating the unobserved outcome of tossing coin A; and  $\theta = (\pi, p, q)$  is the model parameter. This model is the generative model for the above data. Note that the data for the random variable  $y$  is observable, and the data for the random variable  $z$  is unobservable.

Denote the observation data as  $Y = (Y_1, Y_2, \dots, Y_n)^T$  and the un-observation data as  $Z = (Z_1, Z_2, \dots, Z_n)^T$ , then the likelihood function of the observation data is

$$P(Y|\theta) = \sum_Z P(Z|\theta)P(Y|Z, \theta) \quad (9.2)$$

i.e.,

$$P(Y|\theta) = \prod_{j=1}^n [\pi p^{y_j}(1-p)^{1-y_j} + (1-\pi)q^{y_j}(1-q)^{1-y_j}] \quad (9.3)$$

Consider finding the maximum likelihood estimate of the model parameters  $\theta = (\pi, p, q)$ , i.e.,

$$\hat{\theta} = \arg \max_{\theta} \log P(Y|\theta) \quad (9.4)$$

This problem has no analytical solution and can only be solved by an iterative method. The EM algorithm is an iterative algorithm that can solve this problem.

The EM algorithm for the above problem is given below, with its derivation being omitted.

The EM algorithm first selects the initial values of the parameters as  $\theta^{(0)} = (\pi^{(0)}, p^{(0)}, q^{(0)})$ , and then iterates through the following steps to compute the estimated value of the parameters until convergence. The estimated value of the parameter in the  $i$ -th iteration is  $\theta^{(i)} = (\pi^{(i)}, p^{(i)}, q^{(i)})$ . The  $(i+1)^{th}$  iteration of the EM algorithm is as follows:

E-step: Compute the probability that observation  $y_j$  comes from tossing coin B under model parameters  $\pi^{(i)}, p^{(i)}, q^{(i)}$

$$u_j^{(i+1)} = \frac{\pi^{(i)}(p^{(i)})^{y_j}(1-p^{(i)})^{1-y_j}}{\pi^{(i)}(p^{(i)})^{y_j}(1-p^{(i)})^{1-y_j} + (1-\pi^{(i)})(q^{(i)})^{y_j}(1-q^{(i)})^{1-y_j}} \quad (9.5)$$

M-step: Compute the new estimated value of the model parameters

$$\pi^{(i+1)} = \frac{1}{n} \sum_{j=1}^n \mu_j^{(i+1)} \quad (9.6)$$

$$p^{(i+1)} = \frac{\sum_{j=1}^n \mu_j^{(i+1)} y_j}{\sum_{j=1}^n \mu_j^{(i+1)}} \quad (9.7)$$

$$q^{(i+1)} = \frac{\sum_{j=1}^n (1 - \mu_j^{(i+1)}) y_j}{\sum_{j=1}^n (1 - \mu_j^{(i+1)})} \quad (9.8)$$

Conduct the numerical calculation. Assume that the initial value of the model parameter is

$$\pi^{(0)} = 0.5, \quad p^{(0)} = 0.5, \quad q^{(0)} = 0.5$$

According to Eq. (9.5), for both  $y_j = 1$  and  $y_j = 0$ , we have  $\mu_j^{(1)} = 0.5$ . Using the iterative Eqs. (9.6), (9.7) and (9.8), we get

$$\pi^{(1)} = 0.5, \quad p^{(1)} = 0.6, \quad q^{(1)} = 0.6$$

According to Eq. (9.5), we have

$$\mu_j^{(2)} = 0.5, \quad j = 1, 2, \dots, 10$$

Continue the iteration to get

$$\pi^{(2)} = 0.5, \quad p^{(2)} = 0.6, \quad q^{(2)} = 0.6$$

Thus we obtain the maximum likelihood estimation of the model parameter  $\theta$ :

$$\hat{\pi} = 0.5, \quad \hat{p} = 0.6, \quad \hat{q} = 0.6$$

Here,  $\pi = 0.5$  indicates that coin A is uniform, which is easy to understand.

If the initial value takes  $\pi^{(0)} = 0.4$ ,  $p^{(0)} = 0.6$ ,  $q^{(0)} = 0.7$ , then the maximum likelihood estimation of the model parameter will be  $\hat{\pi} = 0.4064$ ,  $\hat{p} = 0.5368$ ,  $\hat{q} = 0.6432$ . It means that the EM algorithm is related to the selection of initial values, i.e., different selection may lead to different estimated values of the parameter.

Generally,  $Y$  represents the data of observed random variables,  $Z$  represents the data of hidden random variables.  $Y$  and  $Z$  together are called the complete-data, and the observation data  $Y$  is also called the incomplete-data. Assuming that the probability distribution of the given observation  $Y$  is  $P(Y|\theta)$ , where  $\theta$  is the model parameter to be estimated, then the likelihood function of the incomplete-data  $Y$  is  $P(Y|\theta)$  and the log-likelihood function is  $L(\theta) = \log P(Y|\theta)$ ; assuming that the joint probability distribution of  $Y$  and  $Z$  is  $P(Y, Z|\theta)$ , then the log-likelihood function of the complete-data is  $\log P(Y, Z|\theta)$ .

The EM algorithm finds the maximum likelihood estimation of  $L(\theta) = \log P(Y|\theta)$  by iterations. Each iteration includes two steps: E-step, to find the expectation, and M-step, to find the maximization. The EM algorithm is described as follows.

### Algorithm 9.1

(EM algorithm) Input: the observed variable data  $Y$ , the hidden variable data  $Z$ , the joint distribution  $P(Y, Z|\theta)$ , the condition distribution  $P(Z|Y, \theta)$ ;

Output: model parameter  $\theta$ .

- (1) Select the initial value  $\theta^{(0)}$  of the parameter and start the iteration;
- (2) E-step: denote  $\theta^{(i)}$  as the estimated value of the parameter  $\theta$  in the  $i$ -th iteration.  
E-step in the  $(i+1)$ th iteration computes

$$\begin{aligned} Q(\theta, \theta^{(i)}) &= E_Z[\log P(Y, Z|\theta)|Y, \theta^{(i)}] \\ &= \sum_Z \log P(Y, Z|\theta) P(Z|Y, \theta^{(i)}) \end{aligned} \quad (9.9)$$

Here,  $P(Z|Y, \theta^{(i)})$  is the conditional probability distribution of the hidden variable data  $Z$  with the given observation data  $Y$  and the current parameter estimation  $\theta^{(i)}$ ;

- (3) M-step: find  $\theta$  that maximizes  $Q(\theta, \theta^{(i)})$  and determine the estimated value  $\theta^{(i+1)}$  of the parameter in the  $(i+1)$ th iteration.

$$\theta^{(i+1)} = \arg \max_{\theta} Q(\theta, \theta^{(i)}) \quad (9.10)$$

- (4) Repeat step (2) and step (3) until convergence.

The function  $Q(\theta, \theta^{(i)})$  in Eq. (9.9) is the core of the EM algorithm and is called the  $Q$  function.

**Definition 9.1** (*Q function*) The expectation of the log-likelihood function  $\log P(Y, Z|\theta)$  of complete-data on the conditional probability distribution  $P(Z|Y, \theta^{(i)})$  of un-observation data  $Z$  under given observation data  $Y$  and the current parameter  $\theta^i$  is called *Q* function, i.e.,

$$Q(\theta, \theta^{(i)}) = E_Z[\log P(Y, Z|\theta)|Y, \theta^{(i)}] \quad (9.11)$$

The following is some explanations about the EM algorithm:

Step (1) The initial values of parameters can be selected arbitrarily, but it should be noted that the EM algorithm is sensitive to the initial values.

Step (2) E-step finds  $Q(\theta, \theta^{(i)})$ . In the *Q* function,  $Z$  is the un-observation data,  $Y$  is the observation data. Note that the first variable of  $Q(\theta, \theta^{(i)})$  represents the parameter to be maximized, and the second variable represents the current estimate of the parameter. Each iteration is actually seeking the *Q* function and its maximum.

Step (3) M-step finds the maximization of  $Q(\theta, \theta^{(i)})$ , gets  $\theta^{(i+1)}$  and finishes one iteration  $\theta^{(i)} \rightarrow \theta^{(i+1)}$ . Later, it will be proved that each iteration makes the likelihood function increase or reach the local maximum.

Step (4) The conditions for stopping iteration are given. Generally, for smaller positive numbers  $\varepsilon_1$  and  $\varepsilon_2$ , the iteration stops if the conditions below are satisfied

$$\|\theta^{(i+1)} - \theta^{(i)}\| < \varepsilon_1 \text{ or } \|Q(\theta^{(i+1)}, \theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)})\| < \varepsilon_2$$

### 9.1.2 Derivation of the EM Algorithm

The EM algorithm is described above. Why can the EM algorithm approximate the maximum likelihood estimation of observation data? Next, the EM algorithm is derived by approximately solving the problem of maximizing the log-likelihood function of the observation data, from which we can clearly see how the EM algorithm works.

We are faced with a probabilistic model with hidden variables, and the goal is to maximize the log-likelihood function of the observation data  $Y$  (incomplete-data) concerning the parameter  $\theta$ , i.e., to maximize

$$\begin{aligned} L(\theta) &= \log P(Y|\theta) = \log \sum_Z P(Y, Z|\theta) \\ &= \log \left( \sum_Z P(Y|Z, \theta) P(Z|\theta) \right) \end{aligned} \quad (9.12)$$

Note that the main difficulty of this maximization is that Eq. (9.12) has un-observation data and the logarithm containing sums (or integrals).

In fact, the EM algorithm approximates to maximize  $L(\theta)$  step by step through iteration. Suppose that the estimated value of  $\theta$  after the  $i$ -th iteration is  $\theta^{(i)}$ . We hope that the new estimation  $\theta$  can increase  $L(\theta)$ , i.e.,  $L(\theta) > L(\theta^{(i)})$ , and gradually reach the maximum. Therefore, consider the difference between

$$L(\theta) - L(\theta^{(i)}) = \log \left( \sum_Z P(Y|Z, \theta) P(Z|\theta) \right) - \log P(Y|\theta^{(i)})$$

Obtain the lower bound using Jensen inequality<sup>1</sup>:

$$\begin{aligned} L(\theta) - L(\theta^{(i)}) &= \log \left( \sum_Z P(Z|Y, \theta^{(i)}) \frac{P(Y|Z, \theta) P(Z|\theta)}{P(Z|Y, \theta^{(i)})} \right) - \log P(Y|\theta^{(i)}) \\ &\geq \sum_Z P(Z|Y, \theta^{(i)}) \log \frac{P(Y|Z, \theta) P(Z|\theta)}{P(Z|Y, \theta^{(i)})} - \log P(Y|\theta^{(i)}) \\ &= \sum_Z P(Z|Y, \theta^{(i)}) \log \frac{P(Y|Z, \theta) P(Z|\theta)}{P(Z|Y, \theta^{(i)}) P(Y|\theta^{(i)})} \end{aligned}$$

Let

$$B(\theta, \theta^{(i)}) \stackrel{\Delta}{=} L(\theta^{(i)}) + \sum_Z P(Z|Y, \theta^{(i)}) \log \frac{P(Y|Z, \theta) P(Z|\theta)}{P(Z|Y, \theta^{(i)}) P(Y|\theta^{(i)})} \quad (9.13)$$

Then

$$L(\theta) \geq B(\theta, \theta^{(i)}) \quad (9.14)$$

i.e., the function  $B(\theta, \theta^{(i)})$  is a lower bound for  $L(\theta)$ , and by Eq. (9.13), we have

$$L(\theta^{(i)}) = B(\theta^{(i)}, \theta^{(i)}) \quad (9.15)$$

Therefore, any  $\theta$  that can increase  $B(\theta, \theta^{(i)})$  can also increase  $L(\theta)$ . To increase  $L(\theta)$  as much as possible,  $\theta^{(i+1)}$  is chosen so that  $B(\theta, \theta^{(i)})$  reaches the maximum, i.e.,

$$\theta^{(i+1)} = \arg \max_{\theta} B(\theta, \theta^{(i)}) \quad (9.16)$$

Now find the expression for  $\theta^{(i+1)}$ . Omitting the term that is constant for the maximization of  $\theta$ , from Eqs. (9.16), (9.13), and (9.10), we have

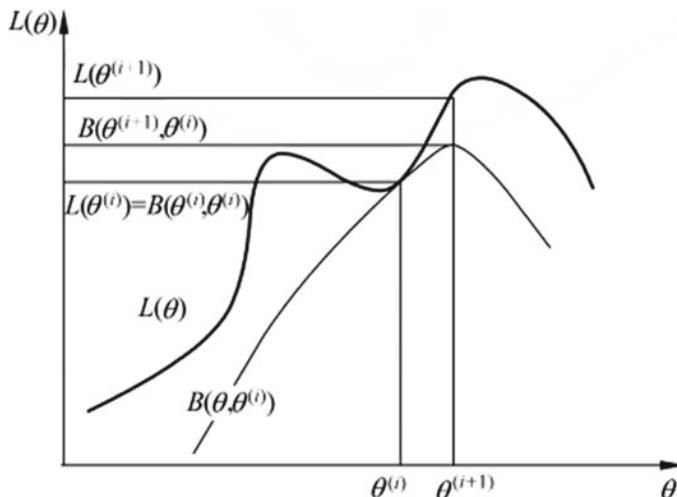
---

<sup>1</sup> Here,  $\log \sum_j \lambda_j y_j \geq \sum_j \lambda_j \log y_j$  is used, where  $\lambda_j \geq 0$ ,  $\sum_j \lambda_j = 1$ .

$$\begin{aligned}
\theta^{(i+1)} &= \arg \max_{\theta} \left( L(\theta^{(i)}) + \sum_Z P(Z|Y, \theta^{(i)}) \log \frac{P(Y|Z, \theta) P(Z|\theta)}{P(Z|Y, \theta^{(i)}) P(Y|\theta^{(i)})} \right) \\
&= \arg \max_{\theta} \left( \sum_Z P(Z|Y, \theta^{(i)}) \log (P(Y|Z, \theta) P(Z|\theta)) \right) \\
&= \arg \max_{\theta} \left( \sum_Z P(Z|Y, \theta^{(i)}) \log P(Y|Z, \theta) \right) \\
&= \arg \max_{\theta} Q(\theta, \theta^{(i)}) \tag{9.17}
\end{aligned}$$

Equation (9.17) is equivalent to one iteration of the EM algorithm, i.e., to find the  $Q$  function and its maximization. The EM algorithm is an algorithm that solves for the maximization of the log-likelihood function by continuously solving the lower bound of the maximization approximation.

A visual interpretation of the EM algorithm is given in Fig. 9.1. The upper curve is  $L(\theta)$  and the lower curve is  $B(\theta, \theta^{(i)})$ . From Eq. (9.14),  $B(\theta, \theta^{(i)})$  is the lower bound of the log-likelihood function  $L(\theta)$ . By Eq. (9.15), the two functions are equal at the point  $\theta = \theta^{(i)}$ . From Eqs. (9.16) and (9.17), the EM algorithm finds the next point  $\theta^{(i+1)}$  that maximizes the function  $B(\theta, \theta^{(i)})$  and also maximizes the function  $Q(\theta, \theta^{(i)})$ . At this point, since  $L(\theta) \geq B(\theta, \theta^{(i)})$ , the increase in the function  $B(\theta, \theta^{(i)})$  ensures that the log-likelihood function  $L(\theta)$  is also increasing in each iteration. The EM algorithm recalculates the  $Q$  function value at the point  $\theta^{(i+1)}$  for the next iteration. In this process, the log-likelihood function  $L(\theta)$  keeps increasing. It can be inferred from the figure that the EM algorithm is not guaranteed to find the global optimum.



**Fig. 9.1** Explanation of EM algorithm

From Eq. (9.15), the two functions are equal at the point  $\theta = \theta^{(i)}$ . From Eq. (9.16) and Eq. (9.17), the EM algorithm finds the next point  $\theta^{(i+1)}$  to maximize the function  $B(\theta, \theta^{(i)})$  and also maximize the function  $Q(\theta, \theta^{(i)})$ . At this time, since  $L(\theta) \geq B(\theta, \theta^{(i)})$ , the function  $B(\theta, \theta^{(i)})$  increases, ensuring that the log-likelihood function  $L(\theta)$  also increases in each iteration. The EM algorithm recalculates the value of the  $Q$  function at the point  $\theta^{(i+1)}$  and proceeds to the next iteration. In this process, the log-likelihood function  $L(\theta)$  keeps increasing. It can be inferred from the figure that the EM algorithm cannot guarantee to find the global optimal value.

### 9.1.3 Application of the EM Algorithm in Unsupervised Learning

Supervised learning is the learning of conditional probability distribution  $P(Y|X)$  from training data  $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$  or using decision function  $Y = f(X)$  as a model. It is applied for tasks such as classification, regression, and labeling. At this time, each sample point in the training data is composed of input and output pairs.

Sometimes the training data has only inputs without corresponding outputs  $\{(x_1, \cdot), (x_2, \cdot), \dots, (x_N, \cdot)\}$ , and learning a model from such data is called an unsupervised learning problem. The EM algorithm can be used for unsupervised learning of the generative model. The generative model is represented by the joint probability distribution  $P(X, Y)$ , and the unsupervised learning training data can be considered as data generated by the joint probability distribution.  $X$  is the observation data, and  $Y$  is the un-observation data.

## 9.2 The Convergence of the EM Algorithm

The EM algorithm provides a method to approximate the maximum likelihood estimation of a probabilistic model containing hidden variables. The greatest advantage of the EM algorithm is its simplicity and generalizability. It is natural to ask: Does the estimated sequence obtained by the EM algorithm converge? If so, does it converge to the global maximum or the local maximum? Two theorems on the convergence of the EM algorithm are given below.

**Theorem 9.1** Let  $P(Y|\theta)$  be the likelihood function of the observation data,  $\theta^{(i)} (i = 1, 2, \dots)$  be the sequence of the parameter estimation obtained by the EM algorithm, and  $P(Y|\theta^{(i)}) (i = 1, 2, \dots)$  be the corresponding likelihood function sequence, then  $P(Y|\theta^{(i)})$  is monotonically increasing, i.e.,

$$P(Y|\theta^{(i+1)}) \geq P(Y|\theta^{(i)}) \quad (9.18)$$

**Proof** Since

$$P(Y|\theta) = \frac{P(Y, Z|\theta)}{P(Z|Y, \theta)}$$

Taking the logarithm, we have

$$\log P(Y|\theta) = \log P(Y, Z|\theta) - \log P(Z|Y, \theta)$$

According to Eq. (9.11)

$$Q(\theta, \theta^{(i)}) = \sum_Z \log P(Y, Z|\theta) P(Z|Y, \theta^{(i)})$$

Let

$$H(\theta, \theta^{(i)}) = \sum_Z \log P(Z|Y, \theta) P(Z|Y, \theta^{(i)}) \quad (9.19)$$

Then the log-likelihood function can be written as

$$\log P(Y|\theta) = Q(\theta, \theta^{(i)}) - H(\theta, \theta^{(i)}) \quad (9.20)$$

Take  $\theta$  as  $\theta^{(i)}$  and  $\theta^{(i+1)}$  in Eq. (9.20) and subtract them, then there is

$$\begin{aligned} & \log P(Y|\theta^{(i+1)}) - \log P(Y|\theta^{(i)}) \\ &= [Q(\theta^{(i+1)}, \theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)})] \\ &\quad - [H(\theta^{(i+1)}, \theta^{(i)}) - H(\theta^{(i)}, \theta^{(i)})] \end{aligned} \quad (9.21)$$

To prove Eq. (9.18), just prove that the right end of the Eq. (9.21) is non-negative. Regarding the first term at the right end of Eq. (9.21), since  $\theta^{(i+1)}$  makes  $Q(\theta, \theta^{(i)})$  maximal, we have

$$Q(\theta^{(i+1)}, \theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)}) \geq 0 \quad (9.22)$$

The second term can be obtained by Eq. (9.19):

$$\begin{aligned} H(\theta^{(i+1)}, \theta^{(i)}) - H(\theta^{(i)}, \theta^{(i)}) &= \sum_Z (\log \frac{P(Z|Y, \theta^{(i+1)})}{P(Z|Y, \theta^{(i)})}) P(Z|Y, \theta^{(i)}) \\ &\leq \log \left( \sum_Z \frac{P(Z|Y, \theta^{(i+1)})}{P(Z|Y, \theta^{(i)})} P(Z|Y, \theta^{(i)}) \right) \\ &= \log \left( \sum_Z P(Z|Y, \theta^{(i+1)}) \right) = 0 \end{aligned} \quad (9.23)$$

The inequality sign here is derived from Jensen's inequality.

From Eq. (9.22) and Eq. (9.23), we know that the right end of Eq. (9.21) is non-negative.

**Theorem 9.2** Let  $L(\theta) = \log P(Y|\theta)$  be the log-likelihood function of the observation data,  $\theta^{(i)}(i = 1, 2, \dots)$  be the parameter estimation sequence obtained by the EM algorithm, and  $L(\theta^{(i)})(i = 1, 2, \dots)$  be the corresponding log-likelihood function sequence.

- (1) If  $P(Y|\theta)$  has an upper bound, then  $L(\theta^{(i)}) = \log P(Y|\theta^{(i)})$  converges to a certain value  $L^*$ ;
- (2) When the functions  $Q(\theta, \theta^{(i)})$  and  $L(\theta)$  satisfy certain conditions, the convergence value  $\theta^*$  of the parameter estimation sequence  $\theta^{(i)}$  obtained by the EM algorithm is the stable point of  $L(\theta)$ .

### Proof

- (1) is obtained immediately by the monotonicity of  $L(\theta) = \log P(Y|\theta^{(i)})$  and the boundedness of  $P(Y|\theta)$ .

The proof of (2) is omitted. Please refer to the literature [5].

The conditions for functions  $Q(\theta, \theta^{(i)})$  and  $L(\theta)$  in Theorem 9.2 are satisfied in most cases. The convergence of the EM algorithm includes the convergence concerning the log-likelihood function sequence  $L(\theta^{(i)})$  and the convergence for the parameter estimation sequence  $\theta^{(i)}$ . The former does not contain the latter. In addition, the theorem can only guarantee that the parameter estimation sequence converges to the stable point of the log-likelihood function sequence, and cannot guarantee the convergence to the maximum point. Therefore, in applications, the selection of the initial value becomes very important. The commonly used method is to select several different initial values for iteration, and then compare the estimated values obtained to select the best one.

## 9.3 Application of the EM Algorithm in the Learning of the Gaussian Mixture Model

An important application of the EM algorithm is the parameter estimation of Gaussian mixture model. The Gaussian mixture model is widely used. In many cases, the EM algorithm is an effective method for learning the Gaussian mixture model.

### 9.3.1 Gaussian Mixture Model

**Definition 9.2** (*Gaussian mixture model*) The Gaussian Mixture Model refers to a probability distribution model with the following form:

$$P(y|\theta) = \sum_{k=1}^K \alpha_k \phi(y|\theta_k) \quad (9.24)$$

where  $\alpha_k$  is the coefficient,  $\alpha_k \geq 0$ ,  $\sum_{k=1}^K \alpha_k = 1$ ;  $\phi(y|\theta_k)$  is the Gaussian distribution density,  $\theta_k = (\mu_k, \sigma_k^2)$ , and

$$\phi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(y - \mu_k)^2}{2\sigma_k^2}\right) \quad (9.25)$$

is called the  $k$ -th sub-model.

The general mixture model can replace the Gaussian distribution density in Eq. (9.25) with any probability distribution density, we only introduce the most commonly used Gaussian mixture model.

### 9.3.2 The EM Algorithm for Parameter Estimation of the Gaussian Mixture Model

Assuming that the observation data  $y_1, y_2, \dots, y_N$  is generated by the Gaussian mixture model,

$$P(y|\theta) = \sum_{k=1}^K \alpha_k \phi(y|\theta_k) \quad (9.26)$$

Among them,  $\theta = (\alpha_1, \alpha_2, \dots, \alpha_K; \theta_1, \theta_2, \dots, \theta_K)$ . We use the EM algorithm to estimate the parameters  $\theta$  of the Gaussian mixture model.

- (1) Identify hidden variables and write out the log-likelihood function of the complete data

It can be imagined that the observation data  $y_j, j = 1, 2, \dots, N$  is generated as follows: first select the  $k$ -th Gaussian distribution sub-model  $\phi(y_j|\theta_k)$  according to the probability  $\alpha_k$ , and then according to the  $k$ -th sub-model The probability distribution  $\phi(y_j|\theta_k)$  generates observation data  $y_j$ . At this time, the observation data  $y_j, j = 1, 2, \dots, N$  are known; the data reflecting the observation data  $y_j$  from the  $k$ -th sub-model is unknown,  $k = 1, 2, \dots, K$ , Expressed by the hidden variable  $\gamma_{jk}$ , which is defined as follows:

$$\gamma_{jk} = \begin{cases} 1, & \text{The } j\text{th observation comes from the } k\text{th submodel} \\ 0, & \text{otherwise} \end{cases} \quad j = 1, 2, \dots, N; \quad k = 1, 2, \dots, K \quad (9.27)$$

$\gamma_{jk}$  is a 0–1 random variable.

With the observation data  $y_j$  and the unobserved data  $\gamma_{jk}$ , then the complete data is

$$(y_j, \gamma_{j1}, \gamma_{j2}, \dots, \gamma_{jK}), \quad j = 1, 2, \dots, N$$

Thus, the likelihood function of complete data can be written:

$$\begin{aligned} P(y, \gamma | \theta) &= \prod_{j=1}^N P(y_j, \gamma_{j1}, \gamma_{j2}, \dots, \gamma_{jK} | \theta) \\ &= \prod_{k=1}^K \prod_{j=1}^N [\alpha_k \phi(y_j | \theta_k)]^{\gamma_{jk}} \\ &= \prod_{k=1}^K \alpha_k^{n_k} \prod_{j=1}^N [\phi(y_j | \theta_k)]^{\gamma_{jk}} \\ &= \prod_{k=1}^K \alpha_k^{n_k} \prod_{j=1}^N \left[ \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(y_j - \mu_k)^2}{2\sigma_k^2}\right) \right]^{\gamma_{jk}} \end{aligned}$$

$$\text{where } n_k = \sum_{j=1}^N \gamma_{jk}, \quad \sum_{k=1}^K n_k = N.$$

Then, the log-likelihood function of the complete data is

$$\log P(y, \gamma | \theta) = \sum_{k=1}^K \left\{ n_k \log \alpha_k + \sum_{j=1}^N \gamma_{jk} \left[ \log\left(\frac{1}{\sqrt{2\pi}}\right) - \log \sigma_k - \frac{1}{2\sigma_k^2} (y_j - \mu_k)^2 \right] \right\}$$

(2) E-step of the EM algorithm: Determine the Q function

$$\begin{aligned} Q(\theta, \theta^{(i)}) &= E[\log P(y, \gamma | \theta) | y, \theta^{(i)}] \\ &= E \left\{ \sum_{k=1}^K \left\{ n_k \log \alpha_k + \sum_{j=1}^N \gamma_{jk} \left[ \log\left(\frac{1}{\sqrt{2\pi}}\right) - \log \sigma_k - \frac{1}{2\sigma_k^2} (y_j - \mu_k)^2 \right] \right\} \right\} \\ &= \sum_{k=1}^K \left\{ \sum_{j=1}^N (E\gamma_{jk}) \log \alpha_k + \sum_{j=1}^N (E\gamma_{jk}) \left[ \log\left(\frac{1}{\sqrt{2\pi}}\right) - \log \sigma_k - \frac{1}{2\sigma_k^2} (y_j - \mu_k)^2 \right] \right\} \quad (9.28) \end{aligned}$$

Here you need to calculate  $E(\gamma_{jk}|y, \theta)$ , which is denoted as  $\hat{\gamma}_{jk}$ .

$$\begin{aligned}\hat{\gamma}_{jk} &= E(\gamma_{jk}|y, \theta) = P(\gamma_{jk} = 1|y, \theta) \\ &= \frac{P(\gamma_{jk} = 1, y_j|\theta)}{\sum_{k=1}^K P(\gamma_{jk} = 1, y_j|\theta)} \\ &= \frac{P(y_j|\gamma_{jk} = 1, \theta)P(\gamma_{jk} = 1, \theta)}{\sum_{k=1}^K P(y_j|\gamma_{jk} = 1, \theta)P(\gamma_{jk} = 1, \theta)} \\ &= \frac{\alpha_k \phi(y_j|\theta_k)}{\sum_{k=1}^K \alpha_k \phi(y_j|\theta_k)}, \quad j = 1, 2, \dots, N; \quad k = 1, 2, \dots, K\end{aligned}$$

$\hat{\gamma}_{jk}$  is the probability that the  $j$ -th observation data comes from the  $k$ -th sub-model under the current model parameters, which is called the response of the sub-model  $k$  to the observation data  $y_j$ .

Substituting  $\hat{\gamma}_{jk} = E\gamma_{jk}$  and  $n_k = \sum_{j=1}^N E\gamma_{jk}$  into Eq. (9.28), we get

$$Q(\theta, \theta^{(i)}) = \sum_{k=1}^K \left\{ n_k \log \alpha_k + \sum_{j=1}^N \hat{\gamma}_{jk} \left[ \log \left( \frac{1}{\sqrt{2\pi}} \right) - \log \sigma_k - \frac{1}{2\sigma_k^2} (y_j - \mu_k)^2 \right] \right\} \quad (9.29)$$

### (3) Determine the M step of the EM algorithm

The M step of the iteration is to find the maximum value of the function  $Q(\theta, \theta^{(i)})$  to  $\theta$ , that is, to find the model parameters of the new iteration:

$$\theta^{(i+1)} = \arg \max_{\theta} Q(\theta, \theta^{(i)})$$

Use  $\hat{\mu}_k$ ,  $\hat{\sigma}_k^2$  and  $\hat{\alpha}_k$ ,  $k = 1, 2, \dots, K$  to represent the parameters of  $\theta^{(i+1)}$ . To find  $\hat{\mu}_k$  and  $\hat{\sigma}_k^2$ , you only need to calculate the partial derivative on Eq. (9.29) of  $\mu_k$  and  $\sigma_k^2$  respectively and set it to 0 to obtain  $\hat{\mu}_k$  and  $\hat{\sigma}_k^2$ ; To find  $\hat{\alpha}_k$  is to obtain the partial derivative under the condition of  $\sum_{k=1}^K \alpha_k = 1$  and set it to 0. The results are as follows:

$$\hat{\mu}_k = \frac{\sum_{j=1}^N \hat{\gamma}_{jk} y_j}{\sum_{j=1}^N \hat{\gamma}_{jk}}, \quad k = 1, 2, \dots, K \quad (9.30)$$

$$\hat{\sigma}_k^2 = \frac{\sum_{j=1}^N \hat{\gamma}_{jk} (y_j - \mu_k)^2}{\sum_{j=1}^N \hat{\gamma}_{jk}}, \quad k = 1, 2, \dots, K \quad (9.31)$$

$$\hat{\alpha}_k = \frac{n_k}{N} = \frac{\sum_{j=1}^N \hat{\gamma}_{jk}}{N}, \quad k = 1, 2, \dots, K \quad (9.32)$$

Repeat the above calculations until the value of the log-likelihood function no longer changes significantly.

The EM algorithm for estimating the parameters of the Gaussian mixture model is summarized as follows.

**Algorithm 9.2** (*the EM algorithm for parameter estimation of Gaussian mixture model*) Input: Observation data  $y_1, y_2, \dots, y_N$ , Gaussian mixture model;

Output: Gaussian mixture model parameters.

- (1) Take the initial value of the parameter and start the iteration;
- (2) E-step: According to the current model parameters, calculate the response of sub-model  $k$  to the observation data  $y_j$

$$\hat{\gamma}_{jk} = \frac{\alpha_k \phi(y_j | \theta_k)}{\sum_{k=1}^K \alpha_k \phi(y_j | \theta_k)}, \quad j = 1, 2, \dots, N; \quad k = 1, 2, \dots, K$$

- (3) M-step: Calculate the model parameters of the new iteration

$$\begin{aligned}\hat{\mu}_k &= \frac{\sum_{j=1}^N \hat{\gamma}_{jk} y_j}{\sum_{j=1}^N \hat{\gamma}_{jk}}, \quad k = 1, 2, \dots, K \\ \hat{\sigma}_k^2 &= \frac{\sum_{j=1}^N \hat{\gamma}_{jk} (y_j - \hat{\mu}_k)^2}{\sum_{j=1}^N \hat{\gamma}_{jk}}, \quad k = 1, 2, \dots, K \\ \hat{\alpha}_k &= \frac{\sum_{j=1}^N \hat{\gamma}_{jk}}{N}, \quad k = 1, 2, \dots, K\end{aligned}$$

- (4) Repeat steps (2) and (3) until convergence.

## 9.4 Extensions of the EM Algorithm

The EM algorithm can also be interpreted as the maximization-maximization algorithm of the  $F$  function, based on this interpretation, there are several variations and extensions, such as the generalized expectation maximization (GEM) algorithm. It will be introduced below.

### 9.4.1 The Maximization-Maximization Algorithm of F-Function

First, we introduce the  $F$  function and discuss its properties.

**Definition 9.3** (*F function*) Assuming that the probability distribution of the hidden variable data  $Z$  is  $\tilde{P}(Z)$ , define the function  $F(\tilde{P}, \theta)$  of the distribution  $\tilde{P}$  and the parameter  $\theta$  as follows:

$$F(\tilde{P}, \theta) = E_{\tilde{P}}[\log P(Y, Z|\theta)] + H(\tilde{P}) \quad (9.33)$$

which is called the *F* function. Here,  $H(\tilde{P}) = -E_{\tilde{P}} \log \tilde{P}(Z)$  is the entropy of the distribution  $\tilde{P}(Z)$ .

In Definition 9.3, it is usually assumed that  $P(Y, Z|\theta)$  is a continuous function of  $\theta$ , so  $F(\tilde{P}, \theta)$  is a continuous function of  $\tilde{P}$  and  $\theta$ . The function  $F(\tilde{P}, \theta)$  has the following important properties.

**Lemma 9.1** For a fixed  $\theta$ , there is a unique distribution  $\tilde{P}_\theta$  that maximizes  $F(\tilde{P}, \theta)$ . At this point,  $\tilde{P}_\theta$  is given by:

$$\tilde{P}_\theta = P(Z|Y, \theta) \quad (9.34)$$

And  $\tilde{P}_\theta$  changes continuously with  $\theta$ .

**Proof** For a fixed  $\theta$ ,  $F(\tilde{P}, \theta)$  can be obtained to achieve a maximum distribution  $\tilde{P}_\theta(Z)$ . To this end, the Lagrangian multiplier  $\lambda$  is introduced, and the Lagrangian function is

$$L = E_{\tilde{P}} \log P(Y, Z|\theta) - E_{\tilde{P}} \log \tilde{P}(Z) + \lambda \left( 1 - \sum_Z \tilde{P}(Z) \right) \quad (9.35)$$

Find the partial derivative with respect to  $\tilde{P}$ :

$$\frac{\partial L}{\partial \tilde{P}(Z)} = \log P(Y, Z|\theta) - \log \tilde{P}(Z) - 1 - \lambda$$

Let the partial derivative equal to 0, we get

$$\lambda = \log P(Y, Z|\theta) - \log \tilde{P}_\theta(Z) - 1$$

It follows that  $\tilde{P}_\theta(Z)$  is proportional to  $P(Y, Z|\theta)$

$$\frac{P(Y, Z|\theta)}{\tilde{P}_\theta(Z)} = e^{1+\lambda}$$

Then Eq. (9.34) is obtained from the constraints  $\sum_Z \tilde{P}(Z)$ .

Assuming that  $P(Y, Z|\theta)$  is a continuous function of  $\theta$ , we get that  $\tilde{P}_\theta$  is a continuous function of  $\theta$ .

**Lemma 9.2** If  $\tilde{P}_\theta = P(Z|Y, \theta)$ , then

$$F(\tilde{P}, \theta) = \log P(Y|\theta) \quad (9.36)$$

The proof is left as an exercise for the reader.

From the above lemma, we can get the explanation of the maximization-maximization algorithm of the  $F$  function used in the EM algorithm.

**Theorem 9.3** Suppose that  $L(\theta) = \log P(Y|\theta)$  is the log-likelihood function of the observation data,  $\theta^{(i)}$ ,  $i = 1, 2, \dots$ , is the parameter estimation sequence obtained by the EM algorithm, and the function  $F(\tilde{P}, \theta)$  is defined by Eq. (9.33). If  $F(\tilde{P}, \theta)$  has a local maximum at  $\tilde{P}^*$  and  $\theta^*$ , then  $L(\theta)$  also has a local maximum at  $\theta^*$ . Similarly, if  $F(\tilde{P}, \theta)$  reaches the global maximum at  $\tilde{P}^*$  and  $\theta^*$ , then  $L(\theta)$  also reaches the global maximum at  $\theta^*$ .

**Proof** According to Lemma 9.1 and Lemma 9.2,  $L(\theta) = \log P(Y|\theta) = F(\tilde{P}_\theta, \theta)$  holds for any  $\theta$ . In particular, for the parameter  $\theta^*$  that maximizes  $F(\tilde{P}, \theta)$ , we have

$$L(\theta^*) = F(\tilde{P}_{\theta^*}, \theta^*) = F(\tilde{P}^*, \theta^*) \quad (9.37)$$

To prove that  $\theta^*$  is the maximum point of  $L(\theta)$ , it is necessary to prove that there is no point  $\theta^{**}$  close to  $\theta^*$  such that  $L(\theta^{**}) > L(\theta^*)$ . If there is such a point  $\theta^{**}$ , then there should be  $F(\tilde{P}^{**}, \theta^{**}) > F(\tilde{P}^*, \theta^*)$ , where  $\tilde{P}^{**} = \tilde{P}_{\theta^{**}}$ . However, since  $\tilde{P}_\theta$  is continuously varying  $\theta$ ,  $\tilde{P}^{**}$  should be close to  $\tilde{P}^*$ , which contradicts the assumption that  $\tilde{P}^*$  and  $\theta^*$  are local maxima of  $F(\tilde{P}, \theta)$ .

Similarly, we prove the conclusion about the global maximum. •

**Theorem 9.4** One iteration of the EM algorithm can be realized by the maximization-maximization algorithm of the  $F$  function.

Let  $\theta^{(i)}$  be the estimate of the parameter  $\theta$  of the  $i$ -th iteration,  $\tilde{P}^{(i)}$  be the estimate of the function  $\tilde{P}$  of the  $i$ -th iteration. The two steps in the  $(i+1)$ th iteration are

- (1) For a fixed  $\theta^{(i)}$ , find  $\tilde{P}^{(i+1)}$  to maximize  $F(\tilde{P}, \theta^{(i)})$ ;
- (2) For a fixed  $\tilde{P}^{(i+1)}$ , find  $\theta^{(i+1)}$  to maximize  $F(\tilde{P}^{(i+1)}, \theta)$ .

**Proof**

- (1) By Lemma 9.1, for a fixed  $\theta^{(i)}$ ,

$$\tilde{P}^{(i+1)}(Z) = \tilde{P}_{\theta^{(i)}}(Z) = P(Z|Y, \theta^{(i)})$$

maximizes  $F(\tilde{P}, \theta^{(i)})$ . At this time,

$$\begin{aligned} F(\tilde{P}^{(i+1)}, \theta) &= E_{\tilde{P}^{(i+1)}}[\log P(Y, Z|\theta)] + H(\tilde{P}^{(i+1)}) \\ &= \sum_Z \log P(Y, Z|\theta) P(Z|Y, \theta^{(i)}) + H(\tilde{P}^{(i+1)}) \end{aligned}$$

From the definition Eq. (9.11) of  $Q(\theta, \theta^{(i)})$ , we have

$$F(\tilde{P}^{(i+1)}, \theta) = Q(\theta, \theta^{(i)}) + H(\tilde{P}^{(i+1)})$$

(2) Fix  $\tilde{P}^{(i+1)}$  and find  $\theta^{(i+1)}$  to maximize  $F(\tilde{P}^{(i+1)}, \theta)$ . We obtain

$$\theta^{(i+1)} = \arg \max_{\theta} F(\tilde{P}^{(i+1)}, \theta) = \arg \max_{\theta} Q(\theta, \theta^{(i)})$$

and complete an iteration of the EM algorithm through the above two steps. It can be seen that the parameter estimation sequence  $\theta^{(i)}$ ,  $i = 1, 2, \dots$ , obtained by the EM algorithm is consistent with the one obtained by the maximization-maximization algorithm of the  $F$  function.

In this way, there is the extension of the EM algorithm. •

#### 9.4.2 GEM Algorithm

**Algorithm 9.3 (GEM Algorithm 1)** Input: observation data,  $F$  function;

Output: model parameters.

- (1) Initialize the parameter  $\theta^{(0)}$  and start iterating;
- (2) The  $i+1$ -th iteration, step 1: mark  $\theta^{(i)}$  as the estimated value of the parameter  $\theta$ ,  $\tilde{P}^{(i)}$  is the estimate of the function  $P$ , find  $\tilde{P}^{(i+1)}$  to maximize  $\tilde{P}$  by  $F(\tilde{P}, \theta^{(i)})$ ;
- (3) Step 2: Find  $\theta^{(i+1)}$  to maximize  $F(\tilde{P}^{(i+1)}, \theta)$ ;
- (4) Repeat (2) and (3) until convergence.

In GEM Algorithm 1, sometimes it is very difficult to maximize  $Q(\theta, \theta^{(i)})$ . The following GEM algorithm 2 and GEM algorithm 3 do not directly find  $\theta^{(i+1)}$  to make  $Q(\theta, \theta^{(i)})$  reach the maximum  $\theta$ , but to find a  $\theta^{(i+1)}$  to make  $Q(\theta^{(i+1)}, \theta^{(i)}) > Q(\theta^{(i)}, \theta^{(i)})$ .

**Algorithm 9.4 (GEM Algorithm 2)** Input: Observation data and function  $Q$ ;

Output: model parameters.

- (1) Initialize the parameter  $\theta^{(0)}$ , and start the iteration;
- (2) The  $i+1$ -th iteration, step 1: mark  $\theta^{(i)}$  as the estimated value of the parameter  $\theta$ , and calculate

$$\begin{aligned} Q(\theta, \theta^{(i)}) &= E_Z[\log P(Y, Z|\theta)|Y, \theta^{(i)}] \\ &= \sum_Z P(Z|Y, \theta^{(i)}) \log P(Y, Z|\theta) \end{aligned}$$

(3) Step 2: Find  $\theta^{(i+1)}$  to make

$$Q(\theta^{(i+1)}, \theta^{(i)}) > Q(\theta^{(i)}, \theta^{(i)})$$

(4) Repeat (2) and (3) until convergence.

When the dimension of the parameter  $\theta$  is  $d$  ( $d \geq 2$ ), a special GEM algorithm can be used, which decomposes the M steps of the EM algorithm into  $d$  conditional maximizations, and only changes one component of the parameter vector each time. The remaining components remain unchanged.

**Algorithm 9.5 (GEM Algorithm 3)** Input: observation data and function  $Q$ ;

Output: model parameters.

- (1) Initialize the parameter  $\theta^{(0)} = (\theta_1^{(0)}, \theta_2^{(0)}, \dots, \theta_d^{(0)})$ , and start the iteration;
- (2) The  $i + 1$ -th iteration, step 1: mark  $\theta^{(i)} = (\theta_1^{(i)}, \theta_2^{(i)}, \dots, \theta_d^{(i)})$  as the estimated value of the parameter  $\theta = (\theta_1, \theta_2, \dots, \theta_d)$ , then calculate

$$\begin{aligned} Q(\theta, \theta^{(i)}) &= E_z[\log P(Y, Z|\theta)|Y, \theta^{(i)}] \\ &= \sum_Z P(Z|y, \theta^{(i)}) \log P(Y, Z|\theta) \end{aligned}$$

(3) Step 2: Perform  $d$  conditions maximization:

First, under the condition that  $\theta_2^{(i)}, \dots, \theta_d^{(i)}$  remains unchanged, find that  $Q(\theta, \theta^{(i)})$  reaches the maximum  $\theta_1^{(i+1)}$ ;

Then, under the conditions of  $\theta_1 = \theta_1^{(i+1)}$ ,  $\theta_j = \theta_j^{(i)}$ ,  $j = 3, 4, \dots, d$ , find  $Q(\theta, \theta^{(i)})$  to reach the maximum  $\theta_2^{(i+1)}$ ;

Continue like this, after  $d$  times of conditional maximization, we get  $\theta^{(i+1)} = (\theta_1^{(i+1)}, \theta_2^{(i+1)}, \dots, \theta_d^{(i+1)})$  to make

$$Q(\theta^{(i+1)}, \theta^{(i)}) > Q(\theta^{(i)}, \theta^{(i)})$$

(4) Repeat (2) and (3) until convergence. •

## 9.5 Summary

1. The EM algorithm is an iterative algorithm for the maximum likelihood estimation or the maximum posteriori estimation of probabilistic models containing hidden variables. The data of the probabilistic model with hidden variables is expressed as  $P(Y, Z|\theta)$ . Here,  $Y$  is the data of the observed variable,  $Z$  is the data of the hidden variable, and  $\theta$  is the model parameter. The EM algorithm achieves maximum likelihood estimation by iteratively solving the maximum of

the log-likelihood function  $L(\theta) = \log P(Y|\theta)$  of the observation data. Each iteration consists of two steps: E-step, finding the expectation, i.e., finding the  $\log P(Y, Z|\theta)$  expectation of  $P(Y, Z|\theta^{(i)})$ :

$$Q(\theta, \theta^{(i)}) = \sum_Z \log P(Y, Z|\theta) P(Z|Y, \theta^{(i)})$$

is called the  $Q$  function, where  $\theta^{(i)}$  is the current estimated value of the parameter; M-step, seeking the maximum, i.e., maximizing the  $Q$  function to obtain the new estimated value of the parameter:

$$\theta^{(i+1)} = \arg \max_{\theta} Q(\theta, \theta^{(i)})$$

When constructing a specific EM algorithm, it is important to define the  $Q$  function. In each iteration, the EM algorithm increases the log-likelihood function  $L(\theta)$  by maximizing the  $Q$  function.

2. The EM algorithm increases the likelihood function value of the observation data after each iteration, i.e.,

$$P(Y|\theta^{(i+1)}) \geq P(Y|\theta^{(i)})$$

Under general conditions, the EM algorithm is convergent, its convergence to the global optimum is not guaranteed.

3. The EM algorithm has a wide application. It is mainly used in the learning of probabilistic models containing hidden variables. The parameter estimation of the Gaussian mixture model is an important application of the EM algorithm. The unsupervised learning of the Hidden Markov Model (HMM) that will be introduced in the next chapter is also an important application of the EM algorithm.
4. The EM algorithm can also be interpreted as the maximization-maximization algorithm of the  $F$  function. There are many variations of the EM algorithm, such as the GEM algorithm, which is characterized by increasing the value of the  $F$  function (not necessarily maximizing the  $F$  function) with each iteration, thus increasing the value of the likelihood function.

## 9.6 Further Reading

The EM algorithm was proposed by Dempster et al. [1] Similar algorithms have been proposed before, such as the Baum-Welch algorithm, but they are not as extensive as the EM algorithm. The introduction of the EM algorithm can be found in the literature [2–4]. Proof of the convergence theorem of the EM algorithm can be found in the literature [5]. GEM was proposed by Neal and Hinton [6].

## 9.7 Exercises

- 9.1 See the three-coin model in Example 9.1. Assuming that the observation data remains unchanged, try to select different initial values, e.g.,  $\pi^{(0)} = 0.46$ ,  $p^{(0)} = 0.55$ ,  $q^{(0)} = 0.67$ , and find the maximum likelihood estimation of the model parameters  $\theta = (\pi, p, q)$ .
- 9.2 Prove Lemma 9.2.
- 9.3 With the known observation data  $-67, -48, 6, 8, 14, 16, 23, 24, 28, 29, 41, 49, 56, 60, 75$ , try to estimate the 5 parameters of the two-component Gaussian mixture model.
- 9.4 The EM algorithm can be used for the unsupervised learning with the Naive Bayes method. Try to write its algorithm.

## References

1. Dempster AP, Laird NM, Rubin DB. Maximum-likelihood from incomplete data via the EM algorithm. *J Royal Statistic Soc (Ser B)*. 1977;39(1):1–38.
2. Hastie T, Tibshirani R, Friedman JH. The elements of statistical learning: data mining, inference, and prediction. Springer; 2001.
3. McLachlan G, Krishnan T. The EM algorithm and extensions. New York: John Wiley & Sons; 1996.
4. Mao SS, Wang JJ, Pu XL. Higher Mathematical Statistics. Beijing: Higher Education Press; Heidelberg: Springer Press; 1998.
5. Wu CFJ. On the convergence properties of the EM algorithm. *Ann Stat*. 1983;11(1):95–103.
6. Neal RM, Hinton GE, Jordan MI. A view of the EM algorithm that justifies incremental, sparse, and other variants. In: Learning in graphical models. Cambridge, The MIT Press; 1999. p. 355–68.

# Chapter 10

## Hidden Markov Model

The Hidden Markov Model (HMM) is a machine learning model that can be used for labeling. It describes the process of randomly generating observation sequences from the hidden Markov chains and is a generative model. This chapter first introduces the basic concepts of the Hidden Markov Model, and then describes the probability calculation algorithms, learning algorithms, and prediction algorithms of the HMM, respectively. The Hidden Markov Model has wide applications in speech recognition, natural language processing, biological information, pattern recognition, and other fields.

### 10.1 The Basic Concept of Hidden Markov Model

#### 10.1.1 Definition of Hidden Markov Model

**Definition 10.1 (Hidden Markov Model)** The Hidden Markov Model is a probabilistic model about time sequence, which describes the process of randomly generating a random sequence of unobservable states from a hidden Markov chain, and then generating an observation from each state to generate a random observation sequence. The sequence of states randomly generated by the hidden Markov chain is called the state sequence. Each state generates an observation, and the resulting random sequence of observations is called the observation sequence. Each position in the sequence can be regarded as a point of time.

The Hidden Markov Model is determined by the initial probability distribution, the state transition probability distribution, and the observation probability distribution. The form of the Hidden Markov Model is defined as follows:

Let  $Q$  be the set of all possible states and  $V$  the set of all possible observations:

$$Q = \{q_1, q_2, \dots, q_N\}, \quad V = \{v_1, v_2, \dots, v_M\}$$

where  $N$  is the number of possible states and  $M$  is the number of possible observations.

$I$  is the state sequence of length  $T$ , and  $O$  is the corresponding observation sequence:

$$I = (i_1, i_2, \dots, i_T), \quad O = (o_1, o_2, \dots, o_T)$$

$A$  is the state transition probability matrix:

$$A = [a_{ij}]_{N \times N} \quad (10.1)$$

Among them,

$$a_{ij} = P(i_{t+1} = q_j | i_t = q_i), \quad i = 1, 2, \dots, N; \quad j = 1, 2, \dots, N \quad (10.2)$$

is the probability of a state transitioning to state  $q_j$  at time  $t + 1$ , under the condition that the state is in  $q_i$  at time  $t$ .

$B$  is the observation probability matrix:

$$B = [b_j(k)]_{N \times M} \quad (10.3)$$

where

$$\begin{aligned} b_j(k) &= P(o_t = v_k | i_t = q_j), \\ k &= 1, 2, \dots, M; \quad j = 1, 2, \dots, N \end{aligned} \quad (10.4)$$

it is the probability of generating observation  $v_k$  under the condition that the state is in  $q_i$  at time  $t$ .

$\pi$  is the initial state probability vector:

$$\pi = (\pi_i) \quad (10.5)$$

Here,

$$\pi_i = P(i_1 = q_i), \quad i = 1, 2, \dots, N \quad (10.6)$$

it is the probability that the state is in state  $q_i$  at time  $t = 1$ .

HMM is determined by the initial state probability vector  $\pi$ , the state transition probability matrix  $A$ , and the observation probability matrix  $B$ .  $\pi$  and  $A$  determine the state sequence, and  $B$  determines the observation sequence. Therefore, the Hidden Markov Model  $\lambda$  can be represented by ternary symbols, i.e.,

$$\lambda = (A, B, \pi) \quad (10.7)$$

$A, B, \pi$  are called the three elements of the Hidden Markov Model.

The state transition probability matrix  $A$  and the initial state probability vector  $\pi$  determine the hidden Markov chain and generate an unobservable state sequence. The observation probability matrix  $B$  determines how to generate observations from the state and integrates with the state sequence to determine how to generate the observation sequence.

From the definition, we know that HMM makes two basic assumptions:

- (1) the homogeneous Markov property hypothesis: it is assumed that the state of the hidden Markov chain at any time  $t$  only depends on its state at the previous time, independent of the state and observations at other times, as well as the time  $t$ :

$$P(i_t | i_{t-1}, o_{t-1}, \dots, i_1, o_1) = P(i_t | i_{t-1}), \quad t = 1, 2, \dots, T \quad (10.8)$$

- (2) Observation independence hypothesis: it is assumed that the observation at any time only depends on the state of the Markov chain at that time and has nothing to do with other observations and states:

$$\begin{aligned} & P(o_t | i_T, o_T, i_{T-1}, o_{T-1}, \\ & \quad \dots, i_{t+1}, o_{t+1}, i_t, i_{t-1}, o_{t-1}, \dots, i_1, o_1) \\ & = P(o_t | i_t) \end{aligned} \quad (10.9)$$

The Hidden Markov Model can be used for labeling, and then the state corresponds to the label. The labeling problem is to predict the corresponding labeling sequence for a given sequence of observations. It can be assumed that HMM generates data for the labeling problem. In this way, we can use the learning and prediction algorithm of the Hidden Markov Model for labeling.

An example of a Hidden Markov Model is described below.

**Example 10.1 (Box and Ball Model)** Suppose there are four boxes, each containing red and white balls. The number of red and white balls in the box is listed in Table 10.1.

Draw the ball in the following way to generate a sequence of observations of the ball's color:

- At first, randomly select one box from the four boxes with equal probability. Randomly draw one ball from the box, record its color, and put it back;

**Table 10.1** Number of red and white balls in each box

	Box			
	1	2	3	4
Number of red balls	5	3	6	8
Number of white balls	5	7	4	2

- Then, randomly transfer from the current box to the next box with the following rules: if the current box is box 1, then the next box must be box 2; if the current box is box 2 or box 3, then transfer to the left or right box with probabilities 0.4 and 0.6, respectively; if the current box is box 4, then stay in box 4 or transfer to box 3 with the same probability of 0.5;
- After determining the box to be transferred, randomly draw a ball from the box, record its color, and put it back;
- Continue like this and repeat it five times to get an observation sequence of the color of the balls:

$$O = (\text{red}, \text{red}, \text{white}, \text{white}, \text{red})$$

In this process, the observer can only observe the sequence of the ball colors, but not the box from which the balls were taken, i.e., the sequence of the boxes is not observed.

There are two random sequences in this example, the sequence of boxes (state sequence), and the observation sequence of the ball colors (observation sequence). The former is hidden, and only the latter is observable. This is an example of a Hidden Markov Model. According to the given conditions, According to the given conditions, the set of states, the set of observations, the length of the sequence, and the three elements of the model can be specified.

When the boxes correspond to states, the set of states is

$$Q = (\text{box 1}, \text{box 2}, \text{box 3}, \text{box 4}), \quad N = 4$$

When the ball's color corresponds to the observation, the set of observations is:

$$V = (\text{red}, \text{white}), \quad M = 2$$

The length of state sequence and observation sequence is  $T = 5$ .

The initial probability distribution is

$$\pi = (0.25, 0.25, 0.25, 0.25)^T$$

The state transition probability distribution is

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.4 & 0 & 0.6 & 0 \\ 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0.5 & 0.5 \end{bmatrix}$$

The observation probability distribution is

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.8 & 0.2 \end{bmatrix}$$

### 10.1.2 The Generation Process of the Observation Sequence

According to the definition of Hidden Markov Model, the generation process of an observation sequence  $O = (o_1, o_2, \dots, o_T)$  with length  $T$  can be described as follows.

#### Algorithm 10.1 (The Generation of the Observation Sequence)

Input: The Hidden Markov Model  $\lambda = (A, B, \pi)$ , observation sequence length  $T$ ;  
Output: observation sequence  $O = (o_1, o_2, \dots, o_T)$ .

- (1) Generate state  $i_1$  according to the initial state distribution  $\pi$ ;
- (2) Let  $t = 1$ ;
- (3) According to the observation probability distribution  $b_{i_t}(k)$  of state  $i_t$ ,  $o_t$  is generated;
- (4) According to the state transition probability distribution  $\{a_{i_t i_{t+1}}\}$  of state  $i_t$ , the state  $i_{t+1}$  is generated with  $i_{t+1} = 1, 2, \dots, N$ ;
- (5) Let  $t = t + 1$ ; if  $t < T$ , turn to step (3); otherwise, terminate.

### 10.1.3 Three Basic Problems of the Hidden Markov Model

The Hidden Markov Model has 3 basic problems:

- (1) The probability calculation problem of. Given the model  $\lambda = (A, B, \pi)$  and the observation sequence  $O = (o_1, o_2, \dots, o_T)$ , calculate  $P(O|\lambda)$ , the probability of the occurrence of observation sequence  $O$  under the model  $\lambda$ .
- (2) The learning problem. Given the observation sequence  $O = (o_1, o_2, \dots, o_T)$ , estimate the parameters of the model  $\lambda = (A, B, \pi)$  such that  $P(O|\lambda)$ , the probability of the observation sequence is the largest under the model. That is, the maximum likelihood estimation method is used to estimate the parameters.
- (3) The prediction problem, also known as the decoding problem. Given the model  $\lambda = (A, B, \pi)$  and the observation sequence  $O = (o_1, o_2, \dots, o_T)$ , find the state sequence  $I = (i_1, i_2, \dots, i_T)$  that maximizes the conditional probability  $P(I|O)$  for the given observation sequence. That is, finding the most probable corresponding state sequence with the given observation sequence.,

Solutions to these problems are presented in the following sections.

## 10.2 Probability Calculation Algorithms

This section introduces the forward and backward algorithms for calculating the probability  $P(O|\lambda)$ . The direct calculation method, which is feasible in concept but not feasible in calculation, is presented first.

### 10.2.1 Direct Calculation Method

Given the model  $\lambda = (A, B, \pi)$  and the observation sequence  $O = (o_1, o_2, \dots, o_T)$ , calculate the probability  $P(O|\lambda)$  of the occurrence of the observation sequence  $O$ . The most direct method is to calculate it directly according to the probability formula. By enumerating all possible state sequences  $I = (i_1, i_2, \dots, i_T)$  of length  $T$ , the joint probability  $P(O, I|\lambda)$  of each state sequence  $I$  and the observation sequence  $O = (o_1, o_2, \dots, o_T)$  is obtained, and then  $P(O|\lambda)$  is obtained by summing all possible state sequences.

The probability of the state sequence  $I = (i_1, i_2, \dots, i_T)$  is:

$$P(I|\lambda) = \pi_{i_1} a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_{T-1} i_T} \quad (10.10)$$

For a fixed state sequence  $I = (i_1, i_2, \dots, i_T)$ , the probability of the observation sequence  $O = (o_1, o_2, \dots, o_T)$  is as follows:

$$P(O|I, \lambda) = b_{i_1}(o_1) b_{i_2}(o_2) \cdots b_{i_T}(o_T) \quad (10.11)$$

The joint probability of simultaneous occurrence of  $O$  and  $I$  is

$$\begin{aligned} P(O, I|\lambda) &= P(O|I, \lambda) P(I|\lambda) \\ &= \pi_{i_1} b_{i_1}(o_1) a_{i_1 i_2} b_{i_2}(o_2) \cdots a_{i_{T-1} i_T} b_{i_T}(o_T) \end{aligned} \quad (10.12)$$

Then, the probability  $P(O|\lambda)$  of the observation sequence  $O$  is obtained by summing all the possible state sequences  $I$ , i.e.,

$$\begin{aligned} P(O|\lambda) &= \sum_I P(O|I, \lambda) P(I|\lambda) \\ &= \sum_{i_1, i_2, \dots, i_T} \pi_{i_1} b_{i_1}(o_1) a_{i_1 i_2} b_{i_2}(o_2) \cdots a_{i_{T-1} i_T} b_{i_T}(o_T) \end{aligned} \quad (10.13)$$

However, using Formula (10.13) is computationally intensive and is of  $O(TN^T)$  order. This algorithm is thus not feasible.

The following is an effective algorithm for calculating the probability  $P(O|\lambda)$  of observation sequence: the forward-backward algorithm.

### 10.2.2 Forward Algorithm

First, forward probability is defined.

**Define 10.2 (Forward Probability)** For a given Hidden Markov Model  $\lambda$ , define the probability that part of the observation sequence at time  $t$  is  $o_1, o_2, \dots, o_t$  and the state is  $q_i$  as the forward probability, and record it as

$$\alpha_t(i) = P(o_1, o_2, \dots, o_t, i_t = q_i | \lambda) \quad (10.14)$$

The forward probability  $\alpha_t(i)$  and the observation sequence probability  $P(O|\lambda)$  can be obtained recursively.

#### Algorithm 10.2 (Forward Algorithm of Observation Sequence Probability)

Input: Hidden Markov Model  $\lambda$ , observation sequence  $O$ ;

Output: observation sequence probability  $P(O|\lambda)$ .

(1) Initial value

$$\alpha_1(i) = \pi_i b_i(o_1), \quad i = 1, 2, \dots, N \quad (10.15)$$

(2) Recursion for  $t = 1, 2, \dots, T - 1$

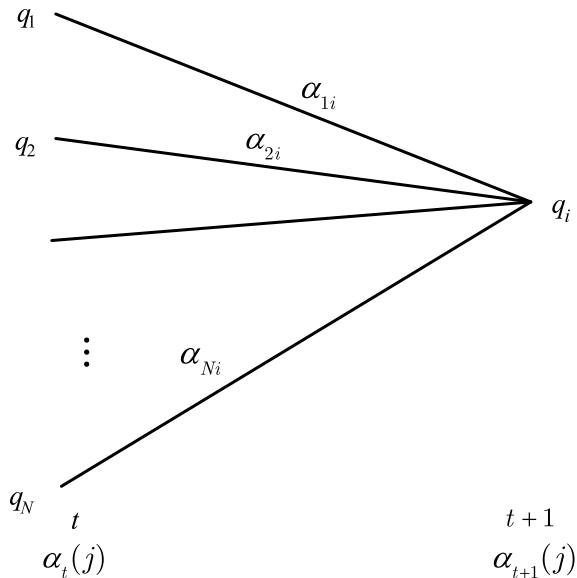
$$\alpha_{t+1}(i) = \left[ \sum_{j=1}^N \alpha_t(j) a_{ji} \right] b_i(o_{t+1}), \quad i = 1, 2, \dots, N \quad (10.16)$$

(3) Termination

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (10.17)$$

In the forward algorithm, step (1) initializes the forward probability, which is the joint probability of the initial state  $i_1 = q_i$  and the observation  $o_1$ . Step (2) is the recursion formula of the forward probability, which calculates the forward probability of the partial observation sequence being  $o_1, o_2, \dots, o_t, o_{t+1}$  at time  $t + 1$  and that of the state being in state  $q_i$  at time  $t + 1$ , as shown in Fig. 10.1. In the square brackets of Eq. (10.16), since  $\alpha_t(j)$  is the forward probability of observing  $o_1, o_2, \dots, o_t$  at time  $t$  and being in state  $q_j$  at time  $t$ , then the product  $\alpha_t(j) a_{ji}$  is the joint probability of observing  $o_1, o_2, \dots, o_t$  at time  $t$ , being in state  $q_j$  at time  $t$  and reaching state  $q_i$  at time  $t + 1$ . Summing this product over all possible  $N$  states  $q_j$  of at time  $t$ , the result is the joint probability of observing  $o_1, o_2, \dots, o_t$  at time  $t$  and being in state  $q_i$  at time  $t + 1$ . The product of the value in square brackets

**Fig. 10.1** Recursive formula for the forward probability



and the observation probability  $b_i(o_{t+1})$  is exactly the forward probability  $\alpha_{t+1}(i)$  of observing  $o_1, o_2, \dots, o_t, o_{t+1}$  at time  $t + 1$  and being in state  $q_i$  at time  $t + 1$ . Step (3) gives the formula for calculating  $P(O|\lambda)$ . Since

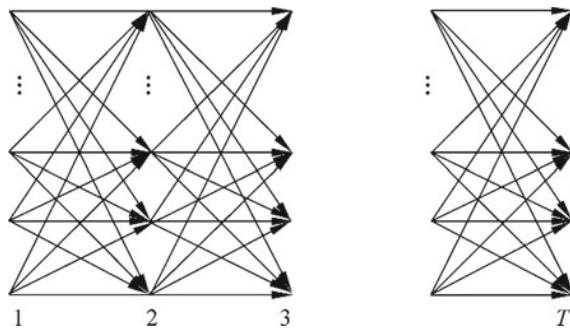
$$\alpha_T(i) = P(o_1, o_2, \dots, o_T, i_T = q_i | \lambda)$$

we have

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i)$$

As shown in Fig. 10.2, the forward algorithm is an algorithm that recursively calculates  $P(O|\lambda)$  based on the “path structure of the state sequence”. The key to the efficient forward algorithm is that it calculates the forward probability locally and then uses the path structure to “recursively” transfer the forward probability to the global level to get  $P(O|\lambda)$ . Specifically, at time  $t = 1$ , calculate the  $N$  values of  $\alpha_1(i)$ ,  $i = 1, 2, \dots, N$ ; at each time  $t = 1, 2, \dots, T - 1$ , calculate the  $N$  values of  $\alpha_{t+1}(i)$ ,  $i = 1, 2, \dots, N$ . The calculation of each  $\alpha_{t+1}(i)$  utilizes  $N\alpha_t(j)$  from the previous time. The reason for reducing the calculation is that each calculation directly refers to the calculation results of the previous point of time to avoid double calculation. In this way, the computation using forward probability to calculate  $P(O|\lambda)$  is of  $O(N^2T)$  order instead of the order  $O(TN^T)$  in direct calculation.

**Fig. 10.2** The path structure of the observation sequence



**Example 10.2** Consider the box and ball model  $\lambda = (A, B, \pi)$ , the state set  $Q = \{1, 2, 3\}$ , and the observation set  $V = \{\text{red}, \text{white}\}$ ,

$$A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix},$$

$$\pi = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.4 \end{bmatrix}$$

Set  $T = 3$ ,  $O = (\text{red}, \text{white}, \text{red})$ , and try the forward algorithm to calculate  $P(O|\lambda)$ .

**Solution** According to Algorithm 10.2

(1) Calculate the initial value

$$\alpha_1(1) = \pi_1 b_1(o_1) = 0.10$$

$$\alpha_1(2) = \pi_2 b_2(o_1) = 0.16$$

$$\alpha_1(3) = \pi_3 b_3(o_1) = 0.28$$

(2) Conduct the recursive calculation

$$\alpha_2(1) = \left[ \sum_{i=1}^3 \alpha_1(i) a_{i1} \right] b_1(o_2) = 0.154 \times 0.5 = 0.077$$

$$\alpha_2(2) = \left[ \sum_{i=1}^3 \alpha_1(i) a_{i2} \right] b_2(o_2) = 0.184 \times 0.6 = 0.1104$$

$$\begin{aligned}\alpha_2(3) &= \left[ \sum_{i=1}^3 \alpha_1(i)a_{i3} \right] b_3(o_2) = 0.202 \times 0.3 = 0.0606 \\ \alpha_3(1) &= \left[ \sum_{i=1}^3 \alpha_2(i)a_{i1} \right] b_1(o_3) = 0.04187 \\ \alpha_3(2) &= \left[ \sum_{i=1}^3 \alpha_2(i)a_{i2} \right] b_2(o_3) = 0.03551 \\ \alpha_3(3) &= \left[ \sum_{i=1}^3 \alpha_2(i)a_{i3} \right] b_3(o_3) = 0.05284\end{aligned}$$

## (3) Termination

$$P(O|\lambda) = \sum_{i=1}^3 \alpha_3(i) = 0.13022$$

**10.2.3 Backward Algorithm**

**Definition 10.3 (Backward Probability)** Given the Hidden Markov Model  $\lambda$ , define that under the condition that the state at time  $t$  is  $q_i$ , the partial observation sequence from  $t+1$  to  $T$  is  $o_{t+1}, o_{t+2}, \dots, o_T$ , which is the backward probability, denoted as

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots, o_T | i_t = q_i, \lambda) \quad (10.18)$$

The backward probability  $\beta_t(i)$  and the observation sequence probability  $P(O|\lambda)$  can be obtained by the recursive method.

**Algorithm 10.3 (Backward Algorithm of Observation Sequence Probability)**

Input: Hidden Markov Model  $\lambda$ , observation sequence  $O$ ;

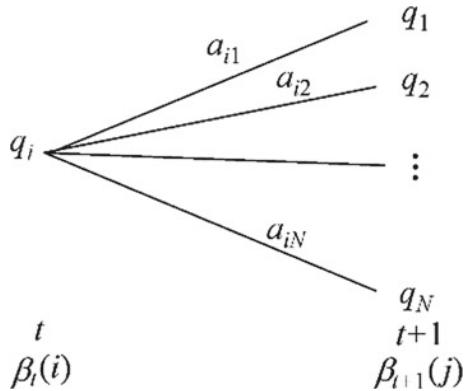
Output: Observation sequence probability  $P(O|\lambda)$ .

$$(1) \beta_T(i) = 1, \quad i = 1, 2, \dots, N \quad (10.19)$$

$$(2) \text{ For } t = T - 1, T - 2, \dots, 1$$

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad i = 1, 2, \dots, N \quad (10.20)$$

**Fig. 10.3** Recursive formula of backward probability



$$(3) P(O|\lambda) = \sum_{i=1}^N \pi_i b_i(o_1) \beta_1(i) \quad (10.21)$$

Step (1) initializes the backward probability, and specifies  $\beta_T(i) = 1$  for all states  $q_i$  at the final point of time. Step (2) is the recursive formula of backward probability. As shown in Fig. 10.3, in order to calculate the backward probability  $\beta_t(i)$  of the observation sequence  $o_{t+1}, o_{t+2}, \dots, o_T$  after the time  $t+1$  when the state is  $q_i$  at time  $t$ , just consider the transition probabilities of all possible  $N$  states  $q_j$  at time  $t+1$  (namely  $a_{ij}$  term), and the observation probability of observation  $o_{t+1}$  in this state (namely  $b_j(o_{t+1})$  term), and then consider the backward probability of the observation sequence after state  $q_j$  (namely  $\beta_{t+1}(j)$  term). The idea of finding  $P(O|\lambda)$  in step (3) is the same as that in step (2), but the initial probability  $\pi_i$  replaces the transition probability.

By using the definitions of forward probability and backward probability, the observation sequence probability  $P(O|\lambda)$  can be written uniformly as

$$P(O|\lambda) = \sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad t = 1, 2, \dots, T-1 \quad (10.22)$$

#### 10.2.4 Calculation of Some Probabilities and Expected Values

By using forward probability and backward probability, we can get the calculation formula of single state probability and two-state probability.

1. Given the model  $\lambda$  and observation  $O$ , the probability of being in state  $q_i$  at time  $t$  is denoted as

$$\gamma_t(i) = P(i_t = q_i | O, \lambda) \quad (10.23)$$

which can be calculated by forward and backward probability. In fact,

$$\gamma_t(i) = P(i_t = q_i | O, \lambda) = \frac{P(i_t = q_i, O | \lambda)}{P(O | \lambda)}$$

From the definitions of forward probability  $\alpha_t(i)$  and backward probability  $\beta_t(i)$  we can see that

$$\alpha_t(i)\beta_t(i) = P(i_t = q_i, O | \lambda)$$

Thus we can get:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O | \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \quad (10.24)$$

2. Given a model  $\lambda$  and an observation  $O$ , the probability of being in state  $q_i$  at time  $t$  and in state  $q_j$  at time  $t + 1$  is denoted as

$$\xi_t(i, j) = P(i_t = q_i, i_{t+1} = q_j | O, \lambda) \quad (10.25)$$

which can be calculated by the forward-backward probability:

$$\begin{aligned} \xi_t(i, j) &= \frac{P(i_t = q_i, i_{t+1} = q_j, O | \lambda)}{P(O | \lambda)} \\ &= \frac{P(i_t = q_i, i_{t+1} = q_j, O | \lambda)}{\sum_{i=1}^N \sum_{j=1}^N P(i_t = q_i, i_{t+1} = q_j, O | \lambda)} \end{aligned}$$

And

$$P(i_t = q_i, i_{t+1} = q_j, O | \lambda) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)$$

Thus

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)} \quad (10.26)$$

3. Summing  $\gamma_t(i)$  and  $\xi_t(i, j)$  over each point of time  $t$  yields some useful expectation values.
- (1) The expected value of the occurrence of state  $i$  under observation  $O$ :

$$\sum_{t=1}^T \gamma_t(i) \quad (10.27)$$

- (2) The expected value of the transfer from state  $i$  under observation  $O$ :

$$\sum_{t=1}^{T-1} \xi_t(i) \quad (10.28)$$

- (3) The expected value of the transfer from state  $i$  to state  $j$  under observation  $O$ :

$$\sum_{t=1}^{T-1} \xi_t(i, j) \quad (10.29)$$

## 10.3 Learning Algorithms

Learning the Hidden Markov Model can be implemented by supervised and unsupervised learning, depending on whether the training data includes the observation sequence and the corresponding state sequence or only the observation sequence. This section first introduces the supervised learning algorithm, while the unsupervised learning algorithm—Baum-Welch algorithm (also known as the EM algorithm) is introduced later.

### 10.3.1 Supervised Learning Methods

Assume that the training data containing  $S$  observation sequences of the same length and the corresponding state sequences  $\{(O_1, I_1), (O_2, I_2), \dots, (O_S, I_S)\}$  has been given, the HMM parameters can be estimated using the maximum likelihood estimation method. The specific method is as follows.

### 10.3.1.1 Estimation of the Transfer Probability $a_{ij}$ .

Let the frequency of transfer from state  $i$  at time  $t$  to state  $j$  at time  $t + 1$  in the sample be  $A_{ij}$ , then the estimation of the state transfer probability  $a_{ij}$  is

$$\hat{a}_{ij} = \frac{A_{ij}}{\sum_{j=1}^N A_{ij}}, \quad i = 1, 2, \dots, N, \quad j = 1, 2, \dots, N \quad (10.30)$$

### 10.3.1.2 Estimation of the Observation Probability $b_j(k)$

Let the number of frequencies in the sample with state  $j$  and observation  $k$  be  $B_{jk}$ , then the estimation of the probability  $b_j(k)$  that state  $j$  is observed as  $k$  is

$$\hat{b}_j(k) = \frac{B_{jk}}{\sum_{k=1}^M B_{jk}}, \quad j = 1, 2, \dots, N; \quad k = 1, 2, \dots, M \quad (10.31)$$

### 10.3.1.3 The Estimate $\pi_i$ of the Initial State Probability

$\hat{\pi}_i$  is the frequency of the initial state  $q_i$  in  $S$  samples.

Since supervised learning requires labeled training data, and manually labeled training data is often costly, unsupervised learning methods are sometimes utilized.

## 10.3.2 Baum-Welch Algorithm

Assume that the given training data contains only  $S$  observation sequences  $\{O_1, O_2, \dots, O_S\}$  of length  $T$  without corresponding state sequences, the objective is to learn the parameter of the Hidden Markov Model  $\lambda = (A, B, \pi)$ . We regard the observation sequence data as observation data  $O$  and the state series data as unobservable hidden data  $I$ . Then the Hidden Markov Model is in fact a probabilistic model with hidden variables

$$P(O|\lambda) = \sum_I P(O|I, \lambda)P(I|\lambda) \quad (10.32)$$

Its parameter learning can be implemented by the EM algorithm.

### 10.3.2.1 Determine the Log-Likelihood Function of the Complete Data

All observation data are written as  $O = (o_1, o_2, \dots, o_T)$ , all hidden data are written as  $I = (i_1, i_2, \dots, i_T)$ , and the complete data are  $(O, I) = (o_1, o_2, \dots, o_T, i_1, i_2, \dots, i_T)$ . The log-likelihood function for the complete data is  $\log P(O, I|\lambda)$ .

### 10.3.2.2 E-step of the EM Algorithm: Finding the $Q$ Function

$$Q(\lambda, \bar{\lambda})^1$$

$$Q(\lambda, \bar{\lambda}) = \sum_I \log P(O, I|\lambda) P(O, I|\bar{\lambda}) \quad (10.33)$$

where  $\bar{\lambda}$  is the current estimate of the HMM parameters and  $\lambda$  is the HMM parameter to be maximized.

$$P(O, I|\lambda) = \pi_{i1} b_{i1}(o_1) a_{i1i2} b_{i2}(o_2) \dots a_{iT-1iT} b_{iT}(o_T)$$

Thus, the function  $Q(\lambda, \bar{\lambda})$  can be written as:

$$\begin{aligned} Q(\lambda, \bar{\lambda}) &= \sum_I \log \pi_{i1} P(O, I|\bar{\lambda}) \\ &+ \sum_I \left( \sum_{t=1}^{T-1} \log a_{i_t i_{t+1}} \right) P(O, I|\bar{\lambda}) \\ &+ \sum_I \left( \sum_{t=1}^T \log b_{i_t}(o_t) \right) P(O, I|\bar{\lambda}) \end{aligned} \quad (10.34)$$

where the summation is performed for the total sequence length  $T$  of all data.

### 10.3.2.3 M-Step of the EM Algorithm: Maximizing $Q$ Function $Q(\lambda, \bar{\lambda})$ to Find the Model Parameter $A, B, \pi$

Since the parameters to be maximized appear in three separate terms in Eq. (10.34), only each term needs to be maximized separately.

- (1) The first term of Eq. (10.34) can be written as:

---

<sup>1</sup> According to the definition of  $Q$  function

$$Q(\lambda, \bar{\lambda}) = E_I [\log P(O, I|\lambda) | O, \bar{\lambda}]$$

Equation (10.33) omits the constant factor  $1 / P(O|\bar{\lambda})$  for  $\lambda$ .

$$\sum_I \log \pi_i P(O, I | \bar{\lambda}) = \sum_{i=1}^N \log \pi_i P(O, i_1 = i | \bar{\lambda})$$

Note that  $\pi_i$  satisfies the constraint  $\sum_{i=1}^N \pi_i = 1$ , and use the Lagrange multiplier method to write the Lagrange function:

$$\sum_{i=1}^N \log \pi_i P(O, i_1 = i | \bar{\lambda}) + \gamma \left( \sum_{i=1}^N \pi_i - 1 \right)$$

Find its partial derivative and set the result to 0

$$\frac{\partial}{\partial \pi_i} \left[ \sum_{i=1}^N \log \pi_i P(O, i_1 = i | \bar{\lambda}) + \gamma \left( \sum_{i=1}^N \pi_i - 1 \right) \right] = 0 \quad (10.35)$$

Then we have

$$P(O, i_1 = i | \bar{\lambda}) + \gamma \pi_i = 0$$

Sum over  $i$  and get  $\gamma$

$$\gamma = -P(O | \bar{\lambda})$$

Substitute it into Eq. (10.35) to get

$$\pi_i = \frac{P(O, i_1 = i | \bar{\lambda})}{P(O | \bar{\lambda})} \quad (10.36)$$

(2) The second term of Eq. (10.34) can be written as

$$\begin{aligned} & \sum_I \left( \sum_{t=1}^{T-1} \log a_{i_t i_{t+1}} \right) P(O, I | \bar{\lambda}) \\ &= \sum_{i=1}^N \sum_{j=1}^N \sum_{t=1}^{T-1} \log a_{ij} P(O, i_t = i, i_{t+1} = j | \bar{\lambda}) \end{aligned}$$

Similar to term 1, the Lagrange multiplier method with constraint  $\sum_{j=1}^N a_{ij} = 1$  can be applied to find

$$a_{ij} = \frac{\sum_{t=1}^{T-1} P(O, i_t = i, i_{t+1} = j | \bar{\lambda})}{\sum_{t=1}^{T-1} P(O, i_t = i | \bar{\lambda})} \quad (10.37)$$

(3) The third term of Eq. (10.34) is

$$\sum_I \left( \sum_{t=1}^T \log b_{i_t}(o_t) \right) P(O, I | \bar{\lambda}) = \sum_{j=1}^N \sum_{t=1}^T \log b_j(o_t) P(O, i_t = j | \bar{\lambda})$$

The Lagrange multiplier method is also used, and the constraint is  $\sum_{k=1}^M b_j(k) = 1$ . Note that the partial derivative of  $b_j(o_t)$  with respect to  $b_j(k)$  is not 0 only when  $o_t = v_k$ , denoted by  $I(o_t = v_k)$ . Get

$$b_j(k) = \frac{\sum_{t=1}^T P(O, i_t = j | \bar{\lambda}) I(o_t = v_k)}{\sum_{t=1}^T P(O, i_t = j | \bar{\lambda})} \quad (10.38)$$

### 10.3.3 Baum-Welch Model Parameter Estimation Formula

By expressing each probability in Eq. (10.36) ~ Eq. (10.38) as  $\gamma_t(i)$ ,  $\xi_t(i, j)$ , the corresponding equation can be written as:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (10.39)$$

$$b_j(k) = \frac{\sum_{t=1, o_t=v_k}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \quad (10.40)$$

$$\pi_i = \gamma_1(i) \quad (10.41)$$

where  $\gamma_t(i)$ ,  $\xi_t(i, j)$  are given according to Eq. (10.24) and Eq. (10.24), respectively. Equations (10.39) ~ (10.41) are the Baum-Welch algorithm proposed by Baum and Welch, which is the specific realization of the EM algorithm in HMM learning.

#### Algorithm 10.4 (Baum-Welch algorithm)

Input: observation data  $O = (o_1, o_2, \dots, o_T)$ ;

Output: the HMM parameter.

- (1) Initialization. For  $n = 0$ , select  $a_{ij}^{(0)}$ ,  $b_j(k)^{(0)}$ ,  $\pi_i^{(0)}$  and obtain a model  $\lambda^{(0)} = (A^{(0)}, B^{(0)}, \pi^{(0)})$ .
- (2) Iteration. For  $n = 1, 2, \dots$ ,

$$a_{ij}^{(n+1)} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_j(k)^{(n+1)} = \frac{\sum_{t=1, o_t=v_k}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$$

$$\pi_i^{(n+1)} = \gamma_1(i)$$

The right end values are calculated according to observation  $O = (o_1, o_2, \dots, o_T)$  and model  $\lambda^{(n)} = (A^{(n)}, B^{(n)}, \pi^{(n)})$ .  $\gamma_t(i)$ ,  $\xi_t(i, j)$  in the equation are given by Eq. (10.24) and Eq. (10.26).

- (3) Termination. Obtain the model parameter  $\lambda^{(n+1)} = (A^{(n+1)}, B^{(n+1)}, \pi^{(n+1)})$ .

## 10.4 Prediction Algorithm

Here are two algorithms of Hidden Markov Model prediction: Approximation algorithm and Viterbi algorithm.

### 10.4.1 Approximation Algorithm

The idea of approximation algorithm is to select the most likely state  $i_t^*$  at each point of time  $t$ , so as to get a state sequence  $I^* = (i_1^*, i_2^*, \dots, i_T^*)$ , which is used as the prediction result.

Given the Hidden Markov Model  $\lambda$  and observing sequence  $O$ , the probability  $\gamma_t(i)$  of being in state  $q_i$  at time  $t$  is

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \quad (10.42)$$

the most likely state  $i_t^*$  at each point of time  $t$  is

$$i_t^* = \arg \max_{1 \leq i \leq N} [\gamma_t(i)], \quad t = 1, 2, \dots, T \quad (10.43)$$

then we can obtain the state sequence  $I^* = (i_1^*, i_2^*, \dots, i_T^*)$ .

The advantage of the approximation algorithm is the simple calculation, and the disadvantage is that there is no guarantee that the predicted state sequence as a whole is the most likely state sequence, because the predicted state sequence may have parts that do not actually occur. In fact, the state sequence obtained by the above method may have adjacent states with a transition probability of 0, i.e., for some  $i, j, a_{ij} = 0$ . Nevertheless, the approximation algorithm is still useful.

### 10.4.2 Viterbi Algorithm

In fact, Viterbi algorithm uses dynamic programming to solve the Hidden Markov Model measurement problem, i.e., using dynamic programming to find the maximum probability path (the optimal path). At this point of time, a path corresponds to a sequence of the state.

According to the principle of dynamic programming, the optimal path has the characteristic that if the optimal path pass through node  $i_t^*$  at point of time  $t$ , then the partial path from node  $i_t^*$  to terminal  $i_T^*$  must be optimal for all possible partial paths from  $i_t^*$  to  $i_T^*$ . Because if not, there is another better partial path from  $i_t^*$  to  $i_T^*$ . If it is connected with the partial path from  $i_1^*$  to  $i_t^*$ , it will form a path better than the original one, which is contradictory. Based on this principle, we only need to calculate the maximum probability of each partial path with state  $i$  at point of time  $t$  from point of time  $t = 1$ , until we get the maximum probability of each path with state  $i$  at point of time  $t = T$ . The maximum probability of point of time  $t = T$  is the probability  $P^*$  of the optimal path, and the endpoint  $i_T^*$  of the optimal path is also obtained. Then, in order to find out the nodes of the optimal path, the nodes  $i_{T-1}^*, \dots, i_1^*$  are obtained from the end point  $i_T^*$ , and then  $I^* = (i_1^*, i_2^*, \dots, i_T^*)$ . This is the Viterbi algorithm.

First, two variables  $\delta$  and  $\psi$  are imported. It is defined that the maximum probability of all single paths  $(i_1, i_2, \dots, i_t)$  with state  $i$  at time  $t$  is

$$\begin{aligned} \delta_t(i) &= \max_{i_1, i_2, \dots, i_{t-1}} P(i_t = i, i_{t-1}, \dots, i_1, o_t, \dots, o_1 | \lambda), \\ i &= 1, 2, \dots, N \end{aligned} \quad (10.44)$$

We can get the iteration equation of variable  $\delta$  from the definition:

$$\begin{aligned}\delta_{t+1}(i) &= \max_{i_1, i_2, \dots, i_t} P(i_{t+1} = i, i_t, \dots, i_1, o_{t+1}, \dots, o_1 | \lambda) \\ &= \max_{1 \leq j \leq N} [\delta_t(j)a_{ji}] b_i(o_{t+1}), \\ i &= 1, 2, \dots, N; t = 1, 2, \dots, T - 1\end{aligned}\quad (10.45)$$

The  $(t - 1)$ -th node of the path with the highest probability in all single paths  $(i_1, i_2, \dots, i_{t-1}, i)$  defined at time  $t$  with state  $i$  is

$$\psi_t(i) = \arg \max_{1 \leq j \leq N} [\delta_{t-1}(j)a_{ji}], \quad i = 1, 2, \dots, N \quad (10.46)$$

Viterbi algorithm is introduced as follows.

#### Algorithm 10.5 (Viterbi Algorithm)

Input: model  $\lambda = (A, B, \pi)$  and observation  $O = (o_1, o_2, \dots, o_T)$ ;

Output: the optimal path  $I^* = (i_1^*, i_2^*, \dots, i_T^*)$ .

(1) Initialization

$$\delta_1(i) = \pi_i b_i(o_1), \quad i = 1, 2, \dots, N$$

$$\psi_1(i) = 0, \quad i = 1, 2, \dots, N$$

(2) Iteration. For  $t = 2, 3, \dots, T$

$$\delta_t(i) = \max_{1 \leq j \leq N} [\delta_{t-1}(j)a_{ji}] b_i(o_t), \quad i = 1, 2, \dots, N$$

$$\psi_t(i) = \arg \max_{1 \leq j \leq N} [\delta_{t-1}(j)a_{ji}], \quad i = 1, 2, \dots, N$$

(3) Termination

$$P^* = \max_{1 \leq j \leq N} \delta_T(i)$$

$$i_T^* = \arg \max_{1 \leq j \leq N} [\delta_T(i)]$$

(4) Optimal path backtracking. For  $t = T - 1, T - 2, \dots, 1$

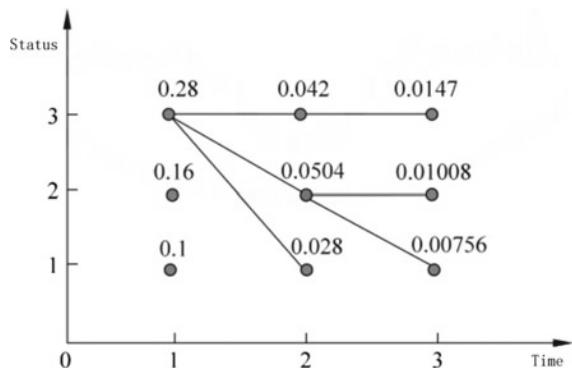
$$i_t^* = \psi_{t+1}(i_{t+1}^*)$$

find the optimal path  $I^* = (i_1^*, i_2^*, \dots, i_T^*)$ .

The Viterbi algorithm is illustrated by an example below.

**Example 10.3** The model  $\lambda = (A, B, \pi)$  of example 10.2,

**Fig. 10.4** Solving for the optimal path



$$A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix},$$

$$\pi = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.4 \end{bmatrix}$$

Given the observation sequence  $O = (\text{Red, White, Red})$ , try to find the optimal state sequence, i.e., the optimal path  $I^* = (i_1^*, i_2^*, i_3^*)$ .

**Solution** As shown in Fig. 10.4, follow the following steps to select an optimal path among all possible paths:

- (1) Initialization. At  $t = 1$ , for each state  $i$ ,  $i = 1, 2, 3$ , find the probability that the state  $i$  observes  $o_1$  as red, and denote this probability as  $\delta_1(i)$ , then

$$\delta_1(i) = \pi_i b_i(o_1) = \pi_i b_i(\text{red}), \quad i = 1, 2, 3$$

Substitute actual data into the above equation:

$$\delta_1(1) = 0.10, \quad \delta_1(2) = 0.16, \quad \delta_1(3) = 0.28$$

Denote that  $\psi_1(i) = 0$  and  $i = 1, 2, 3$ .

- (2) At  $t = 2$ , for each state  $i$ ,  $i = 1, 2, 3$ , find the maximum probability of a path with state  $j$  observing red at  $t = 1$  and state  $i$  observing  $o_2$  white at  $t = 2$ . Denote this maximum probability as  $\delta_2(i)$ , then

$$\delta_2(i) = \max_{1 \leq j \leq 3} [\delta_1(j) a_{ji}] b_i(o_2)$$

Also, for each state  $i$ ,  $i = 1, 2, 3$ , record the previous state  $j$  with the highest probability path.

$$\Psi_2(i) = \underset{1 \leq j \leq 3}{\operatorname{argmax}} [\delta_1(j)a_{ji}], \quad i = 1, 2, 3$$

Calculate:

$$\begin{aligned}\delta_2(1) &= \max_{1 \leq j \leq 3} [\delta_1(j)a_{j1}]b_1(o_2) \\&= \max_j \{0.10 \times 0.5, 0.16 \times 0.3, 0.28 \times 0.2\} \times 0.5 \\&= 0.028 \\ \psi_2(1) &= 3 \\ \delta_2(2) &= 0.0504 \\ \psi_2(2) &= 3 \\ \delta_2(3) &= 0.042 \\ \psi_2(3) &= 3\end{aligned}$$

Similarly, at  $t = 3$ ,

$$\begin{aligned}\delta_3(i) &= \max_{1 \leq j \leq 3} [\delta_2(j)a_{ji}]b_i(o_3) \\ \psi_3(i) &= \arg \max_{1 \leq j \leq 3} [\delta_2(j)a_{ji}] \\ \delta_3(1) &= 0.00756 \\ \psi_3(1) &= 2 \\ \delta_3(2) &= 0.01008 \\ \psi_3(2) &= 2 \\ \delta_3(3) &= 0.0147 \\ \Psi_3(3) &= 3\end{aligned}$$

(3) Denote the probability of the optimal path by  $P^*$ , then

$$P^* = \max_{1 \leq i \leq 3} \delta_3(i) = 0.0147$$

The endpoint of the optimal path is  $i_3^*$ :

$$i_3^* = \arg \max_i [\delta_3(i)] = 3$$

(4) Reverse from the endpoint  $i_3^*$  of the optimal path to find  $i_2^*, i_1^*$ .

$$\text{At } t = 2, i_2^* = \psi_3(i_3^*) = \psi_3(3) = 3$$

$$\text{At } t = 1, i_1^* = \psi_2(i_2^*) = \psi_2(3) = 3$$

The optimal path, i.e., the optimal state sequence  $I^* = (i_1^*, i_2^*, i_3^*) = (3, 3, 3)$  is then found.

## Summary

1. The Hidden Markov Model (HMM) is the probabilistic model of time sequences that describes the process of randomly generating an unobservable random state sequence from a hidden Markov chain, and then generating an observation from each state to generate an observation random sequence.

HMM is determined by the initial state probability vector  $\pi$ , the state transfer probability matrix  $A$  and the observation probability matrix  $B$ . Therefore, the Hidden Markov Model can be written as  $\lambda = (A, B, \pi)$ .

HMM is a generative model that represents the joint distribution of state sequences and observation sequences, but the state sequences are hidden and unobservable.

The Hidden Markov Model can be used for labeling, when the state corresponds to the labeling. The labeling problem is to predict its corresponding labeled sequence given the observation sequence.

2. Probability calculation problem. Given the model  $\lambda = (A, B, \pi)$  and the observation sequence  $O = (o_1, o_2, \dots, o_T)$ , calculate the probability  $P(O|\lambda)$  of the occurrence of the observation sequence  $O$  under the model  $\lambda$ . The forward-backward algorithm can efficiently perform probability calculations for HMMs by recursively computing forward-backward probabilities.
3. Learning issues. The observation sequence  $O = (o_1, o_2, \dots, o_T)$  is known and the model  $\lambda = (A, B, \pi)$  parameters are estimated such that the probability of the observation sequence  $P(O|\lambda)$  is maximized under the model, i.e., the parameters are estimated by the method of great likelihood estimation. The Baum-Welch algorithm, also known as the EM algorithm, can be efficiently trained on the Hidden Markov Model. It is an unsupervised learning algorithm.
4. Prediction problem. Knowing the model  $\lambda = (A, B, \pi)$  and the observation sequence  $O = (o_1, o_2, \dots, o_T)$ , find the state sequence  $I = (i_1, i_2, \dots, i_T)$  that maximize the conditional probability  $P(I|O)$  for a given observation sequence. The Viterbi algorithm applies dynamic programming to efficiently solve the optimal path, i.e., the sequence of states with the highest probability.

## Further Reading

An introduction to the Hidden Markov Model can be found in the literature [1, 2], and in particular, the literature [1] is the classic introductory paper. The Baum-Welch algorithm can be found in the literature [3, 4]. It can be considered that the probabilistic context-free grammar is a generalization of the Hidden Markov Model, where the unobservable data of the Hidden Markov Model are state sequences, while the unobservable data of the probabilistic context-free grammar are context-free grammar trees [5]. A dynamic Bayesian network is a Bayesian network defined on

time sequence data, which contains a Hidden Markov Model and is a special case [6].

## Exercises

- 10.1 The Hidden Markov Model  $\lambda = (A, B, \pi)$  of boxes and balls is given. Here,

$$A = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix},$$

$$\pi = (0.2 \ 0.4 \ 0.4)^T$$

Let  $T = 4$ ,  $O = (\text{red, white, red, white})$  and try the backward algorithm to calculate  $P(O|\lambda)$ .

- 10.2 The Hidden Markov Model  $\lambda = (A, B, \pi)$  of boxes and balls is given. Here,

$$A = \begin{bmatrix} 0.5 & 0.1 & 0.4 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.2 & 0.6 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \\ 0.7 & 0.3 \end{bmatrix},$$

$$\pi = (0.2 \ 0.3 \ 0.5)^T$$

Let  $T = 8$ ,  $O = (\text{red, white, red, red, white, red, white, white})$  and try the backward algorithm to calculate  $P(i_4 = q_3 | O, \lambda)$ .

- 10.3 In Exercise 10.1, try to find the optimal path  $I^* = (i_1^*, i_2^*, i_3^*, i_4^*)$  by the Viterbi algorithm.
- 10.4 Try to derive the formula using forward and backward probabilities:

$$P(O|\lambda) = \sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) \alpha_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad t = 1, 2, \dots, T-1$$

- 10.5 Compare the main differences between the calculations of the variable  $\delta$  in the Viterbi algorithm and the variable  $\alpha$  in the forward algorithm.

## References

1. Rabiner LR, Juang BH. An introduction to hidden Markov Models. *IEEE ASSP Mag.* 1986;3(1):4–16.
2. Rabiner L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE.* 1989;77(2):257–86.
3. Baum L, et al. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann Math Stat.* 1970;41:164–71.
4. Bilmes JA. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. <http://ssl.ee.washington.edu/bilmes/mypubs/bil-mes1997-em.pdf>.
5. Lari K, Young SJ. Applications of stochastic context-free grammars using the Inside-Outside algorithm. *Comput Speech Lang.* 1991;5(3):237–57.
6. Ghahramani Z. Learning dynamic Bayesian networks. In: Processing of sequences and data structures. NN 1997. Lecture notes in computer science, vol. 1387. Berlin, Heidelberg: Springer; 1997. p. 168–197.

# Chapter 11

## Conditional Random Field

The Conditional Random Field (CRF) is the conditional probability distribution model of a set of output random variables with a given set of input random variables. It is characterized by the assumption that the output random variables constitute a Markov Random Field. The CRF can be used in different prediction problems, but this book only deals with its application in labeling problems. Therefore, we focus on the linear chain CRF, where the problem becomes a discriminative model that predicts the output sequence from the input sequence, in the form of a log-linear model. The learning method is usually the maximum likelihood estimation or the regularized maximum likelihood estimation. The application of linear chain CRF to labeling problems was proposed by Lafferty et al. in 2001.

This chapter first introduces the probabilistic undirected graphical model, then describes the definition and various representations of Conditional Random Field, and finally introduces three basic problems of CRF: probability calculation problem, learning problem and prediction problem.

### 11.1 Probabilistic Undirected Graphical Model

The probabilistic undirected graphical model, also called Markov Random Field, is a joint probability distribution that can be represented by undirected graphs. This section first describes the definition of probabilistic undirected graphical model, and then introduces the factorization of probabilistic undirected graphical model.

#### 11.1.1 Model Definition

A graph is a collection of nodes and edges connecting nodes. The nodes and edges are denoted as  $v$  and  $e$ , the sets of nodes and edges are denoted as  $V$  and  $E$ , and the

graph is denoted as  $G = (V, E)$ . An undirected graph is a graph whose edges have no direction.

The probabilistic graphical model is a probability distribution represented by a graph. There is a joint probability distribution  $P(Y)$ .  $Y \in y$  is a group of random variables. The undirected graph  $G = (V, E)$  represents the probability distribution  $P(Y)$ , i.e., in the graph  $G$ , the node  $V \in v$  represents a random variable  $Y_v$ ,  $Y = (Y_v)_{v \in V}$ ; the edge  $e \in E$  represents the probability dependence between random variables.

A joint probability distribution  $P(Y)$  and its undirected graph  $G$  are given. The pairwise Markov property, local Markov property and global Markov property of random variables represented by undirected graph are then defined.

Pairwise Markov property: let  $u$  and  $v$  be any two nodes in an undirected graph  $G$  that have no edge connection, and the nodes  $u$  and  $v$  correspond to random variables  $Y_u$  and  $Y_v$ , respectively. All other nodes are  $O$ , and the corresponding random variable group is  $Y_O$ . Then Pairwise Markov property means that the random variables  $Y_u$  and  $Y_v$  are conditionally independent when the random variable group  $Y_O$  is given, i.e.,

$$P(Y_u, Y_v | Y_O) = P(Y_u | Y_O)P(Y_v | Y_O) \quad (11.1)$$

Local Markov property: let  $v \in V$  be any node in an undirected graph  $G$ ,  $W$  represents all the nodes connected with  $v$ , and  $O$  represents all nodes other than  $v$  and  $W$ . The random variable denoted by  $v$  is  $Y_v$ , the random variable group represented by  $W$  is  $Y_W$ , and the random variable group represented by  $O$  is  $Y_O$ . Local Markov property means that the random variable group  $Y_O$  and the random variable group  $Y_v$  are independent under the given random variable group  $Y_W$ ,

$$P(Y_v, Y_O | Y_W) = P(Y_v | Y_W)P(Y_O | Y_W) \quad (11.2)$$

When  $P(Y_O | Y_W) > 0$ , equivalently:

$$P(Y_v | Y_W) = P(Y_v | Y_W, Y_O) \quad (11.3)$$

Figure 11.1 shows the local Markov property shown by Eq. (11.2) or Eq. (11.3).

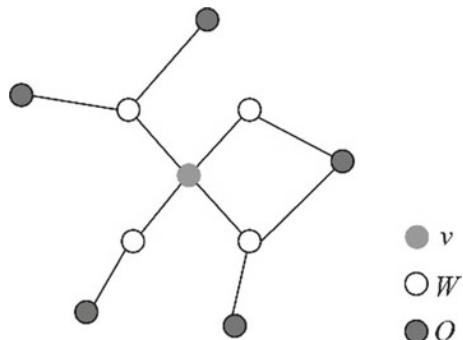
Global Markov property: let node sets  $A$  and  $B$  be any node set separated by node set  $C$  in the undirected graph  $G$ , as shown in Fig. 11.2. The random variable groups corresponding to sets  $A$ ,  $B$  and  $C$  are  $Y_A$ ,  $Y_B$  and  $Y_C$ , respectively. Global Markov property means that the random variable groups  $Y_A$  and  $Y_B$  are conditionally independent when the random variable group  $Y_C$  is given.

$$P(Y_A, Y_B | Y_C) = P(Y_A | Y_C)P(Y_B | Y_C) \quad (11.4)$$

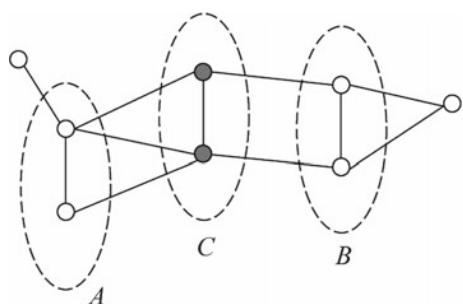
The above definitions of pairwise, local and global Markov properties are equivalent [2].

The probabilistic undirected graphical model is defined below.

**Fig. 11.1** Local Markov property



**Fig. 11.2** Global Markov property



**Definition 11.1** (*Probabilistic undirected graphical model*) A joint probability distribution  $P(Y)$  is represented by the undirected graph  $G = (V, E)$ , in which the nodes represent random variables and the edges represent the dependencies between random variables. If The joint probability distribution  $P(Y)$  is called a probabilistic undirected graphical model, or Markov Random Field, if it satisfies the pairwise, local, or global Markov property.

The above is the definition of the probabilistic undirected graphical model, but in fact, we are more concerned about how to find its joint probability distribution. For a given probabilistic undirected graphical model, we would like to write the overall joint probability as the product of several sub-join probabilities, i.e., factorize the joint probability to facilitate the learning and computation of the model. In fact, the most important feature of the probabilistic undirected graphical model is the easiness of factorization. This result is described below.

### 11.1.2 Factorization of Probabilistic Undirected Graphical Model

The definition of the clique and the maximal clique in the undirected graph is given first.

**Definition 11.2 (Clique and Maximal Clique)** A subset of nodes in the undirected graph  $G$  with any two nodes are connected by edges is called a clique. If  $C$  is a clique of the undirected graph  $G$ , and no more nodes of  $G$  can be added to make it a larger clique, then  $C$  is called the maximal clique.

Figure 11.3 shows an undirected graph composed of 4 nodes. In the figure, there are 5 cliques composed of 2 nodes:  $\{Y_1, Y_2\}$ ,  $\{Y_2, Y_3\}$ ,  $\{Y_3, Y_4\}$ ,  $\{Y_4, Y_2\}$  and  $\{Y_1, Y_3\}$ . There are 2 maximal cliques:  $\{Y_1, Y_2, Y_3\}$  and  $\{Y_2, Y_3, Y_4\}$ . However,  $\{Y_1, Y_2, Y_3, Y_4\}$  is not a clique, because  $Y_1$  and  $Y_4$  have no edges connected.

The operation of expressing the joint probability distribution of the probabilistic undirected graphical model as the product of the function of the random variables on its maximal clique is called the factorization of the probabilistic undirected graphical model.

Give a probabilistic undirected graphical model and let the undirected graph be  $G$ .  $C$  is the maximal clique on  $G$ , and  $Y_C$  denotes the random variable corresponding to  $C$ . Then the joint probability distribution  $P(Y)$  of the probabilistic undirected graphical model can be written as the product of the function  $\Psi_C(Y_C)$  on all the maximal cliques  $C$  in the graph, i.e.,

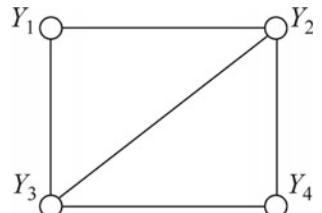
$$P(Y) = \frac{1}{Z} \prod_C \Psi_C(Y_C) \quad (11.5)$$

where  $Z$  is the normalization factor given by the following equation.

$$Z = \sum_Y \prod_C \Psi_C(Y_C) \quad (11.6)$$

The normalization factor ensures that  $P(Y)$  forms a probability distribution. The function  $\Psi_C(Y_C)$  is called the potential function. Here, the potential function  $\Psi_C(Y_C)$  is required to be strictly positive, usually defined as an exponential function:

**Fig. 11.3** The clique and maximum clique of an undirected graph



$$\Psi_C(Y_C) = \exp\{-E(Y_C)\} \quad (11.7)$$

The factorization of the probabilistic undirected graphical model is guaranteed by the following theorem.

**Theorem 11.1** (Hammersley-Clifford theorem) *The joint probability distribution  $P(Y)$  of the probabilistic undirected graphical model can be expressed in the following form:*

$$P(Y) = \frac{1}{Z} \prod_C \Psi_C(Y_C)$$

$$Z = \sum_Y \prod_C \Psi_C(Y_C)$$

where  $C$  is the maximal clique of the undirected graph,  $Y_C$  is the random variable corresponding to the nodes of  $C$ ,  $\Psi_C(Y_C)$  is a strictly positive function defined on  $C$ , and the product is performed on all maximal cliques of the undirected graph.

## 11.2 The Definition and Forms of Conditional Random Field

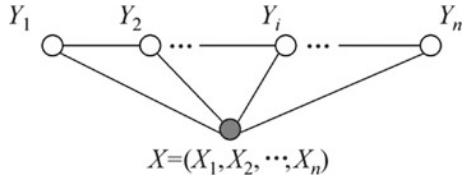
### 11.2.1 The Definition of Conditional Random Field

The Conditional Random Field (CRF) is the Markov Random Field of the random variable  $Y$  under the given random variable  $X$ . The special CRF defined on the linear chain, called the linear chain CRF, is mainly introduced here. The linear chain CRF can be used in problems such as labeling. At this time, in the conditional probability model  $P(Y|X)$ ,  $Y$  is the output variable, which represents the labeling sequence, and  $X$  is the input variable representing the observation sequence to be labeled. The labeling sequence is also called a state sequence (see Hidden Markov Model). For learning, the conditional probability model  $\hat{P}(Y|X)$  is obtained by using the training dataset through maximum likelihood estimation or regularized maximum likelihood estimation; for predicting, for a given input sequence  $x$ , the output sequence  $\hat{y}$  with the maximal conditional probability  $\hat{P}(y|x)$  is found.

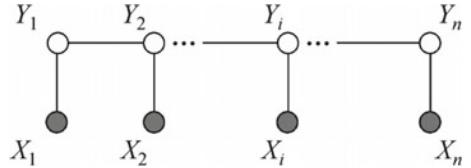
The general Conditional Random Field is defined first, then the linear CRF is defined.

**Definition 11.3** (*Conditional Random Field*) Let  $X$  and  $Y$  be random variables.  $P(Y|X)$  is the conditional probability distribution of  $Y$  with  $X$  given. If the random variable  $Y$  forms a Markov Random Field represented by the undirected graph  $G = (V, E)$ , i.e., the equation

**Fig. 11.4** The linear chain Conditional Random Field



**Fig. 11.5** The linear chain Conditional Random Field when  $X$  and  $Y$  have the same graph structure



$$P(Y_v|X, Y_w, w \neq v) = P(Y_v|X, Y_w, w \sim v) \quad (11.8)$$

holds for any node  $v$ , the conditional probability distribution  $P(Y|X)$  is called a Conditional Random Field. Here,  $w \sim v$  denotes all nodes  $w$  that are edge-connected with node  $v$  in the graph  $G = (V, E)$ , and  $w \neq v$  denotes all nodes except node  $v$ .  $Y_v$ ,  $Y_u$  and  $Y_w$  are the random variables corresponding to nodes  $v$ ,  $u$  and  $w$ .

There is no requirement in the definition that  $X$  and  $Y$  have the same structure. It is generally assumed that  $X$  and  $Y$  have the same graph structure. This book mainly considers the case where the undirected graph is a linear chain as shown in Figs. 11.4 and 11.5, i.e.,

$$G = (V = \{1, 2, \dots, n\}, E = \{(i, i + 1)\}), \quad i = 1, 2, \dots, n - 1$$

In this case,  $X = (X_1, X_2, \dots, X_n)$ ,  $Y = (Y_1, Y_2, \dots, Y_n)$ , and the maximal clique is the set of two adjacent nodes. The linear chain CRF has the following definition.

**Definition 11.4 (Linear Chain Conditional Random Field)** Let  $X = (X_1, X_2, \dots, X_n)$  and  $Y = (Y_1, Y_2, \dots, Y_n)$  both be sequences of random variables represented by linear chains. If the random variables sequence  $X$  is given and the conditional probability distribution  $P(Y|X)$  of random variables sequence  $Y$  forms a conditional random field, i.e., it satisfies the Markov property,

$$\begin{aligned} P(Y_i|X, Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_n) &= P(Y_i|X, Y_{i-1}, Y_{i+1}) \\ i = 1, 2, \dots, n \quad (\text{Only one side is considered when } i = 1 \text{ and } i = n) \end{aligned} \quad (11.9)$$

then  $P(Y|X)$  is said to be a linear chain Conditional Random Field. In the labeling problem,  $X$  represents the input observation sequence, and  $Y$  denotes the corresponding output labeling sequence or state sequence.

### 11.2.2 The Parameterized Form of the Conditional Random Field

According to Theorem 11.1, the factorization formula of the linear chain Conditional Random Field  $P(Y|X)$  can be given, where each factor is a potential function defined on two adjacent nodes (the maximal clique).

**Theorem 11.2** (The parameterized form of the linear chain Conditional Random Field) *Let  $P(Y|X)$  be the linear chain CRF. Under the condition that the value of random variable  $X$  is  $x$ , the conditional probability that the value of random variable  $Y$  is  $y$  has the following form.*

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i)\right) \quad (11.10)$$

where,

$$Z(x) = \sum_y \exp\left(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i)\right) \quad (11.11)$$

Here  $t_k$  and  $s_l$  are feature functions, and  $\lambda_k$  and  $\mu_l$  are the corresponding weights.  $Z(x)$  is the normalization factor, and the summation is performed on all possible output sequences.

Equations (11.10) and (11.11) denote the elementary form of the linear chain CRF model, representing the conditional probability of the output sequence  $y$  with the given input sequence  $x$ . In these two equations,  $t_k$  is the feature function defined on the edge, called the transition feature, which depends on the current and previous positions;  $s_l$  is the feature function defined on the node, called the state feature, which depends on the current location. Both  $t_k$  and  $s_l$  depend on positions and are local feature functions. Generally, the feature functions  $t_k$  and  $s_l$  take the value 1 or 0; when the feature condition is satisfied, the value is 1, otherwise, it is 0. The Conditional Random Field is completely determined by feature functions  $t_k$ ,  $s_l$  and the corresponding weight  $\lambda_k, \mu_l$ .

The linear chain Conditional Random Field is also a log linear model.

Look at a simple example below.

**Example 11.1** We have a labeling problem. The input observation sequence is  $X = (X_1, X_2, X_3)$ , and the output labeling sequence is  $Y = (Y_1, Y_2, Y_3)$ , with  $Y_1, Y_2, Y_3$  taking values in  $\mathcal{Y} = \{1, 2\}$ .

Assume that the features  $t_k$ ,  $s_l$ , and the corresponding weights  $\lambda_k, \mu_l$  are as follows.

$$t_1 = t_1(y_{i-1} = 1, y_i = 2, x, i), \quad i = 2, 3, \quad \lambda_1 = 1$$

Only the condition that the feature value is 1 are noted here, and those that the value is 0 are omitted, i.e.,

$$t_1(y_{i-1}, y_i, x, i) = \begin{cases} 1, & y_{i-1} = 1, y_i = 2, x, i, (i = 2, 3), \\ 0, & \text{other conditions} \end{cases},$$

Same as below.

$$\begin{aligned} t_2 &= t_2(y_1 = 1, y_2 = 1, x, 2) & \lambda_2 &= 0.6 \\ t_3 &= t_3(y_2 = 2, y_3 = 1, x, 3) & \lambda_3 &= 1 \\ t_4 &= t_4(y_1 = 2, y_2 = 1, x, 2) & \lambda_4 &= 1 \\ t_5 &= t_5(y_2 = 2, y_3 = 2, x, 3) & \lambda_5 &= 0.2 \\ s_1 &= s_1(y_1 = 1, x, 1) & \mu_1 &= 1 \\ s_2 &= s_2(y_i = 2, x, i), i = 1, 2 & \mu_2 &= 0.5 \\ s_3 &= s_3(y_i = 1, x, i), i = 2, 3 & \mu_3 &= 0.8 \\ s_4 &= s_4(y_3 = 2, x, 3), & \mu_4 &= 0.5 \end{aligned}$$

For the given observation sequence  $x$ , find the denormalization conditional probability (i.e., the conditional probability that is not divided by the normalization factor) of the labeling sequence  $y = (y_1, y_2, y_3) = (1, 2, 2)$ .

**Solution** From Eq. (11.10), the linear chain CRF model is

$$P(y|x) \propto \exp \left[ \sum_{k=1}^5 \lambda_k \sum_{i=2}^3 t_k(y_{i-1}, y_i, x, i) + \sum_{k=1}^4 \mu_k \sum_{i=1}^3 s_k(y_i, x, i) \right]$$

For the given observation sequence  $x$ , the unnormalized conditional probability of the labeling sequence  $y = (1, 2, 2)$  is

$$P(y_1 = 1, y_2 = 2, y_3 = 2|x) \propto \exp(3.2)$$

### 11.2.3 The Simplified Form of Conditional Random Field

The Conditional Random Field can also be expressed in a simplified form. Noting that the same feature is defined in each position in the CRF equation (11.10), and the same feature can be summed at each position to convert the local feature function into a global feature function, so that the Conditional Random Field can be written as the inner product of the weight vector and the feature vector, i.e., the simplified form of the conditional random field.

For the sake of simplicity, the transition and state features and their weights are first represented by a unified symbol. Suppose there are  $K_1$  transition features and

$K_2$  state features,  $K = K_1 + K_2$ , and denote that

$$f_k(y_{i-1}, y_i, x, i) = \begin{cases} t_k(y_{i-1}, y_i, x, i), & k = 1, 2, \dots, K_1 \\ s_l(y_i, x, i), & k = K_1 + l; l = 1, 2, \dots, K_2 \end{cases} \quad (11.12)$$

The transition and state features are then summed at each position  $i$ , denoted as

$$f_k(y, x) = \sum_{i=1}^n f_k(y_{i-1}, y_i, x, i), \quad k = 1, 2, \dots, K \quad (11.13)$$

Denote the weight of the feature  $f_k(y, x)$  by  $w_k$ , i.e.,

$$w_k = \begin{cases} \lambda_k, & k = 1, 2, \dots, K_1 \\ \mu_l, & k = K_1 + l; l = 1, 2, \dots, K_2 \end{cases} \quad (11.14)$$

Thus, the Conditional Random Field represented by Eqs. (11.10) and (11.11) can be expressed as

$$P(y|x) = \frac{1}{Z(x)} \exp \sum_{k=1}^K w_k f_k(y, x) \quad (11.15)$$

$$Z(x) = \sum_y \exp \sum_{k=1}^K w_k f_k(y, x) \quad (11.16)$$

If the weight vector is denoted by  $w$ , i.e.,

$$w = (w_1, w_2, \dots, w_K)^T \quad (11.17)$$

and the global feature vector is denoted by  $F(y, x)$ , i.e.,

$$F(y, x) = (f_1(y, x), f_2(y, x), \dots, f_K(y, x))^T \quad (11.18)$$

then the CRF can be written as the inner product of the vector  $w$  and  $F(y, x)$ :

$$P_w(y|x) = \frac{\exp(w \cdot F(y, x))}{Z_w(x)} \quad (11.19)$$

where,

$$Z_w(x) = \sum_y \exp(w \cdot F(y, x)) \quad (11.20)$$

### 11.2.4 The Matrix Form of the Conditional Random Field

The conditional random field can also be represented by a matrix. Assume that  $P_w(y|x)$  is a linear chain CRF given by Eqs. (11.15) and (11.16), representing the conditional probability of the corresponding labeling sequence  $y$  for a given observation sequence  $x$ . Introduce a special start state label  $y_0 = \text{start}$  and a stop state label  $y_{n+1} = \text{stop}$  for each labeling sequence, and then the probability  $P_w(y|x)$  of the labeling sequence can be expressed in matrix form and computed effectively.

For each position  $i = 1, 2, \dots, n + 1$  of the observation sequence  $x$ , since  $y_{i-1}$  and  $y_i$  take values in  $m$  labels, an  $m$ -order matrix random variable can be defined:

$$M_i(x) = [M_i(y_{i-1}, y_i | x)] \quad (11.21)$$

The elements of the matrix random variable are

$$M_i(y_{i-1}, y_i | x) = \exp(W_i(y_{i-1}, y_i | x)) \quad (11.22)$$

$$W_i(y_{i-1}, y_i | x) = \sum_{k=1}^K w_k f_k(y_{i-1}, y_i, x, i) \quad (11.23)$$

Here  $w_k$  and  $f_k$  are given by Eqs. (11.14) and (11.12), respectively, and  $y_{i-1}$  and  $y_i$  are the values of labels labeling the random variables  $Y_{i-1}$  and  $Y_i$ .

In this way, with the given observation sequence  $x$ , the denormalization probability of the corresponding labeling sequence  $y$  can be represented by the product of the appropriate elements of the  $n + 1$  matrices in that sequence, i.e.,  $\prod_{i=1}^{n+1} M_i(y_{i-1}, y_i | x)$ . Thus, the conditional probability  $P_w(y|x)$  is

$$P_w(y|x) = \frac{1}{Z_w(x)} \prod_{i=1}^{n+1} M_i(y_{i-1}, y_i | x) \quad (11.24)$$

where  $Z_w(x)$  is the normalization factor and is the  $(\text{start}, \text{stop})$  element of the product of  $n + 1$  matrices, i.e.,

$$Z_w(x) = [M_1(x) M_2(x) \dots M_{n+1}(x)]_{\text{start}, \text{stop}} \quad (11.25)$$

Note that  $y_0 = \text{start}$  and  $y_{n+1} = \text{stop}$  indicate the start and stop states and the normalization factor  $Z_w(x)$  is the sum of the denormalization probabilities  $\prod_{i=1}^{n+1} M_i(y_{i-1}, y_i | x)$  of all paths  $y_1 y_2 \dots y_n$  that pass through the state with  $\text{start}$  as the starting point and  $\text{stop}$  as the endpoint. The following example illustrates this fact.

**Example 11.2** Give a linear chain Condition Random Filed, an observation sequence  $x$ , a state sequence  $y$ ,  $i = 1, 2, 3$ ,  $n = 3$ , and the label  $y_i \in \{1, 2\}$  as shown in

**Fig. 11.6** State paths

$$\begin{array}{llll} a_{01}b_{11}c_{11}, & a_{01}b_{11}c_{12}, & a_{01}b_{12}c_{21}, & a_{01}b_{12}c_{22} \\ a_{02}b_{21}c_{11}, & a_{02}b_{21}c_{12}, & a_{02}b_{22}c_{21}, & a_{02}b_{22}c_{22} \end{array}$$

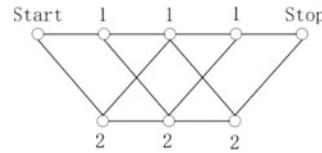


Fig. 11.6, and assume that  $y_0 = start = 1$ , and  $y_4 = stop = 1$ . The random matrices  $M_1(x), M_2(x), M_3(x), M_4(x)$  at each position are

$$\begin{aligned} M_1(x) &= \begin{bmatrix} a_{01} & a_{02} \\ 0 & 0 \end{bmatrix}, & M_2(x) &= \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ M_3(x) &= \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}, & M_4(x) &= \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \end{aligned}$$

Find the denormalization probabilities and the normalization factors of all paths of the state sequence  $y$  with *start* as the starting point and *stop* as the endpoint.

**Solution** First, compute the denormalization probability of each path from *start* to *stop* corresponding to  $y = (1, 1, 1)$ ,  $y = (1, 1, 2)$ ,  $\dots$ ,  $y = (2, 2, 2)$  in Fig. 11.6 as

$$\begin{aligned} &a_{01}b_{11}c_{11}, a_{01}b_{11}c_{12}, a_{01}b_{12}c_{21}, a_{01}b_{12}c_{22} \\ &a_{02}b_{21}c_{11}, a_{02}b_{21}c_{12}, a_{02}b_{22}c_{21}, a_{02}b_{22}c_{22} \end{aligned}$$

Then find the normalization factor according to Eq. (11.25). By computing the matrix product  $M_1(x)M_2(x)M_3(x)M_4(x)$ , we can see that the element of the first row and the first column is

$$\begin{aligned} &a_{01}b_{11}c_{11} + a_{02}b_{21}c_{11} + a_{01}b_{12}c_{21} + a_{02}b_{22}c_{22}, \\ &\quad + a_{01}b_{11}c_{12} + a_{02}b_{21}c_{12} + a_{01}b_{12}c_{22} + a_{02}b_{22}c_{21}, \end{aligned}$$

which is exactly equal to the sum of the denormalization probabilities of all paths from *start* to *stop*, i.e., the normalization factor  $Z(x)$ .

## 11.3 The Probability Computation Problem of Conditional Random Field

With a given Conditional Random Filed  $P(Y|X)$ , an input sequence  $x$ , and an output sequence  $y$ , the problem of computing the probability of the Conditional Random Filed is to calculate the conditional probability  $P(Y_{i-1} = y_i|x)$ ,  $P(Y_{i-1} = y_{i-1}, Y_i = y_i|x)$ , and the corresponding mathematical expectations. For convenience, the forward-backward vector is introduced to compute the above probability and mathematical expectations recursively, as in the Hidden Markov Model (HMM). Such an algorithm is also called the Forward-backward Algorithm.

### 11.3.1 Forward–Backward Algorithm

For each index  $i = 1, 2, \dots, n + 1$ , define the forward vector  $\alpha_i(x)$ :

$$\alpha_0(y|x) = \begin{cases} 1, & y = \text{start} \\ 0, & \text{otherwise} \end{cases} \quad (11.26)$$

The recursion formula is

$$\alpha_i^T(y_i|x) = \alpha_{i-1}^T(y_{i-1}|x)[M_i(y_{i-1}, y_i|x)], \quad i = 1, 2, \dots, n + 1 \quad (11.27)$$

which can be expressed as

$$\alpha_i^T(x) = \alpha_{i-1}^T(x)M_i(x) \quad (11.28)$$

$\alpha_i(y_i|x)$  represents the denormalization probability of the first part of the labeling sequence from 1 to  $i$  when the label at position  $i$  is  $y_i$ . There are  $m$  possible values for  $y_i$ , so  $\alpha_i(x)$  is an  $m$ -dimensional column vector.

Similarly, for each index  $i = 0, 1, 2, \dots, n + 1$ , define the backward vector  $\beta_i(x)$ :

$$\beta_{n+1}(y_{n+1}|x) = \begin{cases} 1, & y_{n+1} = \text{stop} \\ 0, & \text{otherwise} \end{cases} \quad (11.29)$$

$$\beta_i(y_i|x) = [M_{i+1}(y_i, y_{i+1}|x)]\beta_{i+1}(y_{i+1}|x) \quad (11.30)$$

And it can be expressed as

$$\beta_i(x) = M_{i+1}(x)\beta_{i+1}(x) \quad (11.31)$$

$\beta_i(y_i|x)$  represents the denormalization probability of the later part of the labeling sequence from  $i + 1$  to  $n$  that is labeled  $y_i$  at position  $i$ .

### 11.3.2 Probability Computation

Following the definition of the forward-backward vector, it's easy to compute the conditional probabilities of the labeling sequence  $y_i$  at position  $i$  and the labeling sequence  $y_{i-1}$  at position  $i - 1$ :

$$P(Y_i = y_i|x) = \frac{\alpha_i^T(y_i|x)\beta_i(y_i|x)}{Z(x)} \quad (11.32)$$

$$P(Y_{i-1} = y_{i-1}, Y_i = y_i|x) = \frac{\alpha_{i-1}^T(y_{i-1}|x)M_i(y_{i-1}, y_i|x)\beta_i(y_i|x)}{Z(x)} \quad (11.33)$$

where

$$Z(x) = \alpha_n^T(x)1 = 1^T\beta_1(x)$$

$1$  is an  $m$ -dimensional column vector with all elements being  $1$ .

### 11.3.3 The Computation of Expected Value

Using the forward-backward vector, the mathematical expectation of the feature function about the joint distribution  $P(X, Y)$  and the conditional distribution  $P(X|Y)$  can be computed.

The mathematical expectation of the feature function  $f_k$  about the conditional distribution  $P(X|Y)$  is

$$\begin{aligned} E_{P(Y|X)}[f_k] &= \sum_y P(y|x)f_k(y, x) \\ &= \sum_{i=1}^{n+1} \sum_{y_{i-1}y_i} f_k(y_{i-1}, y_i, x, i) \frac{\alpha_{i-1}^T(y_{i-1}|x)M_i(y_{i-1}, y_i|x)\beta_i(y_i|x)}{Z(x)} \\ &\quad k = 1, 2, \dots, K \end{aligned} \quad (11.34)$$

where,

$$Z(x) = \alpha_n^T(x)1$$

Assume that the empirical distribution is  $\tilde{P}(X)$ , the mathematical expectation of the feature function  $f_k$  about the joint distribution  $P(X, Y)$  is

$$\begin{aligned}
E_{P(X,Y)}[f_k] &= \sum_{x,y} P(x, y) \sum_{i=1}^{n+1} f_k(y_{i-1}, y_i, x, i) \\
&= \sum_x \tilde{P}(x) \sum_y P(y|x) \sum_{i=1}^{n+1} f_k(y_{i-1}, y_i, x, i) \\
&= \sum_x \tilde{P}(x) \sum_{i=1}^{n+1} \sum_{y_{i-1}y_i} f_k(y_{i-1}, y_i, x, i) \\
&\quad \frac{\alpha_{i-1}^T(y_{i-1}|x) M_i(y_{i-1}, y_i|x) \beta_i(y_i|x)}{Z(x)} \\
k &= 1, 2, \dots, K
\end{aligned} \tag{11.35}$$

where,

$$Z(x) = \alpha_n^T(x) \mathbf{1}$$

Equations (11.34) and (11.35) are the general formulas for the mathematical expectation of feature functions. For the transition feature  $t_k(y_{i-1}, y_i, x, i)$ ,  $k = 1, 2, \dots, K_1$ , the  $f_k$  in the formula can be replaced with  $t_k$ ; for state features, the  $f_k$  in the formula can be replaced with  $s_l$ , expressed as  $s_l(y_i, x, i)$ ,  $k = K_1 + l$ ,  $l = 1, 2, \dots, K_2$ .

With Eqs. (11.32)–(11.35), for a given observation sequence  $x$  and labeling sequence  $y$ ,  $\alpha_i$  and  $Z(x)$  can be computed by one forward scan, and  $\beta_i$  can be computed through one backward scan, thus computing the expectation of all probabilities and feature.

## 11.4 Learning Algorithms of Conditional Random Field

This section discusses the problem of estimating the parameters of the Conditional Random Field (CRF) model on a given training dataset, i.e., the learning of the Conditional Random Field. The CRF model is a log-linear model defined on time series data, and its learning methods include maximum likelihood estimation and regularized maximum likelihood estimation. Specific optimization implementation algorithms include the Improved Iterative Scaling (IIS), Gradient Descent, and the Quasi-Newton method (see Appendices A and B).

### 11.4.1 Improved Iterative Scaling

With the given training dataset, the empirical probability distribution  $\tilde{P}(X, Y)$  can be known. The model parameters can be obtained by maximizing the log-likelihood function of the training data.

The log likelihood function of the training data is

$$L(w) = L_{\tilde{P}}(P_w) = \log \prod_{x,y} P_w(y|x)^{\tilde{P}(x,y)} = \sum_{x,y} \tilde{P}(x,y) \log P_w(y|x)$$

When  $P_w$  is a Conditional Random Field model given by Eqs. (11.15) and (11.16), the log-likelihood function is

$$\begin{aligned} L(w) &= \sum_{x,y} \tilde{P}(x,y) \log P_w(y|x) \\ &= \sum_{x,y} \left[ \tilde{P}(x,y) \sum_{k=1}^K w_k f_k(y,x) - \tilde{P}(x,y) \log Z_w(x) \right] \\ &= \sum_{j=1}^N \sum_{k=1}^K w_k f_k(y_j, x_j) - \sum_{j=1}^N \log Z_w(x_j) \end{aligned}$$

The Improved Iterative Scaling (IIS) continuously optimizes the lower bound of change in the log-likelihood function by the iterative method to maximize the log-likelihood function. Assume that the current parameter vector of the model is  $w = (w_1, w_2, \dots, w_K)^T$ , the increment of the vector is  $\delta = (\delta_1, \delta_2, \dots, \delta_K)^T$ , and the updated parameter vector is  $w + \delta = (w_1 + \delta_1, w_2 + \delta_2, \dots, w_K + \delta_K)^T$ . In each iteration process, the IIS solves Eqs. (11.36) and (11.37) in turn to obtain  $\delta = (\delta_1, \delta_2, \dots, \delta_K)^T$ . The derivation can refer to Sect. 6.3.1 of this book.

The updated equation for the transition feature  $t_k$  is

$$\begin{aligned} E_{\tilde{P}}[t_k] &= \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \\ &= \sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k T(x, y)) \\ k &= 1, 2, \dots, K_1 \end{aligned} \tag{11.36}$$

The update equation for the state feature  $s_l$  is

$$E_{\tilde{P}}[s_l] = \sum_{x,y} \tilde{P}(x,y) \sum_{i=1}^{n+1} s_l(y_i, x, i)$$

$$= \sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} T(x, y)) \\ k = 1, 2, \dots, K_2 \quad (11.37)$$

Here,  $T(x, y)$  is the sum of all the features that appear in the data  $(x, y)$ :

$$T(x, y) = \sum_k f_k(y, x) = \sum_{k=1}^K \sum_{i=1}^{n+1} f_k(y_{i-1}, y_i, x, i) \quad (11.38)$$

**Algorithm 11.1** (*The Improved Iterative Scaling for Conditional Random Field Model Learning*)

Input: feature function  $t_1, t_2, \dots, t_{K_1}, s_1, s_2, \dots, s_{K_2}$ ; the empirical distribution  $\tilde{P}(X, Y)$ ;

Output: the parameter estimated value  $\hat{w}$ ; model  $P_{\hat{w}}$ .

- (1) For all  $k \in \{1, 2, \dots, K\}$ , take the initial value  $w_k = 0$ ;
- (2) For each  $k \in \{1, 2, \dots, K\}$ :

(a) When  $k=1, 2, \dots, K_1$ , let  $\delta_k$  be the solution of the equation

$$\sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k T(x, y)) = E_{\tilde{P}}[t_k]$$

When  $k = 1, 2, \dots, K_2$ , let  $\delta_{K_1+l}$  be the solution of the equation

$$\sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} T(x, y)) = E_{\tilde{P}}[s_l]$$

where  $T(x, y)$  is given by Eq. (11.38).

- (b) Update the value of  $w_k$ :  $w_k \leftarrow w_k + \delta_k$
- (3) If not all  $w_k$  converge, repeat step (2).

In Eqs. (11.36) and (11.37),  $T(x, y)$  represents the total number of features in the data  $(x, y)$ , which may take different values for different data  $(x, y)$ . To deal with this problem, define the relaxation feature

$$s(x, y) = S - \sum_{i=1}^{n+1} \sum_{k=1}^K f_k(y_{i-1}, y_i, x, i) \quad (11.39)$$

where  $S$  is a constant. Select a constant  $S$  large enough to make  $s(x, y) \geq 0$  hold for all data  $(x, y)$  in the training dataset. In this case, the total number of features can be taken as  $S$ .

From Eq. (11.36), for the transition feature  $t_k$ , the renewal equation of  $\delta_k$  is

$$\sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k S) = E_{\tilde{P}}[t_k] \quad (11.40)$$

$$\delta_k = \frac{1}{S} \log \frac{E_{\tilde{P}}[t_k]}{E_P[t_k]} \quad (11.41)$$

where

$$E_P(t_k) = \sum_x \tilde{P}(x) \sum_{i=1}^{n+1} \sum_{y_{i-1}, y_i} t_k(y_{i-1}, y_i, x, i) \frac{a_{i-1}^T(y_{i-1}|x) M_i(y_{i-1}, y_i|x) \beta_i(y_i|x)}{Z(x)} \quad (11.42)$$

Similarly, from Eq. (11.37), the renewal equation of  $\delta_k$  for the state feature  $s_l$  is

$$\sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} S) = E_{\tilde{P}}[s_l] \quad (11.43)$$

$$\delta_{K_1+l} = \frac{1}{S} \log \frac{E_{\tilde{P}}[s_l]}{E_P[s_l]} \quad (11.44)$$

where

$$E_P(s_l) = \sum_x \tilde{P}(x) \sum_{i=1}^n \sum_{y_i} s_l(y_i, x, i) \frac{\alpha_i^T(y_i|x) \beta_i(y_i|x)}{Z(x)} \quad (11.45)$$

The above algorithm is called the  $S$ -algorithm. In the  $S$ -algorithm, we need to make the constant  $S$  large enough so that the incremental vector at each iteration step will increase and the algorithm convergence will be slower. The  $T$ -algorithm tries to solve this problem and computes the maximum value of the total number of features,  $T(x)$ , for each observation sequence  $x$ .

$$T(x) = \max_y T(x, y) \quad (11.46)$$

$T(x) = t$  can be easily calculated by using the forward-backward recursive formula.

At this time, the updated equation about the transition feature parameters can be written as follows:

$$\begin{aligned}
E_{\tilde{P}}[t_k] &= \sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k T(x)) \\
&= \sum_x \tilde{P}(x) \sum_y P(y|x) \sum_{i=1}^{n+1} t_k(y_{i-1}, y_i, x, i) \exp(\delta_k T(x)) \\
&= \sum_x \tilde{P}(x) a_{k,t} \exp(\delta_k t) \\
&= \sum_{t=0}^{T_{\max}} a_{k,t} \beta_k^t
\end{aligned} \tag{11.47}$$

Here,  $a_{k,t}$  is the expected value of the feature  $t_k$ ,  $\delta_k = \log \beta_k$ .  $\delta_k$  is the only real root of the polynomial Eq. (11.47) and can be obtained by Newton's method. The associated  $\delta_k$  is thus obtained.

Similarly, the parameter updating equation for state features can be written as follows:

$$\begin{aligned}
E_{\tilde{P}}[s_l] &= \sum_{x,y} \tilde{P}(x) P(y|x) \sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} T(x)) \\
&= \sum_x \tilde{P}(x) \sum_y P(y|x) \sum_{i=1}^n s_l(y_i, x, i) \exp(\delta_{K_1+l} T(x)) \\
&= \sum_x \tilde{P}(x) b_{l,t} \exp(\delta_k t) \\
&= \sum_{t=0}^{T_{\max}} b_{l,t} \gamma_l^t
\end{aligned} \tag{11.48}$$

Here,  $b_{l,t}$  is the expected value of the feature  $s_l$ ,  $\delta_l = \log \gamma_l$ , and  $\gamma_l$  is the only real root of the polynomial Eq. (11.48), which can also be obtained by Newton's method.

### 11.4.2 Quasi-Newton Method

Newton's method or the Quasi-Newton method can also be used in Conditional Random Field (CRF) model learning (see Appendix B). For the CRF model

$$P_w(y|x) = \frac{\exp\left(\sum_{i=1}^n w_i f_i(x, y)\right)}{\sum_y \exp\left(\sum_{i=1}^n w_i f_i(x, y)\right)} \tag{11.49}$$

the optimized objective function of learning is

$$\min_{w \in R^n} f(w) = \sum_x \tilde{P}(x) \log \sum_y \exp \left( \sum_{i=1}^n w_i f_i(x, y) \right) - \sum_{x, y} \tilde{P}(x, y) \sum_{i=1}^n w_i f_i(x, y) \quad (11.50)$$

and its gradient function is

$$g(w) = \sum_{x, y} \tilde{P}(x) P_w(y|x) f(x|y) - E_{\tilde{P}}(f) \quad (11.51)$$

The BFGS algorithm of the Quasi-Newton method is as follows.

**Algorithm 11.2** (*The BFGS algorithm for the Conditional Random Field Model Learning*)

Input: feature function  $f_1, f_2, \dots, f_n$ ; the empirical distribution  $\tilde{P}(X, Y)$ ;

Output: the optimal parameter value  $\hat{w}$ ; the optimal model  $P_{\hat{w}}(y|x)$ .

- (1) Select the initial point  $w^{(0)}$ , take  $B_0$  as the positive definite symmetric matrix, and set  $k = 0$ .
- (2) Compute  $g_k = g(w^{(k)})$ . If  $g_k = 0$ , the computation stops; otherwise, turn to (3).
- (3) From  $B_k p_k = -g_k$  to  $p_k$ .
- (4) One dimensional search (Line Search): find  $\lambda_k$  such that

$$f(w^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(w^{(k)} + \lambda p_k)$$

- (5) Set  $w^{(k+1)} = w^{(k)} + \lambda_k p_k$ .
- (6) Compute  $g_{k+1} = g(w^{(k+1)})$ . If  $g_{k+1} = 0$ , stop the computation; otherwise, find  $B_{k+1}$  by the following formula:

$$B_{k+1} = B_k + \frac{y_k y_{kl}^T}{y_k^T \delta_k} - \frac{B_k \delta_k \delta_k^T B_k}{\delta_k^T B_k \delta_k}$$

where

$$y_k = g_{k+1} - g_k, \delta_k = w^{(k+1)} - w^{(k)}$$

- (7) Set  $k = k + 1$ , and turn to (3).

## 11.5 The Prediction Algorithm of Conditional Random Field

The prediction problem of the Conditional Random Field is to find the output sequence (labeling sequence)  $y^*$ , which has the highest conditional probability, given the Conditional Random Field  $P(Y|X)$ , and the input sequence (observation sequence)  $x$ . It is to label the observation sequence. The CRF prediction algorithm is the well-known Viterbi algorithm (see Sect. 10.4 of this book).

From Eq. (11.19), we can get:

$$\begin{aligned} y^* &= \arg \max_y P_w(y|x) \\ &= \arg \max_y \frac{\exp(w \cdot F(y, x))}{Z_w(x)} \\ &= \arg \max_y \exp(w \cdot F(y, x)) \\ &= \arg \max_y (w \cdot F(y, x)) \end{aligned}$$

Thus, the problem of predicting the Conditional Random Field becomes the problem of finding the optimal path with the largest denormalization probability.

$$\max_y (w \cdot F(y, x)) \quad (11.52)$$

Here, the path represents the sequence of labels, and

$$\begin{aligned} w &= (w_1, w_2, \dots, w_K)^T \\ F(y, x) &= (f_1(y, x), f_2(y, x), \dots, f_K(y, x))^T \\ f_k(y, x) &= \sum_{i=1}^n f_k(y_{i-1}, y_i, x, i), \quad k = 1, 2, \dots, K \end{aligned}$$

Note that only the denormalization, instead of the probability, is computed at this point, which can improve the efficiency greatly. To solve for the optimal path, Formula (11.52) is written as follows:

$$\max_y \sum_{i=1}^n w \cdot F_i(y_{i-1}, y_i, x) \quad (11.53)$$

where,

$$F_i(y_{i-1}, y_i, x) = (f_1(y_{i-1}, y_i, x, i), f_2(y_{i-1}, y_i, x, i), \dots, f_K(y_{i-1}, y_i, x, i))^T$$

is the local feature vector.

The Viterbi algorithm is described below. First, find the denormalization probability of each label  $j = 1, 2, \dots, m$  at position 1:

$$\delta_1(j) = w \cdot F_1(y_0 = \text{start}, y_1 = j, x), \quad j = 1, 2, \dots, m \quad (11.54)$$

Generally, by the recursive formula, find the maximum value of the denormalization probability of each label  $l = 1, 2, \dots, m$  to the position  $i$ , and record the path of the maximum denormalization probability:

$$\delta_i(l) = \max_{1 \leq j \leq m} \{\delta_{i-1}(j) + w \cdot F_i(y_{i-1} = j, y_i = l, x)\}, \quad l = 1, 2, \dots, m \quad (11.55)$$

$$\Psi_i(l) = \arg \max_{1 \leq j \leq m} \{\delta_{i-1}(j) + w \cdot F_i(y_{i-1} = j, y_i = l, x)\}, \quad l = 1, 2, \dots, m \quad (11.56)$$

It ends when  $i = n$ . At this time, the maximum value of the denormalization probability is

$$\max_y (w \cdot F(y, x)) = \max_{1 \leq j \leq m} \delta_n(j) \quad (11.57)$$

and the endpoint of the optimal path is

$$y_n^* = \arg \max_{1 \leq j \leq m} \delta_n(j) \quad (11.58)$$

Return from the endpoint of the optimal path,

$$y_i^* = \Psi_{i+1}(y_{i+1}^*), \quad i = n - 1, n - 2, \dots, 1 \quad (11.59)$$

Find the optimal path  $y^* = (y_1^*, y_2^*, \dots, y_n^*)^T$ .

In summary, the Viterbi algorithm for Conditional Random Field prediction is obtained.

### **Algorithm 11.3 (Viterbi algorithm for conditional random field prediction)**

Input: the model feature vector  $F(y, x)$ , the weight vector  $w$ , the observation sequence  $x = (x_1, x_2, \dots, x_n)$ ;

Output: the optimal path  $y^* = (y_1^*, y_2^*, \dots, y_n^*)^T$ .

#### 1. Start

$$\delta_1(j) = w \cdot F_1(y_0 = \text{start}, y_1 = j, x), \quad j = 1, 2, \dots, m$$

#### 2. Recursion. For $i = 2, 3, \dots, n$

$$\delta_i(l) = \max_{1 \leq j \leq m} \{\delta_{i-1}(j) + w \cdot F_i(y_{i-1} = j, y_i = l, x)\}, \quad l = 1, 2, \dots, m$$

$$\Psi_i(l) = \arg \max_{1 \leq j \leq m} \{\delta_{i-1}(j) + w \cdot F_i(y_{i-1} = j, y_i = l, x)\}, \quad l = 1, 2, \dots, m$$

3. End

$$\max_y (w \cdot F(y, x)) = \max_{1 \leq j \leq m} \delta_n(j)$$

$$y_n^* = \arg \max_{1 \leq j \leq m} \delta_n(j)$$

4. Return to the path

$$y_n^* = \Psi_{i+1}(y_{i+1}^*), \quad i = n-1, n-2, \dots, 1$$

Find the optimal path  $y^* = (y_1^*, y_2^*, \dots, y_n^*)^T$ .

The following is an example to illustrate the Viterbi algorithm.

**Example 11.3** In Example 11.1, use the Viterbi algorithm to find the optimal output sequence (labeling sequence)  $y^* = (y_1^*, y_2^*, y_3^*)$  corresponding to the given input sequence (observation sequence)  $x$ .

**Solution** The feature function and the corresponding weight are given in Example 11.1.

Now use the Viterbi algorithm to find the optimal path:

$$\max \sum_{i=1}^3 w \cdot F_i(y_{i-1}, y_i, x)$$

1. Start

$$\delta_1(j) = w \cdot F_1(y_0 = \text{start}, y_1 = j, x), \quad j = 1, 2$$

$$i = 1, \quad \delta_1(1) = 1, \quad \delta_1(2) = 0.5$$

2. Recursion

$$i = 2 \quad \delta_2(l) = \max_j \{\delta_1(j) + w \cdot F_2(j, l, x)\}$$

$$\delta_2(1) = \max\{1 + \lambda_2 t_2 + \mu_3 s_3, 0.5 + \lambda_4 t_4 + \mu_3 s_3\} = 2.4, \quad \Psi_2(1) = 1$$

$$\delta_2(2) = \max\{1 + \lambda_1 t_1 + \mu_2 s_2, 0.5 + \mu_2 s_2\} = 2.5, \quad \Psi_2(2) = 1$$

$$i = 3 \quad \delta_3(l) = \max_j \{\delta_2(j) + w \cdot F_3(j, l, x)\}$$

$$\delta_3(1) = \max\{2.4 + \mu_3 s_3, 2.5 + \lambda_3 t_3 + \mu_3 s_3\} = 4.3, \quad \Psi_3(1) = 2$$

$$\delta_3(2) = \max\{2.4 + \lambda_1 t_1 + \mu_4 s_4, 2.5 + \lambda_5 t_5 + \mu_4 s_4\} = 3.9, \quad \Psi_3(2) = 1$$

3. End

$$\max_y (w \cdot F(y, x)) = \max \delta_3(l) = \delta_3(1) = 4.3$$

$$y_3^* = \arg \max_l \delta_3(l) = 1$$

4. Return

$$\begin{aligned} y_2^* &= \Psi_3(y_3^*) = \Psi_3(1) = 2 \\ y_1^* &= \Psi_2(y_2^*) = \Psi_2(2) = 1 \end{aligned}$$

The optimal labeling sequence

$$y^* = (y_1^*, y_2^*, y_3^*) = (1, 2, 1)$$

## Summary

1. The probabilistic undirected graphical model is a joint probability distribution represented by an undirected graph. The connection relationship between nodes in the undirected graph represents the conditional independence between the sets of random variables of joint distribution, i.e., the Markov property. Therefore, the probabilistic undirected graphical model is also called Markov Random Field.  
The joint probability distribution of the probabilistic undirected graphical model or Markov Random Field can be decomposed into the product of positive functions on the maximal cliques of the undirected graph.
2. The Conditional Random Field (CRF) is a conditional probability distribution model of the output random variable  $Y$  with the given input random variable  $X$ . It is in the form of a parameterized log-linear model. The most important feature of the CRF is the assumption that the joint probability distribution among output variables constitutes a probabilistic undirected graphical model, i.e., the Markov Random Field. The Conditional Random Fields is a discriminative model.
3. The linear chain Conditional Random Field is defined on observation sequences and labeling sequences. It is generally expressed as the conditional probability distribution of the labeling sequence with the given observation sequence and represented by a parameterized log-linear model. The model contains features and corresponding weights. Features are defined on the edges and nodes of the linear chain. The parameterized form is the most basic form of the linear chain CRF model, which has other forms like simplification and deformation. The mathematical expression of the parameterized form is

$$P(y|x) = \frac{1}{Z(x)} \exp \left( \sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i) \right)$$

where

$$Z(x) = \sum_y \exp\left(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} \mu_l s_l(y_i, x, i)\right)$$

4. The forward-backward algorithm is usually used to compute the probability of the linear chain Conditional Random Field.
5. The learning method of the Conditional Random Field is usually the maximum likelihood estimation or the regularized maximum likelihood estimation, i.e., under the given training data, the model parameters are estimated by maximizing the log-likelihood function of the training data. The specific algorithms include the Improved Iterative Scaling (IIS) algorithm, Gradient Descent method, Quasi-Newton method, etc.
6. An important application of the linear chain CRF is labeling. The Viterbi algorithm is a method to find the labeling sequence with the maximum conditional probability given the observation sequence.

### Further Reading

See Refs. [1, 2] for more information about the probabilistic undirected graphical model. For details of the Conditional Random Fields, refer to Refs. [3, 4]. Models such as the Maximum Entropy Markov Model were proposed before the conditional random field was proposed [5]. The Conditional Random Field can be regarded as the generalization of the Maximum Entropy Markov Model in the labeling problem. The Support Vector Machine model has also been extended to the labeling problem [6, 7].

### Exercises

- 11.1 Write the factorization equation of the probabilistic graphical model described by the undirected graph in Fig. 11.3.
- 11.2 Prove that  $Z(x) = \alpha_n^T(x)\mathbf{1} = \mathbf{1}^T\beta_1(x)$ , where  $\mathbf{1}$  is an  $m$ -dimensional sequence vector whose elements are all 1.
- 11.3 Write down the Gradient Descent method for conditional Random Field model learning.
- 11.4 Referring to the state path diagram in Fig. 11.6, assume that the random matrices  $M$   $M_1(x)$ ,  $M_2(x)$ ,  $M_3(x)$ ,  $M_4(x)$  are

$$\begin{aligned} M_1(x) &= \begin{bmatrix} 0 & 0 \\ 0.5 & 0.5 \end{bmatrix} & M_2(x) &= \begin{bmatrix} 0.3 & 0.7 \\ 0.7 & 0.3 \end{bmatrix} \\ M_3(x) &= \begin{bmatrix} 0.5 & 0.5 \\ 0.6 & 0.4 \end{bmatrix} & M_4(x) &= \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Find the probability of the state sequence  $y$  of all paths starting at start = 2 and ending at stop = 2 and the state sequence with the largest probability.

## References

1. Bishop M. Pattern recognition and machine learning. Springer; 2006.
2. Koller D, Friedman N. Probabilistic graphical models: principles and techniques. MIT Press; 2009.
3. Lafferty J, McCallum A, Pereira F. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: International conference on machine learning; 2001.
4. Sha F, Pereira F. Shallow parsing with conditional random fields. In: Proceedings of the 2003 conference of the North American Chapter of Association for Computational Linguistics on Human Language Technology, vol. 1; 2003.
5. McCallum A, Freitag D, Pereira F. Maximum entropy Markov models for information extraction and segmentation. In: Proceedings of the international conference on machine learning; 2000.
6. Taskar B, Guestrin C, Koller D. Max-margin Markov networks. In: Proceedings of the NIPS 2003; 2003.
7. Tsochantaridis I, Hofmann T, Joachims T. Support vector machine learning for inter-dependent and structured output spaces. In: ICML; 2004.

# Chapter 12

## Summary of Supervised Learning Methods

The first part of the book introduces ten dominant machine learning methods, all of which are supervised learning methods. They are the perceptron,  $k$ -Nearest-Neighbor ( $k$ -NN), the Naïve Bayes method, the decision tree, logistic regression and maximum entropy model, Support Vector Machine (SVM), Boosting, the EM algorithm, Hidden Markov Model (HMM), and Conditional random field (CRF). The characteristics of these ten supervised learning methods are summarized in Table 12.1.

The characteristics and relationships of each method are briefly discussed in the following sections.

### 12.1 Application

Part one of this book is an introduction to supervised learning methods. Supervised learning can be thought of as learning a model so that it can predict the corresponding output for a given input. It includes classification, labeling, and regression. The first part of this book focuses on the learning methods of the first two. The classification problem is the prediction problem from the instance's feature vector to the class label, and the labeling problem is the prediction problem from the observation sequences to the labeling sequences (or state sequences). It can be considered that the classification problem is a particular case of the labeling problem. In a classification problem, the possible predictions are binary classes or multi-classes. The possible prediction results in the labeling problem are all the labeling sequences, of which the number is exponential.

Perceptron,  $k$ -Nearest Neighbor, Naïve Bayes method, the decision tree, logistic regression and maximum entropy model, Support Vector Machine (SVM), and Boosting are classification methods. The original perceptron, SVM, and Boosting are for binary classification and can be extended to multi-class classification. Hidden Markov Model (HMM), and Conditional random

**Table 12.1** Summary of 10 supervised learning methods

Method	Application	Model feature	Model type	Learning strategy	Loss function for learning	Learning algorithm
Perceptron	Binary classification	The separating hyperplane	Discriminative model	Minimizing the distance from the misclassified point to the hyperplane	Distance from misclassified point to the hyperplane	Stochastic gradient descent
<i>k</i> -Nearest-Neighbor	Multi-class classification & regression	Feature space & sample point	Discriminative model	—	—	—
Naïve Bayes method	Multi-class classification	Joint probability distribution of features and class, conditional independence assumption	Generative model	Maximum likelihood estimation, maximum posterior probability estimation	Log-likelihood loss function	Probability calculation formula, the EM algorithm
Decision tree	Multi-class classification & regression	Classification tree & decision tree	Discriminative model	The regularized maximum likelihood estimation	Log-likelihood loss function	Feature selection, generation, pruning
Logistic regression and maximum entropy model	Multi-class classification	Conditional probability distributions of classes under feature conditions, log-linear models	Discriminative model	Maximum likelihood estimation, the regularized maximum likelihood estimation	Logistic loss	Improved iterative scaling algorithm, Gradient Descent the Quasi-Newton method

(continued)

**Table 12.1** (continued)

Method	Application	Model feature	Model type	Learning strategy	Loss function for learning	Learning algorithm
Support Vector Machine	Binary classification	The separating hyperplane & the kernel trick	Discriminative model	Minimizing the regularized hinge loss, soft margin maximization	Hinge loss	Sequential minimal optimization (SMO)
Boosting	Binary classification	Linear combination of weak classifiers	Discriminative model	Minimizing the exponential loss of the additive model	Exponential loss	Forward stagewise addition algorithm
EM algorithm <sup>1</sup>	Parameter estimation of probabilistic models	Probabilistic models with hidden models	–	Maximum likelihood estimation, maximum posterior probability estimation	Log-likelihood loss function	Iterative algorithms

(continued)

<sup>1</sup> The EM algorithm is somewhat special here, as it is a general method and does not have a specific model.

**Table 12.1** (continued)

Method	Application	Model feature	Model type	Learning strategy	Loss function for learning	Learning algorithm
Hidden Markov Model	Labeling	Joint probability distribution model of observation sequence and state sequence	Generative model	Maximum likelihood estimation, maximum posterior probability estimation	Log-likelihood loss function	Probability calculation formula, the EM algorithm
Condition Random Field	Labeling	Conditional probability distribution of observation sequence under the condition of state sequence, log-linear model	Discriminative model	Maximum likelihood estimation, the regularized maximum likelihood estimation	Log-likelihood loss function	Improved iterative scaling algorithm, Gradient Descent, the Quasi-Newton method

field (CRF) are labeling methods. The EM algorithm is a general learning algorithm of a probability model containing hidden variables, which can be used for unsupervised learning of the generative model.

Perceptron,  $k$ -NN, the Naïve Bayes method, and decision tree are simple classification methods, characteristic of their intuitive models, simple methods, and easy implementation. Logistic regression and maximum entropy model, SVM, and Boosting are more complex but more effective classification methods, which tend to have higher classification accuracy. Hidden Markov Model (HMM), and Conditional random field (CRF) are the dominant labeling methods. Generally speaking, the labeling by CRF is more accurate.

## 12.2 Models

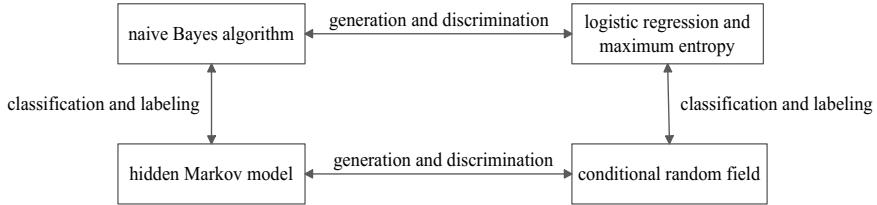
The prediction models for both classification and labeling can be considered as the mapping from the input space to the output space. They can be written in the form of a conditional probability distribution  $P(Y|X)$  or a decision function  $Y = f(X)$ . The former represents a probabilistic model of the output under given input, and the latter represents a non-probabilistic input–output model. Sometimes the model is more directly expressed as a probabilistic model or a non-probabilistic model, but sometimes the model is interpreted in both ways.

The Naïve Bayes method and Hidden Markov Model (HMM) are probabilistic models. Perceptron,  $k$ -Nearest Neighbor ( $k$ -NN), Support Vector Machine (SVM), and Boosting are non-probabilistic models. The decision tree, logistic regression and maximum entropy model, and Conditional Random Field (CRF) can be regarded as both probabilistic and non-probabilistic models.

Methods that directly learn the conditional probability distribution  $P(Y|X)$  or the decision function  $Y = f(X)$  is the discriminant method, and the corresponding model is the discriminative model. Perceptron,  $k$ -NN, decision tree, logistic regression and maximum entropy model, SVM, Boosting, and CRF are discriminative methods. The method of learning the joint probability distribution  $P(X, Y)$  to obtain the conditional probability distribution  $P(Y|X)$  is the generative method, whose corresponding model is named the generative model. The Naïve Bayes method and HMM are generative methods. Figure 12.1 shows the relationship between some of these models.

Unsupervised learning methods can be used to learn generative models. Specifically, the EM algorithm can be applied to the learning of the Naïve Bayes model and the hidden Markov model.

The decision tree is defined on a general feature space and can contain either continuous or discrete variables. The feature space of the perceptron, SVM, and  $k$ -NN is Euclidean space (more generally, Hilbert space). The model of the boosting method is a linear combination of weak classifiers, and the feature space of the weak classifier is the feature space of the boosting model.



**Fig. 12.1** Relationship between some models

The perceptron model is a linear model, while the logistic regression and maximum entropy model, and CRF are log-linear models. The  $k$ -NN method, decision tree, SVM (including kernel function), and Boosting use nonlinear models.

Figure 12.1 depicts the relationship between several machine learning methods in terms of generation and discrimination, and classification and labeling.

### 12.3 Learning Strategies

In supervised learning for binary classification, SVM, logistic regression and maximum entropy model, and Boosting use hinge loss function, logistic loss function, and exponential loss function, respectively. The three loss functions are written as

$$[1 - yf(x)]_+ \quad (12.1)$$

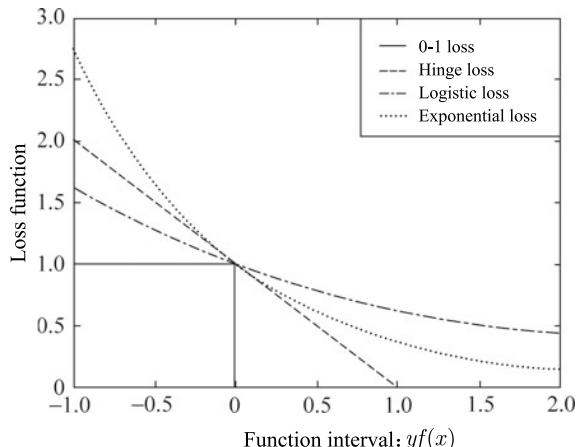
$$\log[1 + \exp(-yf(x))] \quad (12.2)$$

$$\exp(-yf(x)) \quad (12.3)$$

These three loss functions are all the upper bounds of the 0–1 loss function and have similar shapes, as shown in Fig. 12.2. Therefore, it can be considered that SVM, logistic regression and maximum entropy model, and Boosting use different surrogate loss functions to represent the loss of classification, and define empirical risk or structural risk function to achieve binary classification learning tasks. The learning strategy is to optimize the following structural risk function:

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f) \quad (12.4)$$

**Fig. 12.2** The relationship among 0–1 loss function, hinge loss function, logistic loss function, and exponential loss function



Here, the first term is the empirical risk (empirical loss), the second term is the regularization term,  $L(y, f(x))$  is the loss function,  $J(f)$  is the complexity of the model, and  $\lambda \geq 0$  is the coefficient.

Support Vector Machine denotes the model's complexity with the  $L_2$  norm. The original logistic regression and maximum entropy models do not have a regularization term and can be regularized by adding the  $L_2$  norm. The boosting method does not have an explicit regularization term and is usually regularized by early stopping.

The above binary classification learning methods can be extended to multi-class classification learning and labeling problems. For example, the Conditional Random Field for labeling can be regarded as a generalization of the maximum entropy model of the classification problem.

The learning of the probability model can be formalized as maximum likelihood estimation or the maximum posterior probability estimation of Bayesian estimation.

At this time, the learning strategy is to minimize the log-likelihood loss or minimize the regularized log-likelihood loss. The log-likelihood loss can be written as

$$-\log P(y|x)$$

When estimating the maximum posterior probability, the regularization term is the negative logarithm of the prior probability.

The strategy of decision tree learning is the regularized maximum likelihood estimation, the loss function is the log-likelihood loss, and the regularization term is the complexity of the decision tree.

The learning strategies of logistic regression and maximum entropy model and Conditional Random Field can be regarded as maximum likelihood estimation (or regularized maximum likelihood estimation). Moreover, they also can be regarded as minimizing logistic loss (or the regularized logistic loss).

The unsupervised learning of the Naïve Bayes model and the hidden Markov model is also maximum likelihood estimation or maximum posterior probability estimation. However, the model contains hidden variables at this time.

## 12.4 Learning Algorithms

Once the machine learning has a concrete form, it becomes an optimization problem. Sometimes, optimization problems are simple, and an analytical solution exists, i.e., the optimal solution can be simply computed by the formula. In most cases, however, there is no analytical solution, and the problem needs to be solved numerically or by heuristic methods.

In the supervised learning of the Naïve Bayes method and the hidden Markov model, the optimal solution is the maximum likelihood estimation value, which can be directly computed by the probability calculation formula.

The learning of perceptron, logistic regression and maximum entropy model, and CRF use methods such as gradient descent method, Quasi-Newton method, etc. These are general solutions to unconstrained optimization problems.

SVM learning, which can solve the dual problem of convex quadratic programming, is based on methods such as the sequential minimal optimization algorithm.

Decision tree learning is a typical example based on heuristic algorithms. It can be considered that feature selection, generation, and pruning are heuristically regularized maximum likelihood estimations.

The boosting method takes advantage of the fact that the learning model is additive, and the loss function is an exponential loss function. Heuristically, the model is learned step by step from front to back to achieve the goal of approximating and optimizing the objective function.

The EM algorithm is an iterative method for solving the parameters of the probability model with hidden variables. Its convergence can be guaranteed, but it can not guarantee convergence to the global optimum.

SVM learning, logistic regression and maximum entropy model learning, and CRF learning are convex optimization problems, and the globally optimal solution is guaranteed to exist. Other learning problems are not convex optimization problems.

# Chapter 13

## Introduction to Unsupervised Learning

Part two of this book deals with unsupervised learning methods in statistical learning or machine learning. Unsupervised learning is the machine learning problem of learning models from unlabeled data and is an important part of machine learning.

This chapter is an overview of unsupervised learning, first describing the basic principles of unsupervised learning, followed by the basic problems and fundamental methods of unsupervised learning. The basic problems include clustering, dimensionality reduction, topic modeling, and graph analytics.

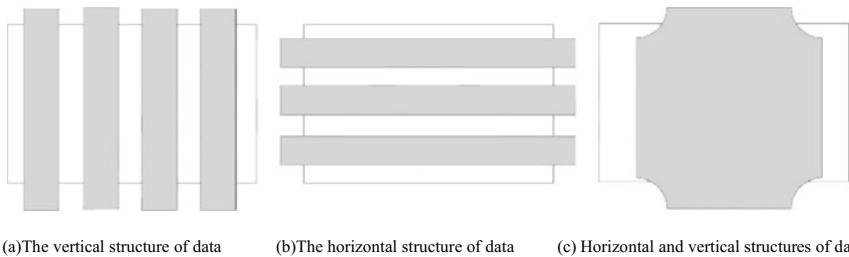
### 13.1 The Fundamentals of Unsupervised Learning

Unsupervised learning is machine learning to learn the statistical laws or internal structure of data from unlabeled data, which mainly includes clustering, dimensionality reduction, and probability estimation. Unsupervised learning can be used for data analysis or pre-processing of supervised learning.

Unsupervised learning uses unlabeled data  $U = \{x_1, x_2, \dots, x_N\}$  to learn or train, where  $x_i, i = 1, 2, \dots, N$  are samples (instances) consisting of feature vectors. The model for unsupervised learning is the function  $z = g_\theta(x)$  with the conditional probability distribution  $P_\theta(z|x)$  or conditional probability distribution  $P_\theta(x|z)$ . Here,  $x \in X$  is the input, representing the sample;  $z \in Z$  is the output, representing the result of the analysis of the sample, which can be a category, transformation, or probability; and  $\theta$  is the parameter.

Assume that the training dataset consists of  $N$  samples, each of which is an  $m$ -dimensional vector. The training data can be represented by a matrix where each row corresponds to a feature, and each column corresponds to a sample.

$$X = \begin{bmatrix} x_{11} & \dots & x_{1N} \\ \dots & & \dots \\ x_{M1} & \dots & x_{MN} \end{bmatrix}$$



(a)The vertical structure of data      (b)The horizontal structure of data      (c) Horizontal and vertical structures of data

**Fig. 13.1** Basic idea of unsupervised learning **a** the vertical structure of data **b** the horizontal structure of data **c** horizontal and vertical structures of data

where  $x_{ij}$  is the  $i$ -th dimension of the  $j$ -th vector;  $i = 1, 2, \dots, M$ ;  $j = 1, 2, \dots, N$ .

Unsupervised learning is not easy because the data is not labeled, i.e., there is no human guidance, and the machine needs to figure out the patterns from the data itself. The input  $x$  of the model is observable in the data, while the output  $z$  is hidden in the data. Unsupervised learning usually requires a large amount of data because discovering hidden patterns in the data requires sufficient observations.

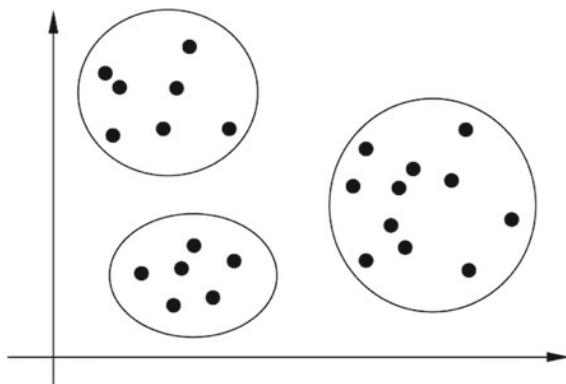
The basic idea of unsupervised learning is to “compress” the given data (matrix data) in order to find the underlying structure of the data. It is assumed that the result of compression with minimal loss is the most intrinsic structure. Figure 13.1 is a schematic diagram of this idea. We can consider exploring the vertical structure of the data and clustering similar samples into the same type, i.e., clustering the data. We can also attempt to uncover the horizontal structure of the data and convert vectors in the high-dimensional space into vectors in the low-dimensional space, i.e., reducing the dimensionality of the data. It is also possible to discover the vertical and horizontal structure of the data simultaneously, assuming that the data is generated by a probabilistic model with an implicit structure, and this model can be learned from the data.

## 13.2 Basic Issues

### 13.2.1 Clustering

Clustering is assigning similar samples (instances) in a sample set to the same class and assigning dissimilar ones to different classes. In clustering, the samples are usually vectors in Euclidean space, and the classes are not given in advance but discovered automatically from the data. The number of classes is usually given in advance. The similarity or distance between samples is determined by the application. If a sample belongs to only one class, it is called hard clustering; if a sample belongs to more than one class, it is called soft clustering. Figure 13.2 gives an example of

**Fig. 13.2** Example of clustering



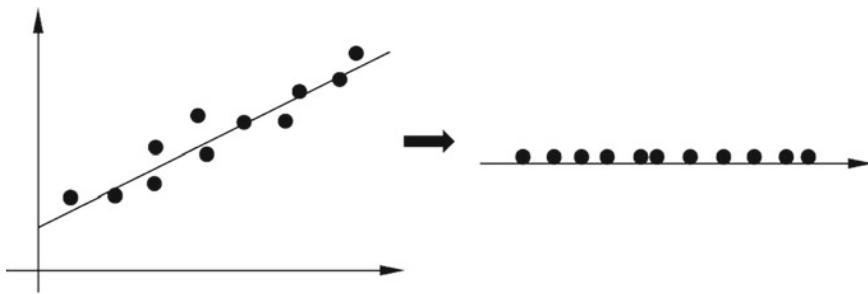
clustering (hard clustering). The samples in the two-dimensional space are divided into three different classes.

Suppose the input space is the Euclidean space  $X \subseteq R^d$  and the output space is the set of classes  $Z = \{1, 2, \dots, k\}$ . The clustering model is the function  $z = g_\theta(x)$  or the conditional probability distribution  $P_\theta(z|x)$ , where  $x \in X$  is the vector of samples,  $z \in Z$  is the class of samples, and  $\theta$  is a parameter. The former function is a hard clustering model, and the latter conditional probability distribution is a soft clustering model.

The process of clustering is the process of learning a clustering model. In hard clustering, each sample belongs to a class  $z_i = g_\theta(x_i)$ ,  $i = 1, 2, \dots, N$ ; in soft clustering, each sample belongs to each class  $P_\theta(z_i|x_i)$ ,  $i = 1, 2, \dots, N$  according to probability. As shown in Fig. 13.1, clustering can help discover the hidden vertical structure in the data. (There are exceptions. Co-clustering is a clustering algorithm that clusters both samples and features while discovering the vertical and horizontal structure in the data.)

### 13.2.2 Dimensionality Reduction

Dimensionality reduction is to transfer the samples (instances) in the training data from a high-dimensional space to a low-dimensional space. Assuming that the samples originally exist in the low-dimensional space, or approximately the low-dimensional space, the dimensionality reduction can better represent the structure of the sample data, i.e., the relationship between samples. The high-dimensional space is usually a high-dimensional Euclidean space, while the low-dimensional space is a low-dimensional Euclidean space or manifold. The low-dimensional space is not given in advance but discovered automatically from the data, and its dimensionality is usually given in advance. In dimensionality reduction from high-dimensional to low-dimensional, it is necessary to ensure the minimal information loss in the sample. There are linear dimensionality reduction and nonlinear dimensionality reduction.



**Fig. 13.3** Example of dimensionality reduction

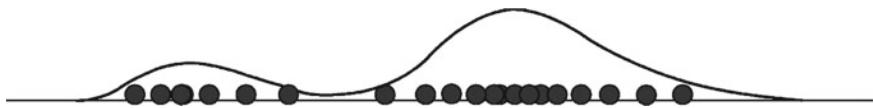
An example of dimensionality reduction is given in Fig. 13.3. A sample in the two-dimensional space exists in the vicinity of a straight line and can be transferred from the two-dimensional space to the one-dimensional space. The relationship between samples can be better represented by dimensionality reduction.

Suppose the input space is the Euclidean space  $X \subseteq R^d$  and the output space is also the Euclidean space  $Z \subseteq R^{d'}, d' \ll d$ , the dimensionality of the latter is lower than that of the former. The model for dimensionality reduction is the function  $z = g_\theta(x)$ , where  $x \in X$  is the high-dimensional vector of samples,  $z \in Z$  is the low-dimensional vector of samples, and  $\theta$  is the parameter. The function can be linear or nonlinear.

The process of dimensionality reduction is learning the dimensionality reduction model. In dimensionality reduction, each sample is transformed from a high-dimensional vector to a low-dimensional vector  $z_i = g_\theta(x_i)$ ,  $i = 1, 2, \dots, N$ . As shown in Fig. 13.1, dimensionality reduction can help discover the hidden horizontal structure in data.

### 13.2.3 Probability Model Estimation

Probability model estimation, or probability estimation for short, assumes that the training data is generated by a probability model and learns the structure and parameters of the probability model from the training data. The structure type of the probability model, or the set of probability models, is given in advance, while the specific structure and parameters of the model are learned automatically from the data. The learning goal is to find the structure and parameters most likely to generate data. Probability models include the mixture model, probabilistic graphical model, etc. Probabilistic graphical models include the directed graph model and the undirected graph model. Figure 13.4 shows an example of mixture model estimation. Assuming that the data is generated by the Gaussian Mixture Model (GMM), the learning goal is to estimate the parameters of the model.



**Fig. 13.4** Example of probability model estimation

The probability model is expressed as the conditional probability distribution  $P_\theta(x|z)$ , where the random variable  $x$  represents the observation data, which can be a continuous variable or a discrete variable; the random variable  $z$  represents the implicit structure and is a discrete variable; the random variable  $\theta$  represents the parameter. When the model is a mixture model,  $z$  represents the number of components; when the model is a probabilistic graphical model,  $z$  represents the graph structure.

A special case of the probability model is that the implicit structure does not exist, i.e., it satisfies  $P_\theta(x|z) = P_\theta(x)$ . At this point, the conditional probability distribution estimation becomes the probability distribution estimation, and it is sufficient to estimate the parameters of the distribution  $P_\theta(x)$ . It is the case for probability density estimation in traditional statistics, such as Gaussian distribution parameter estimation.

Probability model estimation is to learn the structure and parameters of the model  $P_\theta(x|z)$  from the given training data  $U = \{x_1, x_2, \dots, x_N\}$ , which allows the computation of arbitrary marginal distributions and conditional distributions associated with the model. Note that the random variable  $x$  is a multi-variable, even a high-dimensional multi-variable. As shown in Fig. 13.1, probability model estimation can help discover the horizontal and vertical structures hidden in the data.

Soft clustering can also be regarded as a probability model estimation problem. According to Bayes' Formula

$$P(z|x) = \frac{P(z)P(x|z)}{P(x)} \propto P(z)P(x|z) \quad (13.1)$$

Assuming that the prior probability obeys the stationary distribution, only the conditional probability distribution  $P_\theta(x|z)$  needs to be estimated. In this way, soft clustering can be performed by estimating the conditional probability distribution  $P_\theta(x|z)$ , where  $z$  represents the class and  $\theta$  represents the parameter.

### 13.3 Three Elements of Machine Learning

Like supervised learning, unsupervised learning has three elements: model, strategy and algorithm.

The model can be expressed by the function  $z = g_\theta(x)$ , the conditional probability distribution  $P_\theta(z|x)$ , or the conditional probability distribution  $P_\theta(x|z)$ . It has

different forms in clustering, dimensionality reduction and probability model estimation. For example, the output of the model in clustering is the class, while the one in dimensionality reduction is a low dimensional vector. The model in probability model estimation can be a mixture probability model, a directed probabilistic graphical model and an undirected probabilistic graphical model.

Strategies have different forms in different problems, but they all can be expressed as the optimization of the objective function, such as minimizing the distance between samples and their class centers in clustering, minimizing the information loss in sample transformation from high-dimensional space to low-dimensional space in dimensionality reduction, and maximizing the probability of model-generated data in probability model estimation.

The algorithm is usually an iterative algorithm, which achieves the optimization of the objective function through iteration, e.g., the Gradient Descent method.

Hierarchical clustering and  $k$ -means clustering are hard clustering methods, and the EM algorithm for Gaussian Mixture Model is a soft clustering method. Principal Component Analysis (PCA) and Latent Semantic Analysis (LSA) are dimensionality reduction methods. Probabilistic Latent Semantic Analysis (PLSA) and Latent Dirichlet Allocation (LDA) are methods of probability model estimation.

## 13.4 Unsupervised Learning Methods

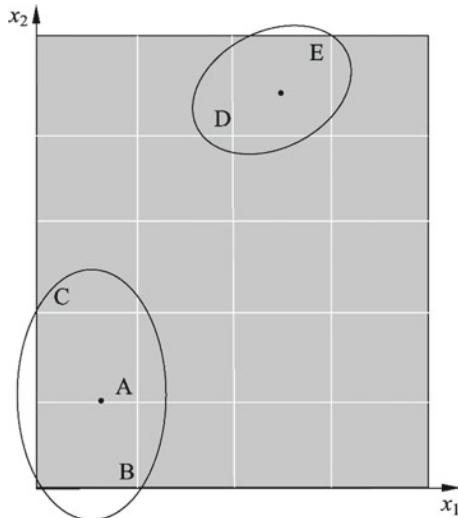
### 13.4.1 Clustering

Clustering is mainly used for data analysis and can also be used for preprocessing of supervised learning. Clustering can help discover statistical rules in data. The data is usually represented by continuous variables, but it can also be discrete. Chapter 14 will introduce clustering methods, including hierarchical clustering and  $k$ -means clustering.

Table 13.1 shows a simple dataset that contains five samples, A, B, C, D, and E, each with two-dimensional features  $x_1$  and  $x_2$ . Figure 13.5 shows the position of the samples in the two-dimensional real number space. Through the clustering algorithm, these samples can be allocated to two classes. Suppose  $k$ -means clustering is used,  $k = 2$ . First, take any two points as the centers of the two classes; and allocate samples to the two classes according to the Euclidean distance between samples and class centers. Then compute the mean value of samples in the two classes and use them as the new centers. Repeat the above operation until the two classes no longer change. Finally the clustering result is obtained, with A, B and C in one class and D and E in the other class.

**Table 13.1** Clustering data ( $x_1, x_2$ )

	A	B	C	D	E
$x_1$	1	1	0	2	3
$x_2$	1	0	2	4	5

**Fig. 13.5** Results of clustering

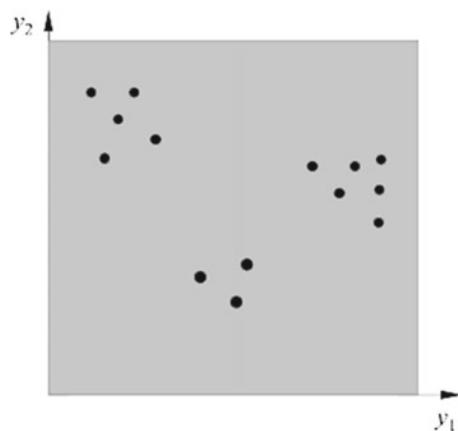
### 13.4.2 Dimensionality Reduction

Dimensionality reduction is mainly for data analysis. But it is also applicable in the preprocessing of supervised learning. Dimensionality reduction can help discover statistical rules in high-dimensional data. The data is represented as continuous variables. Chapter 16 will describe the Principal Component Analysis (PCA) in the dimensionality reduction method, and Chap. 15 will introduce the basic Singular Value Decomposition (SVD).

Table 13.2 shows a simple dataset. There are 14 samples, A, B, C, D, etc., each with 9-dimensional features  $x_1, x_2, \dots, x_9$ . Since the data is high-dimensional (multivariate), it is difficult to observe the sample distinguishing ability of variables and the relationship between samples. For example, when samples represent cells, and features denote indicators in cells, it is difficult to directly observe from the data which variables help distinguish between cells, i.e., which cells are similar and which are not. Performing a dimensionality reduction on the data, such as the Principal Component Analysis (PCA), allows for a more direct analysis of the above issues. Figure 13.6 shows the results of the dimensionality reduction (using PCA) on the sample set. As a result, there are two-dimensional new features  $y_1$  and  $y_2$  in the new two-dimensional real space with 14 samples distributed in different positions.

**Table 13.2** Clustering data ( $x_1$ – $x_9$ )

	A	B	C	D	...
$x_1$	3	0.25	2.8	0.1	...
$x_2$	2.9	0.8	2.2	1.8	...
$x_3$	2.2	1	1.5	3.2	...
$x_4$	2	1.4	2	0.3	...
$x_5$	1.3	1.6	1.6	0	...
$x_6$	1.5	2	2.1	3	...
$x_7$	1.1	2.2	1.2	2.8	...
$x_8$	1	2.7	0.9	0.3	...
$x_9$	0.4	3	0.6	0.1	...

**Fig. 13.6** Results of dimensionality reduction (principal component analysis)

By dimensionality reduction, it is evident that the samples can be divided into three classes, and the original features define the new two-dimensional features.

### 13.4.3 Topic Modeling

The topic modeling is a technique of text analysis. Given a text collection, the topic modeling aims to discover the topic of each text. A set of words is applied to denote the topic. Note that sufficient texts are assumed here. If only one or a few texts are available, it is impossible to conduct the topic modeling. The topic modeling can be formalized as a probability model estimation problem or a dimensionality reduction problem. Chapters 17, 18, and 20 will introduce Latent Semantic Analysis (LSA), Probabilistic Latent Semantic Analysis (PLSA), and Latent Dirichlet Allocation (LDA) in topic modeling. Chapter 19 will describe the basic Markov Chain

**Table 13.3** Data for topic modeling

Word	Text					
	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
Word 1	1	1				
Word 2	1		1			
Word 3		1	1			
Word 4				1	1	
Word 5				1		1
Word 6					1	1

**Table 13.4** Results of topic modeling (with LDA)

Word	Topic		Text	Topic	
	Topic 1	Topic 2		Topic 1	Topic 2
Word 1	0.33	0	Doc 1	1	0
Word 2	0.33	0	Doc 2	1	0
Word 3	0.33	0	Doc 3	1	0
Word 4	0	0.33	Doc 4	0	1
Word 5	0	0.33	Doc 5	0	1
Word 6	0	0.33	Doc 6	0	1

Monte Carlo (MCMC).

Table 13.3 gives a collection of text data with six texts and six words. The numbers in the table indicate the number of occurrences of the words in the text. The data is subjected to topic modeling, such as LDA, to obtain the topics represented by the set of words and the texts represented by the set of topics. As shown in Table 13.4, topics are specifically expressed as the probability distribution of words, and texts are represented as the probability distribution of topics. LDA is a model containing these probability distributions. Intuitively, a topic contains semantically similar words, and a text contains several topics.

#### 13.4.4 Graph Analytics

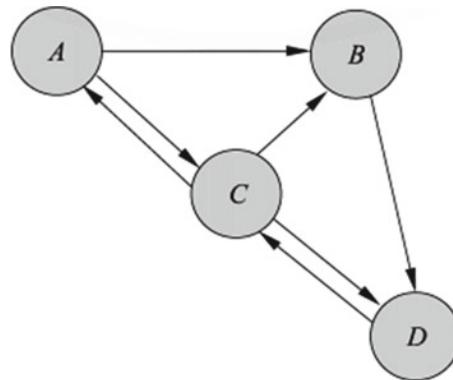
Data in many applications is in the form of graphs that denote the relationships between entities. Graphical data include directed graphs, undirected graphs, and hypergraphs. Graph analytics aims to uncover the statistical patterns or underlying structures hidden in the graphs. Link analysis is a type of graph analytics, including the PageRank algorithm, which focuses on discovering important nodes in directed graphs. Chapter 21 will introduce the PageRank algorithm.

The PageRank algorithm is an unsupervised learning method. With a given directed graph, a random walk, i.e., a Markov chain, is defined on the graph. The random walk hops randomly on the directed graph, reaches a node and hops to the linked-out node with equal probability, and continues this process. The PageRank algorithm is the algorithm that solves the stationary distribution of this Markov chain. The stationary probability on a node indicates the node's importance and is called the PageRank value of that node. The more nodes are pointed to, the higher the PageRank value of the node; the higher the PageRank value of the node being pointed to, the greater the PageRank value of the node. Intuitively, the larger the PageRank value, the more meaningful the node is.

Here is a brief introduction to the principle of PageRank. Figure 13.7 shows a simple directed graph with four nodes A, B, C, and D. With the given graph, the PageRank algorithm iterates through the nodes to find the PageRank values. First, the probability values are initialized for each node, indicating the arrival probability of each node, which is assumed to be equal. In the next step, the probability of each node is the sum of the probabilities that each node may jump to that node in the previous step. By iterating continuously, the distribution of the arrival probabilities of nodes tends to a stationary distribution, i.e., the distribution of PageRank values. The iterative process is shown in Table 13.5. It can be seen that the PageRank values of nodes C and D are larger.

The PageRank algorithm was originally proposed for Internet search. The Internet can be viewed as a giant directed graph, with web pages as nodes and hyperlinks to

**Fig. 13.7** Directed graph data



**Table 13.5** Result of PageRank computation

Node	Step		
	Step 1	Step 2	Step 3
A	1/4	2/24	3/24
B	1/4	5/24	4/24
C	1/4	9/24	9/24
D	1/4	8/24	8/24

web pages as directed edges. The PageRank algorithm can compute the PageRank values of web pages and indicates their importance, which plays a vital role in ranking search engines.

## Summary

1. Machine learning or statistical learning generally includes supervised learning, unsupervised learning, and reinforcement learning.

Unsupervised learning refers to the machine learning problem of learning models from unlabeled data. The unlabeled data is naturally obtained, and the model represents the class, transformation, or probability of the data. The essence of unsupervised learning is to learn statistical patterns or underlying structures in data, mainly including clustering, dimensionality reduction, and probability estimation.

2. Unsupervised learning can be used to analyze existing data and predict future data. The models obtained by learning are the function  $z = g(x)$ , the conditional probability distribution  $P(z|x)$ , or the conditional probability distribution  $P(x|z)$ .

The basic idea of unsupervised learning is to “compress” the given data (matrix data) to find the underlying structure of the data, assuming that the compression with minimal loss results in the most intrinsic structure. It is possible to consider discovering the vertical structure of the data corresponding to clustering, or the horizontal structure of the data, which corresponds to dimensionality reduction. It is also possible to consider discovering both the vertical and horizontal structures of the data, corresponding to probability model estimation.

3. Clustering is to assign similar samples (instances) in the sample set to the same class and dissimilar samples to different classes. Clustering includes hard clustering and soft clustering, and its methods are hierarchical clustering and  $k$ -means clustering.
4. Dimensionality reduction is the conversion of samples (instances) in a sample set from a high-dimensional space to a low-dimensional space. Assuming that the sample originally exists or approximately exists in the low-dimensional space, the dimensionality reduction can better represent the structure of the sample data, i.e., better represent the relationship between the samples. There is linear and nonlinear dimensionality reduction, whose method is Principal Component Analysis (PCA).
5. Probability model estimation assumes that the training data is generated by a probabilistic model, and the training data is used to learn the structure and parameters of the probabilistic model. Probabilistic models include mixture models, probabilistic graphical models, etc. Probabilistic graphical models include directed graph models and undirected graph models.
6. Topic modeling is a technique of text analysis. Given a collection of texts, topic modeling aims to discover the topic of each text in the text collection, which is represented by a collection of words. The methods of topic modeling are Latent

Semantic Analysis (LSA), Probabilistic Latent Semantic Analysis (PLSA), and Latent Dirichlet Allocation (LDA).

7. The purpose of graph analytics is to uncover the statistical patterns or underlying structures hidden in the graph. Link analysis is a type of graph analytics that focuses on discovering important nodes in directed graphs, including the PageRank algorithm.

## Further Reading

For more about unsupervised learning, please refer to references [1–7].

## References

1. Hastie T, Tibshirani R, Friedman J. *The elements of statistical learning: data mining, inference, and prediction*. Springer; 2001.
2. Bishop M. *Pattern Recognition and Machine Learning*. Springer; 2006.
3. Koller D, Friedman N. *Probabilistic graphical models: principles and techniques*. Cambridge, MA: MIT Press; 2009.
4. Goodfellow I, Bengio Y, Courville A. *Deep learning*. Cambridge, MA: MIT Press; 2016.
5. Michale T M. *Machine Learning*. McGraw-Hill Companies, Inc.; 1997.
6. Barber D. *Bayesian reasoning and machine learning*. Cambridge, UK: Cambridge University Press; 2012.
7. Zhou ZH. Zhou Zhi-Hua. *Machine Learning*: Tsinghua University Press; 2017.

# Chapter 14

## Clustering

Clustering is a data analysis problem in which given samples are grouped into several “classes” or “clusters” based on the similarity or distance of their features. A class is a subset of a given sample set. Intuitively, similar samples cluster in the same class, while dissimilar samples scatter in different classes. Here, the similarity or distance between samples plays an important role.

The purpose of clustering is to discover the characteristics of the data or process the data by the obtained classes or clusters. It has wide applications in data mining, pattern recognition and other fields. Clustering belongs to unsupervised learning since it only groups samples according to their similarity or distance without knowing the classes or clusters in advance.

There are many clustering algorithms. This chapter introduces the two most commonly used clustering algorithms: hierarchical clustering and  $k$ -means clustering. Hierarchical clustering has two methods: agglomerative (bottom-up) and divisive (top-down). The agglomerative method starts by assigning each sample into a class. Afterward, the two closest classes merge to create a new class, and the operation repeats until the stopping condition is met. Hierarchical classes are thus obtained. The divisive method starts by assigning all samples to one class. Then it divides the most distant samples in the existing classes into two new classes and repeats this operation until the stopping condition is satisfied. Hierarchical classes are thus obtained.  $k$ -means clustering is a center-based clustering method that iteratively divides samples into  $k$  classes such that each sample is closest to the center or mean of the class to which it belongs. Thus,  $k$  “flat” and non-hierarchical classes constituting the space division are obtained. The  $k$ -means clustering algorithm was proposed by MacQueen in 1967.

Section 14.1 of this chapter introduces the basic concepts of clustering, while Sects. 14.2 and 14.3 describe hierarchical clustering and  $k$ -means clustering, respectively.

## 14.1 Basic Concepts of Clustering

This section introduces the basic concepts of clustering, including the distance or similarity between samples, the distance between classes or clusters, and the distance between classes.

### 14.1.1 Similarity or Distance

The object of clustering is observation data, or sample collection. Suppose there are  $n$  samples, and each sample is composed of feature vectors of  $m$  attributes. The sample set can be represented by a matrix  $X$

$$X = [x_{ij}]_{m \times n} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad (14.1)$$

The  $j$ -th column of the matrix represents the  $j$ -th sample,  $j = 1, 2, \dots, n$ ; the  $i$ -th row represents the  $j$ -th attribute,  $i = 1, 2, \dots, m$ ; the matrix element  $x_{ij}$  represents the  $j$ -th attribute value of samples,  $i = 1, 2, \dots, m$ ;  $j = 1, 2, \dots, n$ .

The core concept of clustering is similarity or distance. There are many definitions of similarity or distance. Because similarity directly affects the result of clustering, its choice is the fundamental issue of clustering. Which degree of similarity is more appropriate depends on the characteristics of the application problem.

#### 14.1.1.1 Minkowski Distance<sup>1</sup>

In clustering, the sample set can be regarded as a set of points in a vector space, and the distance in the space represents the similarity between the samples. Commonly used distances are Minkowski distance, especially Euclidean distance. The greater the Minkowski distance, the smaller the similarity, and the smaller the distance, the greater the similarity.

**Definition 14.1** Given a sample set  $X$ ,  $X$  is the set of points in the  $m$ -dimensional real vector space  $\mathbf{R}^m$ , where  $x_i, x_j \in X$ ,  $x_i = (x_{1i} x_{2i} \cdots x_{mi})^T$ ,  $x_j = (x_{1j} x_{2j} \cdots x_{mj})^T$ , the Minkowski distance between sample  $x_i$  and sample  $x_j$  is defined as.

---

<sup>1</sup> The Minkowski distance was described in Chapter 3. Now it is restated with a change in notation.

$$d_{ij} = \left( \sum_{k=1}^m |x_{ki} - x_{kj}|^p \right)^{\frac{1}{p}} \quad (14.2)$$

Here  $p > 1$ . When  $p = 2$ , it is called Euclidean distance, that is

$$d_{ij} = \left( \sum_{k=1}^m |x_{ki} - x_{kj}|^2 \right)^{\frac{1}{2}} \quad (14.3)$$

When  $p = 1$ , it is called Manhattan distance, namely

$$d_{ij} = \sum_{k=1}^m |x_{ki} - x_{kj}| \quad (14.4)$$

When  $p = \infty$ , it is called Chebyshev distance, take the maximum value of the absolute value of each coordinate value difference, i.e.

$$d_{ij} = \max_k |x_{ki} - x_{kj}| \quad (14.5)$$

#### 14.1.1.2 Mahalanobis Distance

Mahalanobis distance is another commonly used similarity metric that considers the correlation between each component (feature) and is independent of the scale of each component. The greater the Mahalanobis distance, the smaller the similarity; the smaller the distance, the greater the similarity.

**Definition 14.2** Given a sample set  $X$ ,  $X = [x_{ij}]_{m \times n}$ , the covariance matrix is denoted as  $S$ . The Mahalanobis distance  $d_{ij}$  between sample  $x_i$  and sample  $x_j$  is defined as.

$$d_{ij} = \left[ (x_i - x_j)^T S^{-1} (x_i - x_j) \right]^{\frac{1}{2}} \quad (14.6)$$

where

$$x_i = (x_{1i}, x_{2i}, \dots, x_{mi})^T, x_j = (x_{1j}, x_{2j}, \dots, x_{mj})^T \quad (14.7)$$

When  $S$  is the identity matrix, that is, each component of the sample data is independent of each other and the variance of each component is 1, we know from Eq. (14.6) that Mahalanobis distance is Euclidean distance, so Mahalanobis distance is the generalization of Euclidean distance.

### 14.1.1.3 Correlation Coefficient

The similarity between samples can also be expressed by the correlation coefficient. The closer the absolute value of correlation coefficient is to 1, the more similar the samples are; the closer it is to 0, the less similar the samples are.

**Definition 14.3** The correlation coefficient between sample  $x_i$  and sample  $x_j$  is defined as.

$$r_{ij} = \frac{\sum_{k=1}^m (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)}{\left[ \sum_{k=1}^m (x_{ki} - \bar{x}_i)^2 \sum_{k=1}^m (x_{kj} - \bar{x}_j)^2 \right]^{\frac{1}{2}}} \quad (14.8)$$

where

$$\bar{x}_i = \frac{1}{m} \sum_{k=1}^m x_{ki}, \quad \bar{x}_j = \frac{1}{m} \sum_{k=1}^m x_{kj}$$

### 14.1.1.4 Cosine of the Angle

The similarity between samples can also be expressed by the cosine of the angle. The closer the cosine is to 1, the more similar the samples are; the closer it is to 0, the less similar the samples are.

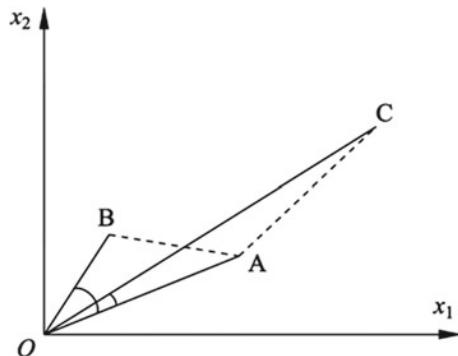
**Definition 14.4** The cosine of the angle between sample  $x_i$  and sample  $x_j$  is defined as.

$$s_{ij} = \frac{\sum_{k=1}^m x_{ki}x_{kj}}{\left[ \sum_{k=1}^m x_{ki}^2 \sum_{k=1}^m x_{kj}^2 \right]^{\frac{1}{2}}} \quad (14.9)$$

It can be seen from the above definition that when the distance is used to measure the similarity, the smaller the distance is, the more similar the samples are; when the correlation coefficient is used, the larger the correlation coefficient, the more similar the samples are. Note that the results of different similarity measures are not necessarily consistent. Please refer to Fig. 14.1.

It can be seen from the figure that  $A$  and  $B$  are more similar than  $A$  and  $C$  from the perspective of distance; but  $A$  and  $C$  are more similar than  $A$  and  $B$  in terms of the

**Fig. 14.1** The relationship between the distance and the correlation coefficient



correlation coefficient. Therefore, it is very important to select the proper distance or similarity when clustering.

### 14.1.2 Class or Cluster

The class or cluster obtained by clustering is essentially a subset of samples. If a clustering method assumes that a sample can only belong to one class, or the intersection of classes is empty, it is called hard clustering. Otherwise, if a sample can belong to multiple classes, or the intersection of classes is not empty, the method is called soft clustering. This chapter only considers the hard clustering method.

$G$  is used to represent the class or cluster,  $x_i$  and  $x_j$  are used to represent the samples in the class,  $n_G$  is used to represent the number of samples in  $G$ , and  $d_{ij}$  is used to represent the distance between  $x_i$  and  $x_j$ . There are many definitions of class or cluster. Here are some common definitions.

**Definition 14.5** Let  $T$  be a given positive number if any two samples  $x_i, x_j$  in the set  $G$  have.

$$d_{ij} \leq T$$

Then  $G$  is called a class or cluster.

**Definition 14.6** Let  $T$  be a given positive number. For any sample  $x_i$  of set  $G$ , there must be another sample  $x_j$  in  $G$  such that.

$$d_{ij} \leq T$$

Then  $G$  is called a class or cluster.

**Definition 14.7** Let  $T$  be a given positive number. If for any sample  $x_i$  in  $G$ , another sample  $x_j$  in  $G$  satisfies.

$$\frac{1}{n_G - 1} \sum_{x_j \in G} d_{ij} \leq T$$

where  $n_G$  is the number of samples in  $G$ , then  $G$  is called a class or cluster.

**Definition 14.8** Let  $T$  and  $V$  be two given positive numbers. If the distance  $d_{ij}$  of any two samples  $x_i, x_j$  in the set  $G$  satisfies.

$$\begin{aligned} \frac{1}{n_G(n_G - 1)} \sum_{x_i \in G} \sum_{x_j \in G} d_{ij} &\leq T \\ d_{ij} &\leq V \end{aligned}$$

Then  $G$  is called a class or cluster.

Of the above four definitions, the first one is the most commonly used. The other three can be derived from it.

The features of class can be described from different perspectives. There are three common features:

(1)  $\bar{x}_G$ , the mean value of a class is also called the center of a class

$$\bar{x}_G = \frac{1}{n_G} \sum_{i=1}^{n_G} x_i \quad (14.10)$$

where  $n_G$  is the number of samples of class  $G$ .

(2)  $D_G$ , the diameter of class

$D_G$ , the diameter of a class is the maximum distance between any two samples in the class, i.e.,

$$D_G = \max_{x_i, x_j \in G} d_{ij} \quad (14.11)$$

(3) The scatter matrix  $A_G$  and sample covariance matrix  $S_G$  of a class

The sample scatter matrix  $A_G$  of class is

$$A_G = \sum_{i=1}^{n_G} (x_i - \bar{x}_G)(x_i - \bar{x}_G)^T \quad (14.12)$$

The sample covariance matrix  $S_G$  is

$$\begin{aligned} S_G &= \frac{1}{n_G - 1} A_G \\ &= \frac{1}{n_G - 1} \sum_{i=1}^{n_G} (x_i - \bar{x}_G)(x_i - \bar{x}_G)^T \end{aligned} \quad (14.13)$$

### 14.1.3 Distance Between Classes

Next, consider the distance  $D(p, q)$  between class  $G_p$  and class  $G_q$ , also known as linkage. The distance between classes also has many definitions.

Let class  $G_p$  contain  $n_p$  samples and  $G_q$  contain  $n_q$  samples. Let  $\bar{x}_p$  and  $\bar{x}_q$  be used to represent the mean value of  $G_p$  and  $G_q$  respectively, that is, the center of the class.

#### (1) Shortest distance or single linkage

The shortest distance between the samples of  $G_p$  and  $G_q$  is defined as the distance between the two classes

$$D_{pq} = \min\{d_{ij} | x_i \in G_p, x_j \in G_q\} \quad (14.14)$$

#### (2) Longest distance or complete linkage

The longest distance between the samples of  $G_p$  and  $G_q$  is defined as the distance between the two classes

$$D_{pq} = \max\{d_{ij} | x_i \in G_p, x_j \in G_q\} \quad (14.15)$$

#### (3) Center distance

The distance between the centers  $\bar{x}_p$  and  $\bar{x}_q$  of class  $G_p$  and class  $G_q$  is defined as the distance between the two classes

$$D_{pq} = d_{\bar{x}_p \bar{x}_q} \quad (14.16)$$

#### (4) Average distance

The average distance between any two samples of class  $G_p$  and class  $G_q$  is defined as the distance between the two classes

$$D_{pq} = \frac{1}{n_p n_q} \sum_{x_i \in G_p} \sum_{x_j \in G_q} d_{ij} \quad (14.17)$$

## 14.2 Hierarchical Clustering

Hierarchical clustering assumes a hierarchical structure between classes and clusters samples into hierarchical classes. There are two types of hierarchical clustering: agglomerative or bottom-up clustering, and divisive or top-down clustering. Since each sample belongs to only one class, hierarchical clustering is hard clustering.

Agglomerative clustering starts by assigning each sample to a class; then the two closest classes are merged to create a new class; repeat this operation until the stopping condition is satisfied; thus the hierarchical classes are obtained. Divisive clustering starts by grouping all samples into one class; then groups the most distant samples in the existing classes into two new classes; repeat this operation until the stopping condition is met; the hierarchical classes are obtained. This book only introduces agglomerative clustering.

The specific process of agglomerative clustering is as follows: for a given set of samples, each sample is assigned to a class at first; then the two classes that best satisfy the rule condition are merged according to certain rules, such as the minimum distance between classes; repeat this process, reducing one class at a time until the stopping condition, such as all samples are clustered into one class, is satisfied.

It can be seen that agglomerative clustering needs to pre-determine the following three elements:

- (1) Distance or similarity;
- (2) Merging rules;
- (3) Stopping condition.

According to different combinations of these elements, different clustering methods can be formed. The distance or similarity can be Minkowski distance, Mahalanobis distance, correlation coefficient, cosine. The merging rule is generally that the distance between classes is the smallest, and the distance between classes can be the shortest distance, the longest distance, the center distance, and the average distance. The stopping condition can be that the number of classes reaches the threshold (the number of extreme cases is 1), and the diameter of the class exceeds the threshold.

If the Euclidean distance is used as the distance between samples; the smallest distance between classes is the merge rule, and the shortest distance is the distance between classes; the number of classes is 1, that is, all samples are clustered into one class, which is the stopping condition, then the algorithm for clustering is as follows.

### Algorithm 14.1 (Agglomerative Clustering Algorithm)

Input: the sample set composed of  $n$  samples and the distance between samples;

Output: a hierarchical clustering of sample sets.

- (1) Calculate the Euclidean distance  $\{d_{ij}\}$  between  $n$  samples, denoted as the matrix  $D = [d_{ij}]_{n \times n}$ .
- (2) Construct  $n$  classes, each class contains only one sample.
- (3) Combine the two classes with the smallest distance between classes, where the shortest distance is the distance between classes to construct a new class.

- (4) Calculate the distance between the new class and the current class. If the number of classes is 1, terminate the calculation, otherwise, go back to step (3).

It can be seen that the complexity of the agglomerative hierarchical clustering algorithm is  $O(n^3m)$ , where  $m$  is the dimension of the sample and  $n$  is the number of samples.

The following is an example to illustrate the agglomerative hierarchical clustering algorithm.

**Example 14.1** Given a set of five samples, the Euclidean distance between the samples is represented by the following matrix  $D$ :

$$D = [d_{ij}]_{5 \times 5} = \begin{bmatrix} 0 & 7 & 2 & 9 & 3 \\ 7 & 0 & 5 & 4 & 6 \\ 2 & 5 & 0 & 8 & 1 \\ 9 & 4 & 8 & 0 & 5 \\ 3 & 6 & 1 & 5 & 0 \end{bmatrix}$$

where  $d_{ij}$  represents the Euclidean distance between the  $i$ -th sample and the  $j$ -th sample. Obviously,  $D$  is a symmetric matrix. Clustering of these 5 samples is carried out using the agglomerative hierarchical clustering method.

### Solution

- (1) First, use five samples to construct five classes,  $G_i = \{x_i\}, 1, 2, \dots, 5$ . In this way, the distance between samples becomes the distance between classes. Therefore, the distance matrix between the five classes is also  $D$ .
- (2) It can be seen from the matrix  $D$  that  $D_{35} = D_{53} = 1$  is the smallest, so  $G_3$  and  $G_5$  are combined into a new class, denoted as  $G_6 = \{x_3, x_5\}$ .
- (3) Calculate the shortest distance between  $G_6$  and  $G_1, G_2, G_4$ , and we have

$$D_{61} = 2, D_{62} = 5, D_{64} = 5$$

Note that the distance between the other two categories is

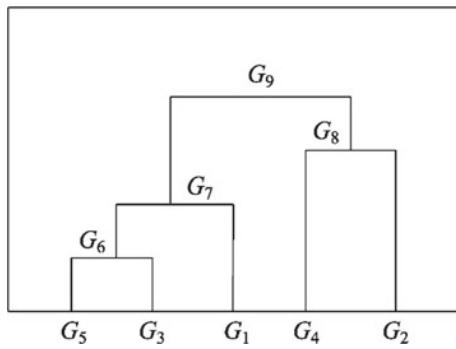
$$D_{12} = 7, D_{14} = 9, D_{24} = 4$$

Obviously,  $D_{61} = 2$  is the smallest. Then  $G_1$  and  $G_6$  are combined into a new class, denoted as  $G_7 = \{x_1, x_3, x_5\}$ .

- (4) Calculate the shortest distance between  $G_7$  and  $G_2, G_4$ ,

$$D_{72} = 5, D_{74} = 5$$

**Fig. 14.2** Hierarchical clustering diagram



Notice that

$$D_{24} = 4$$

Obviously,  $D_{24} = 4$  is the smallest, so  $G_2$  and  $G_4$  are combined into a new class, denoted as  $G_8 = \{x_2, x_4\}$ .

- (5) Combine  $G_7$  and  $G_8$  into a new class, denoted as  $G_9 = \{x_1, x_2, x_3, x_4, x_5\}$ , i.e., cluster all samples into one class. The clustering ends.

The above hierarchical clustering process can be represented by the above hierarchical clustering diagram (Fig. 14.2).

### 14.3 k-means Clustering

The  $k$ -means clustering is a clustering algorithm based on the division of sample sets. The  $k$ -means clustering divides the sample set into  $k$  subsets to form  $k$  classes, and divides  $n$  samples into  $k$  classes, and the distance between each sample and the center of the class to which it belongs is the smallest. Each sample can only belong to one class, so  $k$ -means clustering is hard clustering. The following introduces the model, strategy, and algorithm of  $k$ -means clustering, and discusses the characteristics of the algorithm and related issues.

#### 14.3.1 Model

Given a set of  $n$  samples  $X = \{x_1, x_2, \dots, x_n\}$ , each sample is represented by a feature vector, and the dimension of the feature vector is  $m$ . The goal of  $k$ -means clustering is to divide  $n$  samples into  $k$  different classes or clusters, assuming that  $k < n$ . The  $k$  classes  $G_1, G_2, \dots, G_k$  form the division of the sample set  $X$ , where  $G_i \cap$

$G_j = \emptyset$ ,  $\bigcup_{i=1}^k G_i = X$ . Using  $C$  to represent the division, a division corresponds to a clustering result.

### 14.3.2 Strategy

The  $k$ -means clustering boils down to the division of the sample set  $X$ , or the selection of the function from the sample to the class. The strategy of  $k$ -means clustering is to select the optimal partition or function  $C^*$  by minimizing the loss function.

First, the squared Euclidean distance is used as the distance between samples  $d(x_i, x_j)$

$$\begin{aligned} d(x_i, x_j) &= \sum_{k=1}^m (x_{ki} - x_{kj})^2 \\ &= \|x_i - x_j\|^2 \end{aligned} \quad (14.18)$$

Then, the sum of the distances between the sample and the center of its class is defined as the loss function, that is,

$$W(C) = \sum_{l=1}^k \sum_{C(i)=l} \|x_i - \bar{x}_l\|^2 \quad (14.19)$$

where  $\bar{x}_l = (\bar{x}_{1l}, \bar{x}_{2l}, \dots, \bar{x}_{ml})^T$  is the mean value or center of the  $l$ -th class, and  $n_l = \sum_{i=1}^n I(C(i) = l)$ ,  $I(C(i) = l)$  is indicating function with the value of 1 or 0. Function  $W(C)$ , also called energy, indicates the degree to simple similarity in the same class.

The  $k$ -means clustering is to solve the optimization problem:

$$\begin{aligned} C^* &= \arg \min_C W(C) \\ &= \arg \min_C \sum_{l=1}^k \sum_{C(i)=l} \|x_i - \bar{x}_l\|^2 \end{aligned} \quad (14.20)$$

When similar samples are clustered into the same class, the value of loss function is the smallest, and the optimization of this objective function can achieve the purpose of clustering. But this is a combinatorial optimization problem, with  $n$  samples divided into  $k$  classes, and the number of all possible partition methods is as follows:

$$S(n, k) = \frac{1}{k!} \sum_{l=1}^k (-1)^{k-l} \binom{k}{l} l^n \quad (14.21)$$

This number is exponential. In fact, the optimal solution of  $k$ -means clustering is NP-hard. In reality, iterative method is used to solve the problem.

### 14.3.3 Algorithm

The  $k$ -means clustering algorithm is an iterative process, and each iteration includes two steps. First, select the centers of  $k$  classes, assign the samples one by one into the classes with the nearest center, and get a clustering result; then, update the mean value of the samples of each class as the new center of this class; repeat the above steps until convergence. The specific process is as follows.

First, for given central values  $(m_1, m_2, \dots, m_k)$ , find a partition  $C$  that minimizes the objective function:

$$\min_C \sum_{l=1}^k \sum_{C(i)=l} ||x_i - m_l||^2 \quad (14.22)$$

That is to say, when the class center is determined, each sample is divided into a class to minimize the sum of distance between sample and its class center. Find the solution to make each sample assign to the class  $G_l$  of the nearest center  $m_l$ .

Then, for the given partition  $C$ , find the center  $(m_1, m_2, \dots, m_k)$  of each class again to minimize the objective function:

$$\min_{m_1, \dots, m_k} \sum_{l=1}^k \sum_{C(i)=l} ||x_i - m_l||^2$$

That is to say, when the partition is determined, minimize the distance between samples and the center of the class to which it belongs. Find the solution, and update its mean value  $m_l$  for each class  $G_l$  containing  $n_l$  samples:

$$m_l = \frac{1}{n_l} \sum_{C(i)=l} x_i, \quad l = 1, \dots, k$$

Repeat the above two steps until the division does not change, and the clustering result is obtained. Now the  $k$ -means clustering algorithm is described as follows.

#### Algorithm 14.2 ( $k$ -means clustering algorithm)

Input: the set  $X$  of  $n$  samples

Output: the clustering class  $C^*$  of sample set.

- (1) Initialization. Let  $t = 0$ , and select randomly  $k$  sample points as clustering the class centers  $m^{(0)} = (m_1^{(0)}, \dots, m_l^{(0)}, \dots, m_k^{(0)})$ .

- (2) Cluster the samples. For the fixed class centers  $m^{(t)} = (m_1^{(t)}, \dots, m_l^{(t)}, \dots, m_k^{(t)})$ , where  $m_l^{(t)}$  is the center of the class  $G_l$ , calculate the distance between each sample and class center, and assign each sample to the cluster with the nearest center to form the clustering result  $C^{(t)}$ .
- (3) Calculate the new class center. For clustering result  $C^{(t)}$ , calculate the mean of samples of the current each class as the new class center  $m^{(t+1)} = (m_1^{(t+1)}, \dots, m_l^{(t+1)}, \dots, m_k^{(t+1)})$ .
- (4) If the iteration converges or meets the stopping conditions, output  $C^* = C^{(t)}$ .

Otherwise, let  $t = t + 1$ , return to step (2).

The complexity of  $k$ -means clustering algorithm is  $O(mnk)$ , where  $m$  is the sample dimension,  $n$  is the number of samples,  $k$  is the number of categories.

**Example 14.2** Given a set of 5 samples

$$X = \begin{bmatrix} 0 & 0 & 1 & 5 & 5 \\ 2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Try  $k$ -means clustering algorithm to cluster the samples to 2 classes.

**Solution** According to the Algorithm 14.2.

- (1) Select two sample points as the class center. Suppose that  $m_1^{(0)} = x_1 = (0, 2)^T, m_2^{(0)} = x_2 = (0, 0)^T$  are selected.
- (2) Take  $m_1^{(0)}, m_2^{(0)}$  as the centers of the class  $G_1^{(0)}, G_2^{(0)}$  and calculate the Euclidean squared distances between  $x_3 = (1, 0)^T, x_4 = (5, 0)^T, x_5 = (5, 2)^T$  and  $m_1^{(0)} = (0, 2)^T, m_2^{(0)} = (0, 0)^T$ .

For  $x_3 = (1, 0)^T, d(x_3, m_1^{(0)}) = 5, d(x_3, m_2^{(0)}) = 1$ , sort  $x_3$  into class  $G_2^{(0)}$ .

For  $x_4 = (5, 0)^T, d(x_4, m_1^{(0)}) = 29, d(x_4, m_2^{(0)}) = 25$ , sort  $x_4$  into class  $G_2^{(0)}$ .

For  $x_5 = (5, 2)^T, d(x_5, m_1^{(0)}) = 25, d(x_5, m_2^{(0)}) = 29$ , sort  $x_5$  into class  $G_1^{(0)}$ .

- (3) Get the new class  $G_1^{(1)} = \{x_1, x_5\}, G_2^{(1)} = \{x_2, x_3, x_4\}$  and calculate the class center  $m_1^{(1)}, m_2^{(1)}$ :

$$m_1^{(1)} = (2.5, 2.0)^T, m_2^{(1)} = (2, 0)^T.$$

- (4) Repeat Step (2) and Step (3).

Assign  $x_1$  to class  $G_1^{(1)}$ ,  $x_2$  to class  $G_2^{(1)}$ ,  $x_3$  to class  $G_2^{(1)}$ ,  $x_4$  to class  $G_2^{(1)}$ , and  $x_5$  to class  $G_1^{(1)}$ .

Get the new class  $G_1^{(2)} = \{x_1, x_5\}, G_2^{(2)} = \{x_2, x_3, x_4\}$ .

Since the new classes obtained are not changed, the clustering stops. Get the clustering results:

$$G_1^* = \{x_1, x_5\}, G_2^* = \{x_2, x_3, x_4\}.$$

### 14.3.4 Algorithm Characteristics

#### 14.3.4.1 Overall Characteristics

$K$ -means clustering has following characteristics: partition-based clustering method; Number of classes  $k$  is specified in advance; the squared of the Euclidean distance represents the distance between the samples, and the mean of centers or samples represents class; The sum of the distance between the sample and the center of the class to which it belongs is the optimal objective function; The obtained class is flat and non-hierarchical; The algorithm is an iterative algorithm and cannot guarantee the global optimum.

#### 14.3.4.2 Convergence

The  $k$ -means clustering is a heuristic method, which cannot guarantee convergence to the global optimum, the selection of initial center will directly influence the clustering result. Note that the clustering center will move during the clustering process, but usually not very much, because the sample will be assigned to the class with the nearest center at each step.

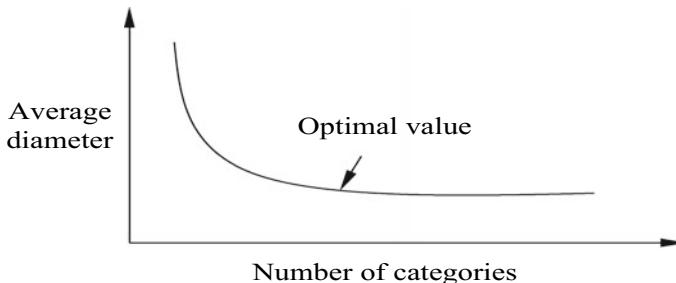
#### 14.3.4.3 Choice of Initial Class

Choosing different initial centers will result in different clustering results. For Example 14.2 above, if we change the initial centers of the two classes, such as choosing  $m_1^{(0)} = x_1$  and  $m_2^{(0)} = x_5$ , then  $x_2, x_3$  will be assigned to  $G_1^{(0)}$ , and  $x_4$  will be assigned to  $G_2^{(0)}$ , forming the clustering result  $G_1^{(1)} = \{x_1, x_2, x_3\}, G_2^{(1)} = \{x_4, x_5\}$ . Center is  $m_1^{(1)} = (0.33, 0.67)^T, m_2^{(1)} = (5, 1)^T$ . Continue to iterate, the clustering result is still  $G_1^{(2)} = \{x_1, x_2, x_3\}, G_2^{(2)} = \{x_4, x_5\}$ . The clustering stops.

The initial center can be chosen, for example, by clustering the samples using hierarchical clustering. It stops when  $k$  classes are obtained. Then a point closest to the center is selected from each class.

#### 14.3.4.4 Choice of the Number of Classes $k$

The number  $k$  of categories in  $k$ -means clustering needs to be specified in advance, and the optimal  $k$  value is unknown in practical applications. One way to solve this problem is to try clustering with different  $k$  values, test the quality of the different clustering results, and infer the optimal  $k$  value. The quality of the clustering results can be measured by the average diameter of the classes. In general, the average diameter increases when the number of classes gets smaller and remains constant when the number of classes gets larger than a certain value, which is the optimal  $k$ .



**Fig. 14.3** The relationship between the number of categories and the average diameter

value. Figure 14.3 illustrates the relationship between the number of categories and the average diameter. In the experiment, binary search can be used to find the optimal  $k$  value quickly.

### Summary

1. Clustering is a data analysis problem of grouping the given samples into several “classes” or “clusters” based on the similarity or distance of their attributes. A class is a subset of the sample. Intuitively, similar samples cluster in the same class, while dissimilar ones are grouped into different classes.
2. Distance or similarity metric plays an important role in clustering.

Minkowski distance is a commonly-used distance metrics, including Euclidean distance, Manhattan distance, Chebyshev distance, and Mahalanobis distance. Commonly-used similarity metrics are the correlation coefficient and cosine of the angle.

When using distance to measure similarity, the smaller the distance, the more similar the samples; when the correlation coefficient is used, the larger the correlation coefficient, the more similar the samples.

3. The class is a subset of the sample and has the following basic definition:

$G$  is used to denote the class or cluster,  $x_i, x_j$ , etc. are used to represent the samples in the class, and  $d_{ij}$  is used to represent the distance between sample  $x_i$  and sample  $x_j$ . If for any  $x_i, x_j \in G$ , there is

$$d_{ij} \leq T$$

then  $G$  is called class or cluster.

The indicators describing the features of the class include center, diameter, scatter matrix, and covariance matrix.

4. The distance between classes used in the clustering process is also called linkage. The distance between classes includes the shortest distance, the longest distance, the center distance, and the average distance.

5. Hierarchical clustering assumes a hierarchical structure between categories and clusters samples into hierarchical classes. There are two types of hierarchical clustering: agglomerative or bottom-up clustering, and divisive or top-down clustering.

Agglomerative clustering starts by assigning each sample to a class; then the two closest classes are merged to create a new class. Repeat this operation until the stopping condition is met. The hierarchical classes are obtained. Divisive clustering starts by grouping all samples into one class and then grouping the farthest samples in the existing classes into two new classes. Repeat the operation until the stopping condition is satisfied. Thus the hierarchical classes are obtained. Agglomerative clustering requires three pre-determined elements:

- (1) Distance or similarity;
- (2) Merging rules;
- (3) Stopping condition.

According to different combinations of these concepts, different clustering methods can be obtained.

6.  $k$ -means clustering is a commonly-used clustering algorithm with the following characteristics. It is a clustering method based on division; the number of categories  $k$  is specified in advance; the distance or similarity between the samples is expressed by the square of the Euclidean distance, and the center or the mean of the samples represents the class; the sum of the distance between samples and the center of the class to which it belongs is the optimized objective function; the obtained class is flat and non-hierarchical; the algorithm is iterative and does not guarantee the global optimum.

The  $k$ -means clustering algorithm first selects the centers of  $k$  classes and divides the samples into the classes closest to the center to obtain a clustering result; then it calculates the mean value of the samples of each class as the new centers of the classes. Repeat the above steps until convergence.

### Further Reading

There are many clustering methods, and details of each method can be seen in the literature [1, 2]. Hierarchical clustering methods can be seen in the literature [2], and the  $k$ -means clustering can be found in the literature [3, 4].  $k$ -means clustering is extended by X-means [5]. Other commonly-used clustering methods include methods based on mixture distribution, such as the Gaussian mixture model and the EM algorithm; methods based on density, like DBScan [6]; methods based on spectral clustering, such as Normalized Cuts [7]. The above methods are clustering of samples, and there are also methods of clustering samples and attributes simultaneously, such as Co-Clustering [8].

## Exercises

- 14.1 Try to write a division clustering algorithm to cluster the data top-down and give the complexity of the algorithm.
- 14.2 Prove that the first of the four definitions of a class or cluster leads to the other three definitions.
- 14.3 Prove that the Eq. (14.21) holds, i.e., the number of possible solutions for  $k$ -means is exponential.
- 14.4 Compare the similarities and differences between  $k$ -means clustering and the Gaussian mixture model plus the EM algorithm.

## References

1. Jain AK, Dubes RC. Algorithms for clustering data. Prentice-Hall; 1988.
2. Aggarwal CC, Reddy CK. Data clustering: algorithms and applications. CRC Press; 2013.
3. MacQueen J B. Some methods for classification and analysis of multivariate observations. Proceedings of 5th Berkley Symposium on Mathematical Statistics and Probability; 1967:281–97.
4. Hastie T, Tibshirani R, Friedman J H. The elements of statistical learning: data mining, inference, and prediction. Springer; 2001.
5. Pelleg D, Moore AW. X-means: extending K-means with efficient estimation of the number of clusters. Proceedings of ICML. 2000;1:727–34.
6. Ester M, Kriegel HP, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of ACM SIGKDD. 1996;96(43):226–31.
7. Shi J, Malik J. Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell. 2000;22(8):888–905.
8. Dhillon IS. Co-clustering documents and words using bipartite spectral graph partitioning. Proceedings of ACM SIGKDD; 2001:269–74.

# Chapter 15

## Singular Value Decomposition

### 15.1 Introduction

Singular Value Decomposition (SVD) is a matrix factorization method. It is a concept of linear algebra but is widely used as a vital tool in machine learning. For example, both the Principal Component Analysis (PCA) and Latent Semantic Analysis (LSA) introduced in this book involve SVD.

Any matrix  $m \times n$  can be expressed as the product (factorization) of three matrices, which are an  $m$ -order orthogonal matrix, an  $m \times n$  rectangular diagonal matrix consisting of non-negative diagonal elements arranged in descending order, and an  $n$ -order orthogonal matrix. It is called the Singular Value Decomposition (SVD) of the matrix. Singular value decomposition of matrix must exist, but it is not unique. The SVD can be regarded as a method of matrix data compression, i.e., an approximation of the original matrix by a factorization, which is the optimal approximation regarding square loss.

Section 15.1 describes the definition and fundamental theorem of matrix SVD, and introduces its compact and truncated form, geometric interpretation, and main properties; Sect. 15.2 deals with the algorithm of singular value decomposition; Sect. 15.3 discusses Singular Value Decomposition as an optimal approximation method of matrix.

### 15.2 Definition and Properties of Singular Value Decomposition

#### 15.2.1 Definition and Theorem

**Definition 15.1** (*Singular Value Decomposition*)

The Singular Value Decomposition (SVD) of a matrix is the factorization of a non-zero  $m \times n$  real matrix  $A$ ,  $A \in R^{m \times n}$  by representing it as the product of three real matrices<sup>1</sup>:

$$A = U \sum V^T \quad (15.1)$$

where  $U$  is an  $m$ -order orthogonal matrix,  $V$  is an  $n$ -order orthogonal matrix, and  $\sum$  is an  $m \times n$  rectangular diagonal matrix composed of non-negative diagonal elements arranged in descending order, which satisfy

$$\begin{aligned} UU^T &= I \\ VV^T &= I \\ \sum &= \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \\ \sigma_1 &\geq \sigma_2 \geq \dots \geq \sigma_p \geq 0 \\ p &= \min(m, n) \end{aligned}$$

$U \sum V^T$  is called the Singular Value Decomposition (SVD) of matrix  $A$ ,  $\sigma_i$  is called the singular value of matrix  $A$ , the column vector of  $U$  is called the left singular vector, and the column vector of  $V$  is called the right singular vector.

Note that Singular Value Decomposition (SVD) does not require matrix  $A$  to be a square matrix. The SVD of a matrix can be regarded as a generalization of the diagonalization of a square matrix.

See below an example of Singular Value Decomposition (SVD).

**Example 15.1** Give a  $5 \times 4$  matrix  $A$ .

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

Its SVD is given by the product  $U \sum V^T$  of three matrices,  $U$ ,  $\Sigma$ ,  $V^T$ .

$$U = \begin{bmatrix} 0 & 0 & \sqrt{0.2} & 0 & \sqrt{0.8} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \sqrt{0.8} & 0 & -\sqrt{0.2} \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

---

<sup>1</sup> The Singular Value Decomposition can be more generally defined on complex matrices, which is not covered here.

$$V^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The Matrix  $\Sigma$  is a diagonal matrix with all the elements outside the diagonal being 0 and the elements on the diagonal being non-negative and arranged in descending order. The Matrices  $U$  and  $V$  are orthogonal. They are unit matrices when multiplied by their respective transpose matrices, i.e.,

$$UU^T = I_5, VV^T = I_4$$

The SVD of the matrix is not unique. In this case, if  $U$  is selected as

$$U = \begin{bmatrix} 0 & 0 & \sqrt{0.2} & \sqrt{0.4} & -\sqrt{0.4} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{0.5} & \sqrt{0.5} \\ 0 & 0 & \sqrt{0.8} & -\sqrt{0.1} & \sqrt{0.1} \end{bmatrix}$$

while  $\Sigma$  and  $V$  are unchanged, then  $U\Sigma V^T$  is also a SVD of  $A$ .

For an arbitrary real matrix, does its SVD exist? The answer is yes, which is guaranteed by the following basic theorem of Singular Value Decomposition.

**Theorem 15.1** (basic theorem of Singular Value Decomposition) *If  $A$  is a  $m \times n$  real matrix,  $A \in R^{m \times n}$ , then there exists a SVD of  $A$ .*

$$A = U\Sigma V^T \quad (15.2)$$

where  $U$  is an  $m$ -order orthogonal matrix,  $V$  is an  $n$ -order orthogonal matrix, and  $\Sigma$  is an  $m \times n$  rectangular diagonal matrix, whose diagonal elements are nonnegative and arranged in descending order.

**Proof** The proof is constructive, i.e., for a given matrix  $A$ , the Singular Value Decomposition matrices are constructed. For convenience, suppose  $m \geq n$ . If  $m < n$ , the proof still holds. The proof consists of three steps.<sup>2</sup>

### 1. Determine $V$ and $\Sigma$ .

Firstly, the  $n$ -order orthogonal real matrix  $V$  and the  $m \times n$  rectangular diagonal real matrix  $\Sigma$  are constructed.

If matrix  $A$  is a  $m \times n$  real matrix, then the matrix  $A^T A$  is an  $n$ -order real symmetric matrix. Therefore, the eigenvalues of  $A^T A$  are all real numbers, and there is an  $n$ -order orthogonal real matrix  $V$  to realize the diagonalization of  $A^T A$ , which makes

---

<sup>2</sup> Basic knowledge of linear algebra can be found in the references in this chapter.

$V^T(A^T A)V = \Lambda$  hold. Here,  $\Lambda$  is an  $n$ -order diagonal matrix whose diagonal elements are composed of the eigenvalues of  $A^T A$ .

Moreover, the eigenvalues of  $A^T A$  are nonnegative. In fact, let  $\lambda$  be an eigenvalue of  $A^T A$  and  $x$  be the corresponding eigenvector, then

$$\|Ax\|^2 = x^T A^T Ax = \lambda x^T x = \lambda \|x\|^2$$

Thus

$$\lambda = \frac{\|Ax\|^2}{\|x\|^2} \geq 0 \quad (15.3)$$

It can be assumed that the arrangement of the columns of the orthogonal matrix  $V$  makes the corresponding eigenvalues arranged in descending order

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$$

Calculate the square root of the eigenvalue (actually the singular value of matrix  $A$ )

$$\sigma_j = \sqrt{\lambda_j}, \quad j = 1, 2, \dots, n$$

Suppose the rank of matrix  $A$  is  $r$ ,  $\text{rank}(A) = r$ , then the rank of the matrix  $A^T A$  is also  $r$ . Since  $A^T A$  is a symmetric matrix, its rank is equal to the number of positive eigenvalues, so

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0, \quad \lambda_{r+1} = \lambda_{r+2} = \cdots = \lambda_n = 0 \quad (15.4)$$

Correspondingly

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0, \quad \sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_n = 0 \quad (15.5)$$

Let

$$V_1 = [v_1, v_2, \dots, v_r], \quad V_2 = [v_{r+1}, v_{r+2}, \dots, v_n]$$

where  $v_1, \dots, v_r$  are the eigenvectors corresponding to the positive eigenvalues of  $A^T A$ , and  $v_{r+1}, \dots, v_n$  are the eigenvector corresponding to the eigenvalue 0, then

$$V = [V_1 \ V_2] \quad (15.6)$$

The matrix  $V$  is the  $n$ -order matrix in the Singular Value Decomposition of matrix  $A$ .

Let

$$\sum_1 = \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \\ & & & \sigma_r \end{bmatrix}$$

Then  $\sum_1$  is an  $r$ -order diagonal matrix whose diagonal elements are positive  $\sigma_1, \dots, \sigma_r$  in descending order, and the  $m \times n$  rectangular diagonal matrix  $\sum$  can be expressed as

$$\sum = \begin{bmatrix} \sum_1 & 0 \\ 0 & 0 \end{bmatrix} \quad (15.7)$$

This is the  $m \times n$  rectangular diagonal matrix  $\sum$  in the SVD of the matrix  $A$ .

A formula to be used later is derived below. In Eq. (15.6), the column vectors of  $V_2$  are the eigenvectors of  $A^T A$  corresponding to the eigenvalue 0. Therefore

$$A^T A v_j = 0, \quad j = r + 1, \dots, n \quad (15.8)$$

Then, the column vectors of  $V_2$  constitute the null space  $N(A^T A)$  of  $A^T A$ , and  $N(A^T A) = N(A)$ . Then the column vectors of  $V_2$  form a set of orthonormal bases for the null space of  $A$ . Thus,

$$A V_2 = 0 \quad (15.9)$$

Since  $V$  is an orthogonal matrix, Eq. (15.6) gives

$$I = V V^T = V_1 V_1^T + V_2 V_2^T \quad (15.10)$$

$$A = A I = A V_1 V_1^T + A V_2 V_2^T = A V_1 V_1^T \quad (15.11)$$

## 2. Determine $U$ .

Then construct the  $m$ -order orthogonal real matrix  $U$ .

Let

$$u_j = \frac{1}{\sigma_j} A v_j, \quad j = 1, 2, \dots, r \quad (15.12)$$

$$U_1 = [u_1, u_2, \dots, u_r] \quad (15.13)$$

Then there is

$$A V_1 = U_1 \sum_1 \quad (15.14)$$

The column vectors of  $U_1$  form a set of orthonormal vectors since

$$\begin{aligned}
 u_i^T u_j &= \left( \frac{1}{\sigma_j} v_i^T A^T \right) \left( \frac{1}{\sigma_j} A v_j \right) \\
 &= \frac{1}{\sigma_i \sigma_j} v_i^T (A^T A v_j) \\
 &= \frac{\sigma_j}{\sigma_i} v_i^T v_j \\
 &= \sigma_{ij}, i = 1, 2, \dots, r; j = 1, 2, \dots, r
 \end{aligned} \tag{15.15}$$

From Eqs. (15.12) and (15.15),  $u_1, u_2, \dots, u_r$  constitute a set of the orthonormal bases of the column space of  $A$  with dimension  $r$ . If  $A$  is regarded as a linear transformation from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ , the column space of  $A$  and the value range  $R(A)$  of  $A$  are the same. Therefore,  $u_1, u_2, \dots, u_r$  are also a set of orthonormal bases of  $R(A)$ .

If  $R(A)^\perp$  represents the orthogonal complement of  $R(A)$ , then the dimension of  $R(A)$  is  $r$ , the dimension of  $R(A)^\perp$  is  $m - r$ , and the sum of the two dimensions is equal to  $m$ . And  $R(A)^\perp = N(A^T)$  holds.<sup>3</sup>

Let  $\{u_{r+1}, u_{r+2}, \dots, u_m\}$  be a set of orthonormal bases of  $N(A^T)$ , and let

$$\begin{aligned}
 U_2 &= [u_{r+1}, u_{r+2}, \dots, u_m] \\
 U &= [U_1, U_2]
 \end{aligned} \tag{15.16}$$

Then  $u_1, u_2, \dots, u_m$  constitute a set of the orthonormal bases of  $\mathbb{R}^m$ . Therefore,  $U$  is an  $m$ -order orthogonal matrix, which is the  $m$ -order orthogonal matrix in the SVD of the matrix  $A$ .

3. Prove that  $U \sum V^T = A$ .

From Equations (15.6), (15.7), (15.11), (15.14), and (15.16), we have

$$\begin{aligned}
 U \sum V^T &= [U_1, U_2] \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \\
 &= U_1 \sum_1 V_1^T \\
 &= A V_1 V_1^T \\
 &= A
 \end{aligned} \tag{15.17}$$

It proves the existence of the Singular Value Decomposition of matrix  $A$ .

---

<sup>3</sup> See Appendix D.

### 15.2.2 Compact Singular Value Decomposition and Truncated Singular Value Decomposition

The Singular Value Decomposition is given by Theorem 15.1

$$A = U \Sigma V^T$$

It is also called the full Singular Value Decomposition. In practice, the compact and truncated form of Singular Value Decomposition are commonly used. The compact singular value decomposition is the SVD with an equal rank to the original matrix, and the truncated one is the SVD with a lower rank than the original matrix.

#### 15.2.2.1 Compact Singular Value Decomposition

**Definition 15.2** Suppose an  $m \times n$  real matrix  $A$  with  $\text{rank}(A) = r$ ,  $r \leq \min(m, n)$ . Then  $U_r \sum_r V_r^T$  is called the compact SVD of  $A$ , i.e.,

$$A = U_r \sum_r V_r^T \quad (15.18)$$

where  $U_r$  is an  $m \times r$  matrix,  $V_r$  is an  $n \times r$  matrix, and  $\Sigma_r$  is an  $r$ -order diagonal matrix. The matrix  $U_r$  is obtained from the first  $r$  columns of  $U$  in the full SVD. The matrices  $V_r$  and  $\Sigma_r$  are from the first  $r$  columns of  $V$ , and the first  $r$  diagonal elements of  $\Sigma$ . The rank of the diagonal matrix  $\Sigma_r$  of the compact SVD is equal to that of the original matrix  $A$ .

**Example 15.2** The rank of matrix  $A$  given by Example 15.1 is  $r = 3$ .

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

The compact SVD of  $A$  is

$$A = U_r \Sigma_r V_r^T$$

Where

$$U_r = \begin{bmatrix} 0 & 0 & \sqrt{0.2} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \sqrt{0.8} \end{bmatrix}, \Sigma_r = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & \sqrt{5} \end{bmatrix}, V_r^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

### 15.2.2.2 Truncated Singular Value Decomposition

In the Singular Value Decomposition of a matrix, its truncated SVD is obtained by taking only the part corresponding to the largest  $k$  singular values ( $k < r$ , and  $r$  is the rank of the matrix). When talking about the SVD of a matrix in practical applications, it usually refers to truncated Singular Value Decomposition.

**Definition 15.3** Let  $A$  be a  $m \times n$  real matrix with  $\text{rank}(A) = r$  and  $0 < k < r$ . Then  $U_k \Sigma_k V_k^T$  is called the truncated Singular Value Decomposition of matrix  $A$ .

$$A \approx U_k \Sigma_k V_k^T \quad (15.19)$$

Here  $U_k$  is an  $m \times k$  matrix,  $V_k$  is an  $n \times k$  matrix, and  $\Sigma_k$  is a  $k$ -order diagonal matrix. The matrix  $U_k$  is from the first  $k$  columns of  $U$  in the complete SVD, and the matrix  $V_k$  is from the first  $k$  columns of  $V$ . The matrix  $\Sigma_k$  is obtained from the first  $k$  diagonal elements of  $\Sigma$ . The rank of the diagonal matrix  $\Sigma_k$  is lower than that of the original matrix  $A$ .

**Example 15.3** The matrix  $A$  given in Example 15.1

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

The rank of  $A$  is 3. If  $k = 2$ , and its truncated SVD is

$$A \approx A_2 = U_2 \Sigma_2 V_2^T$$

where

$$U_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 4 & 0 \\ 0 & 3 \end{bmatrix}, V_2^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A_2 = U_2 \Sigma_2 V_2^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Here  $U_2$  and  $V_2$  are the first two columns of  $U$  and  $V$  in Example 15.1, and  $\Sigma_2$  is the first two rows and first two columns of  $\Sigma$ . Comparing  $A_2$  with  $A$ , we can see elements 1 and 2 of  $A$  become 0 in  $A_2$ .

In practical applications, it is often necessary to compress the data of a matrix and express it approximately. Singular Value Decomposition provides a way to do this. As described later, the SVD is the optimal approximation of the matrix regarding square loss (Frobenius norm). The compact SVD corresponds to lossless compression, and truncated SVD corresponds to lossy compression.

### 15.2.3 Geometry Interpretation

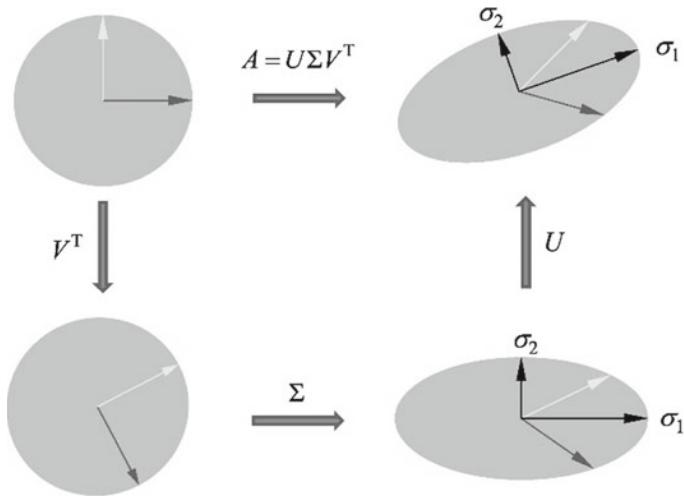
Understanding the Singular Value Decomposition from the perspective of the linear transformation, the  $m \times n$  matrix  $A$  represents a linear transformation from the  $n$ -dimensional space  $\mathbf{R}^n$  to the  $m$ -dimensional space  $\mathbf{R}^m$ ,

$$T : x \rightarrow Ax$$

with  $x \in \mathbf{R}^n$  and  $Ax \in \mathbf{R}^m$ .  $x$  and  $Ax$  are vectors in their respective spaces. The linear transformation can be decomposed into three simple transformations: rotation or reflection transformation of a coordinate system, a scaling transformation of one coordinate axis, and a rotation or reflection transformation of another coordinate system. The singular value theorem guarantees that this decomposition must exist. The above is the geometric interpretation of the Singular Value Decomposition.

Perform singular value decomposition on matrix  $A$  and get  $A = U \Sigma V^T$ .  $V$  and  $U$  are both orthogonal matrices, so the column vectors of  $V$ ,  $v_1, v_2, \dots, v_n$  constitute a set of orthonormal bases in  $\mathbf{R}^n$  space, representing the rotation or reflection transformation of the orthogonal coordinate system in  $\mathbf{R}^n$ ; the column vectors of  $U$ ,  $u_1, u_2, \dots, u_m$  constitute a set of orthonormal bases in  $\mathbf{R}^m$  space, representing the rotation or reflection transformation of the orthogonal coordinate system in  $\mathbf{R}^m$ . The diagonal elements of  $\Sigma$ ,  $\sigma_1, \sigma_2, \dots, \sigma_n$  is a set of non-negative real numbers, representing the scaling transformation of the coordinate axis of the original orthogonal coordinate system in  $\mathbf{R}^n$  by a factor of  $\sigma_1, \sigma_2, \dots, \sigma_n$ .

The linear transformation that any vector  $x \in \mathbf{R}^n$  undergoes based on  $A = U \Sigma V^T$  is equivalent to obtaining the vector  $Ax \in \mathbf{R}^m$  after the rotation or reflection transformation of the coordinate system ( $V^T$ ), the scaling transformation of the coordinate axis ( $\Sigma$ ), and the rotation or reflection transformation of the coordinate system ( $U$ ).



**Fig. 15.1** The geometric interpretation of singular value decomposition (see color diagram in Appendix)

Figure 15.1 gives an intuitive geometric explanation (see the color diagram before the text). The orthonormal bases of the original space (red and yellow) undergoes the rotation transformation of the coordinate system ( $V^T$ ), the scaling transformation of the coordinate axis ( $\Sigma$ ) (black  $\sigma_1, \sigma_2$ ), and the rotation transformation of the coordinate system ( $U$ ), and obtain the result equivalent to the linear transformation  $A$ .

The following is an example to illustrate the geometric meaning of Singular Value Decomposition intuitively.

**Example 15.4** Give a 2nd-order matrix

$$A = \begin{bmatrix} 3 & 1 \\ 2 & 1 \end{bmatrix}$$

whose SVD is

$$U = \begin{bmatrix} 0.8174 & -0.5760 \\ 0.5760 & 0.8174 \end{bmatrix}, \Sigma = \begin{bmatrix} 3.8643 & 0 \\ 0 & 0.2588 \end{bmatrix}, V^T = \begin{bmatrix} 0.9327 & 0.3606 \\ -0.3606 & 0.9327 \end{bmatrix}$$

Observe the linear transformation of the orthonormal bases of  $R^2$  based on the SVD of matrix  $A$ .

$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

First,  $V^T$  denotes a rotational transformation that rotates the orthonormal bases  $e_1$  and  $e_2$  to obtain the vectors  $V^T e_1$  and  $V^T e_2$ :

$$V^T e_1 = \begin{bmatrix} 0.9327 \\ -0.3606 \end{bmatrix}, \quad V^T e_2 = \begin{bmatrix} 0.3606 \\ 0.9327 \end{bmatrix}$$

Second,  $\Sigma$  denotes a scaling transformation that scales the vectors  $V^T e_1$  and  $V^T e_2$  by a factor of  $\sigma_1$  and  $\sigma_2$  in the direction of the coordinate axes to obtain the vectors  $\Sigma V^T e_1$  and  $\Sigma V^T e_2$ :

$$\Sigma V^T e_1 = \begin{bmatrix} 3.6042 \\ -0.0933 \end{bmatrix}, \quad \Sigma V^T e_2 = \begin{bmatrix} 1.3935 \\ 0.2414 \end{bmatrix}$$

Finally,  $U$  denotes a rotational transformation that rotates the vectors  $\Sigma V^T e_1$  and  $\Sigma V^T e_2$  to obtain the vectors  $U \Sigma V^T e_1$  and  $U \Sigma V^T e_2$ , i.e., the vectors  $Ae_1$  and  $Ae_2$ :

$$Ae_1 = U \Sigma V^T e_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \quad \Omega Ae_2 = \Sigma V^T e_2 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

In summary, the Singular Value Decomposition of a matrix can also be the decomposition of its corresponding linear transformation into a combination of rotation, scaling, and rotation transformations. According to Theorem 15.1, this combination of transformations must exist.

#### 15.2.4 Main Properties

- (1) Let the Singular Value Decomposition of the matrix  $A$  be  $A = U \Sigma V^T$ . Then the following relation holds:

$$A^T A = (U \Sigma V^T)^T (U \Sigma V^T) = V (\Sigma^T \Sigma) V^T \quad (15.20)$$

$$AA^T = (U \Sigma V^T)(U \Sigma V^T)^T = U (\Sigma \Sigma^T) U^T \quad (15.21)$$

That is to say, the eigendecomposition of the matrices  $A^T A$  and  $AA^T$  exists and can be represented by the matrix of the SVD of matrix  $A$ . The column vectors of  $V$  are the eigenvectors of  $A^T A$ , the column vectors of  $U$  are the eigenvectors of  $AA^T$ , and the singular values of  $\Sigma$  are the square roots of the eigenvalues of  $A^T A$  and  $AA^T$ .

- (2) In the Singular Value Decomposition of matrix  $A$ , there is a correspondence between the singular values, the left singular vector, and the right singular vector.

From  $A = U \Sigma V^T$ , we can see

$$AV = U\Sigma$$

Comparing the  $j$ -th column at both ends of this equation, we have

$$Av_j = \sigma_j u_j, \quad j = 1, 2, \dots, n \quad (15.22)$$

It is the relationship between the right singular vector and the singular values and the left singular vector of matrix  $A$ .

Similarly, from

$$A^T V = U\Sigma^T$$

we have

$$A^T v_j = \sigma_j u_j, \quad j = 1, 2, \dots, n \quad (15.23)$$

$$A^T v_j = 0, \quad j = n+1, n+2, \dots, m \quad (15.24)$$

It is the relationship between the left singular vector and the singular values and the right singular vector of matrix  $A$ .

- (3) The singular values  $\sigma_1, \sigma_2, \dots, \sigma_n$  are unique in the SVD of matrix  $A$ , while the matrices  $U$  and  $V$  are not unique.
- (4) The matrices  $A$  and  $\Sigma$  are of equal rank, which is equal to  $r$ , the number of positive singular values  $\sigma_i$  (containing repeated singular values).
- (5) The  $r$  right singular vectors  $v_1, v_2, \dots, v_r$  of matrix  $A$  constitute a set of orthonormal bases of  $R(A^T)$ , the value domain of  $A^T$ . Since the matrix  $A^T$  is a linear transformation mapped from  $R^m$  to  $R^n$ ,  $R(A^T)$  is the same as the column space of  $A^T$ .  $v_1, v_2, \dots, v_r$  is a set of orthonormal bases of  $A^T$  and thus a set of orthonormal bases of  $R(A^T)$ .<sup>4</sup>

The  $n-r$  right singular vectors  $v_{r+1}, v_{r+2}, \dots, v_n$  of a matrix  $A$  constitute a set of orthonormal bases of the null space  $N(A^T)$  of  $A^T$ .

The  $r$  left singular vectors  $u_1, u_2, \dots, u_r$  of matrix  $A$  form a set of orthonormal bases of the value domain  $R(A)$ .

The  $m-r$  left singular vectors  $u_{r+1}, u_{r+2}, \dots, u_m$  of matrix  $A$  form a set of orthonormal bases of the null space  $N(A^T)$  of  $A^T$ .

---

<sup>4</sup> See Appendix D.

## 15.3 Computation of Singular Value Decomposition

The proof of the basic theorem of Singular Value Decomposition implies the computation of Singular Value Decomposition. The SVD of matrix  $A$  can be obtained by finding the eigenvalues and eigenvectors of the symmetric matrix  $A^T A$ , whose eigenvectors form the columns of the orthogonal matrix  $V$ . The square root of the eigenvalue  $\lambda_j$  of  $A^T A$  is the singular value  $\sigma_j$ , i.e.

$$\sigma_j = \sqrt{\lambda_j}, \quad j = 1, 2, \dots, n$$

Arrange the values of  $\sigma_j$  in descending order and use them as diagonal elements to form the diagonal matrix  $\Sigma$ . The left singular vector corresponding to the positive singular value is found, and then the orthonormal bases of the expanded  $A^T$  are found, forming the columns of the orthogonal matrix  $U$ . The SVD of  $A = U \Sigma V^T$  is thus obtained.

Given an  $m \times n$  matrix  $A$ , the computational procedure of the matrix SVD can be written as described above.

- (1). First, find the eigenvalues and eigenvectors of  $A^T A$ .

Compute the symmetric matrix  $W = A^T A$ .

Solve the characteristic equation

$$(W - \lambda I)x = 0$$

Obtain the eigenvalues  $\lambda_j$  and rank the eigenvalues from largest to smallest

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$$

The eigenvalues  $\lambda_j (j = 1, 2, \dots, n)$  are substituted into the characteristic equation to obtain the corresponding eigenvectors.

- (2) Find the  $n$ -order orthogonal matrix  $V$ .

The eigenvectors are unitized to obtain the unit eigenvectors  $v_1, v_2, \dots, v_n$ , which form the  $n$ -order orthogonal matrix  $V$ :

$$V = [v_1, v_2, \dots, v_n]$$

- (3) Find the  $m \times n$  diagonal matrix  $\Sigma$ .

Calculate the singular value of  $A$

$$\sigma_j = \sqrt{\lambda_j}, \quad j = 1, 2, \dots, n$$

Construct  $m \times n$  rectangular diagonal matrix  $\Sigma$  with the main diagonal elements being singular and the remaining elements being zero,

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$$

(4). Find the  $m$ -order orthogonal matrix  $U$ .

For the first  $r$  positive singular values of  $A$ , let

$$u_j = \frac{1}{\sigma_j} Av_j, \quad j = 1, 2, \dots, r$$

and obtain

$$U_1 = [u_1, u_2, \dots, u_r]$$

Find a set of orthonormal bases  $\{u_{r+1}, u_{r+2}, \dots, u_m\}$  in the null space, let

$$U_2 = [u_{r+1}, u_{r+2}, \dots, u_m]$$

and

$$U = [U_1, U_2]$$

(5) Obtain the Singular Value Decomposition

$$A = U \Sigma V^T$$

The following is a simple example to illustrate the computation of Singular Value Decomposition.

**Example 15.5** Try to find the Singular Value Decomposition of the matrix.

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$$

### Solution

(1) Find the eigenvalues and eigenvectors of matrix  $A^T A$ .

Find the symmetric matrix  $A^T A$

$$A^T A = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$$

The eigenvalues  $\lambda$  and the eigenvectors  $x$  satisfy the characteristic equation

$$(A^T A - \lambda I)x = 0$$

Obtain the homogeneous linear equation system

$$\begin{cases} (5 - \lambda)x_1 + 5x_2 = 0 \\ 5x_1 + (5 - \lambda)x_2 = 0 \end{cases}$$

The necessary and sufficient condition for this system to have a non-zero solution is

$$\begin{vmatrix} 5 - \lambda & 5 \\ 5 & 5 - \lambda \end{vmatrix} = 0$$

i.e.,

$$\lambda^2 - 10\lambda = 0$$

Solving this equation yields the eigenvalues  $\lambda_1 = 10$  and  $\lambda_2 = 0$  of matrix  $A^T A$ .

Substitute the eigenvalue  $\lambda_1 = 10$  into the linear equation system to obtain the corresponding unit eigenvector

$$v_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Similarly, the unit eigenvector corresponding to eigenvalues  $\lambda_2 = 0$  is obtained.

$$v_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

(2) Find the orthogonal matrix  $V$ .

Construct the orthogonal matrix  $V$

$$V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

(3) Find the diagonal matrix  $\Sigma$ .

The singular values are  $\sigma_1 = \sqrt{\lambda_1} = \sqrt{10}$  and  $\sigma_2 = 0$ . Construct the diagonal matrix

$$\Sigma = \begin{bmatrix} \sqrt{10} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Note that zero row vectors should be added in  $\Sigma$  so that  $\Sigma$  can perform matrix multiplication with  $U$  and  $V$ .

(4) Find the orthogonal matrix  $U$ .

The column vector  $u_1$  is obtained based on the positive singular values of  $A$

$$u_1 = \frac{1}{\sigma_1} A v_1 = \frac{1}{\sqrt{10}} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \\ 0 \end{bmatrix}$$

The column vectors  $u_2$  and  $u_3$  are a set of orthonormal bases of the null space  $N(A^T)$  of  $A^T$ . To obtain  $u_2$  and  $u_3$ , solve the following linear equation system

$$A^T x = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

i.e.,

$$\begin{aligned} x_1 + 2x_2 + 0x_3 &= 0 \\ x_1 &= -2x_2 + 0x_3 \end{aligned}$$

Take  $(x_2, x_3)$  as  $(1, 0)$  and  $(0, 1)$ , respectively, the basis of  $N(A^T)$  is obtained as

$$(-2, 1, 0)^T, (0, 0, 1)^T$$

A set of orthonormal bases of  $N(A^T)$  is

$$u_2 = \left( -\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0 \right)^T, u_3 = (0, 0, 1)^T$$

Construct the orthogonal matrix  $UU$

$$U = \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} & 0 \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(5) The Singular Value Decomposition of matrix  $A$

$$A = U \Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} & 0 \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{10} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

The above algorithm and examples are to illustrate the process of computation. It is not the algorithm in practical application. It can be seen that the key to the Singular Value Decomposition algorithm is the computation of the eigenvalues of  $A^T A$ . The SVD algorithm for practical applications is performed by finding the eigenvalues of  $A^T A$  but not directly computing  $A^T A$ . The reader is referred to the references [1, 2] for more efficient algorithms for matrix Singular Value Decomposition, which are not presented here.

## 15.4 Singular Value Decomposition and Matrix Approximation

### 15.4.1 Frobenius Norm

Singular Value Decomposition is also a method of matrix approximation, which is an approximation in the sense of the Frobenius norm. The Frobenius norm of a matrix is a direct generalization of the  $L_2$  norm of a vector, which corresponds to the quadratic loss function in machine learning.

**Definition 15.4 (Frobenius norm)** Let the matrix  $A \in \mathbf{R}^{m \times n}$ ,  $A = [a_{ij}]_{m \times n}$ , and define the Frobenius norm of the matrix  $A$  as

$$\|A\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n (a_{ij})^2 \right)^{\frac{1}{2}} \quad (15.25)$$

**Lemma 15.1** Let the matrix  $A \in \mathbf{R}^{m \times n}$  and the Singular Value Decomposition of  $A$  be  $U \Sigma V^T$ , where  $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$ , then

$$\|A\|_F = (\sigma_1^2 + \sigma_2^2 + \dots + \sigma_n^2)^{\frac{1}{2}} \quad (15.26)$$

**Proof** In general, if  $Q$  is an  $m$ -order orthogonal matrix, then we have

$$\|QA\|_F = \|A\|_F \quad (15.27)$$

Since

$$\begin{aligned}\|QA\|_F^2 &= \|(Qa_1, Qa_2, \dots, Qa_n)\|_F^2 \\ &= \sum_{i=1}^n \|Qa_i\|_2^2 = \sum_{i=1}^n \|a_i\|_2^2 = \|A\|_F^2\end{aligned}$$

Similarly, if  $P$  is an  $n$ -order orthogonal matrix, then we have

$$\|AP^T\|_F = \|A\|_F \quad (15.28)$$

so

$$\|A\|_F = \|U\Sigma V^T\|_F = \|\Sigma\|_F \quad (15.29)$$

i.e.,

$$\|A\|_F = (\sigma_1^2 + \sigma_2^2 + \dots + \sigma_n^2)^{\frac{1}{2}} \quad (15.30)$$

### 15.4.2 Optimal Approximation of the Matrix

The Singular Value Decomposition is an optimal approximation of the matrix in the sense of square loss (Frobenius norm), i.e., data compression.

**Theorem 15.2** *Let a matrix  $A \in \mathbf{R}^{m \times n}$  with  $\text{rank}(A) = r$  and let  $\mathcal{M}$  be the set of all matrices whose rank is not larger than  $k$  in  $\mathbf{R}^{m \times n}$ ,  $0 < k < r$ . Then there exists a matrix  $X \in \mathcal{M}$  whose rank is  $k$  to make*

$$\|A - X\|_F = \min_{S \in \mathcal{M}} \|A - S\|_F \quad (15.31)$$

*The matrix  $X$  is said to be the optimal approximation of matrix  $A$  in the sense of the Frobenius norm.*

Instead of proving this theorem, this book will apply this result to find the approximate matrix  $X$  by the Singular Value Decomposition of matrix  $A$ .

**Theorem 15.3** *Let a matrix  $A \in \mathbf{R}^{m \times n}$  with  $\text{rank}(A) = r$  and Singular Value Decomposition  $A = U\Sigma V^T$  and let  $\mathcal{M}$  be the set of all matrices whose rank is not larger than  $k$  in  $\mathbf{R}^{m \times n}$  with  $0 < k < r$ . If the matrix  $X \in \mathcal{M}$  whose rank is  $k$  satisfies*

$$\|A - X\|_F = \min_{S \in \mathcal{M}} \|A - S\|_F \quad (15.32)$$

Then

$$\|A - X\|_F = (\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_n^2)^{\frac{1}{2}} \quad (15.33)$$

In particular, if  $A' = U\Sigma'V^T$ , where

$$\Sigma' = \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_k & \\ & & 0 & \\ 0 & & & \ddots \\ & & & 0 \end{bmatrix} = \begin{bmatrix} \Sigma_k & 0 \\ 0 & 0 \end{bmatrix}$$

Then

$$\|A - A'\|_F = (\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_n^2)^{\frac{1}{2}} = \min_{S \in \mathcal{M}} \|A - S\|_F \quad (15.34)$$

**Proof** Let  $X \in \mathcal{M}$  be a matrix satisfying Eq. (15.32). Since

$$\|A - X\|_F \leq \|A - A'\|_F = (\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_n^2)^{\frac{1}{2}} \quad (15.35)$$

prove that

$$\|A - X\|_F \geq (\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_n^2)^{\frac{1}{2}}$$

thus Eq. (15.33) holds.

Let the Singular Value Decomposition of  $X$  be  $Q\Omega P^T$ , where

$$\Omega = \begin{bmatrix} \omega_1 & & & \\ & \ddots & & 0 \\ & & \omega_k & \\ & & 0 & \\ 0 & & & \ddots \\ & & & 0 \end{bmatrix} = \begin{bmatrix} \Omega_k & 0 \\ 0 & 0 \end{bmatrix}$$

If we have the matrix  $B = Q^TAP$ , then  $A = QBP^T$ , resulting in

$$\|A - X\|_F = \|Q(B - \Omega)P^T\|_F = \|B - \Omega\|_F \quad (15.36)$$

Block  $B$  by the  $\Omega$  blocking method

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

where  $B_{11}$  is a  $k \times k$  submatrix,  $B_{12}$  is a  $k \times (n - k)$  submatrix,  $B_{21}$  is an  $(m - k) \times k$  submatrix, and  $B_{22}$  is an  $(m - k) \times (n - k)$  submatrix. This yields

$$\begin{aligned} \|A - X\|_F^2 &= \|B - \Omega\|_F^2 \\ &= \|B_{11} - \Omega_k\|_F^2 + \|B_{12}\|_F^2 + \|B_{21}\|_F^2 + \|B_{22}\|_F^2 \end{aligned} \quad (15.37)$$

Use proofs by contradiction to prove  $B_{12} = 0$  and  $B_{21} = 0$ . If  $B_{12} \neq 0$ , let

$$Y = Q \begin{bmatrix} B_{11} & B_{12} \\ 0 & 0 \end{bmatrix} P^T$$

Then  $Y \in M$ , and

$$\|A - Y\|_F^2 = \|B_{21}\|_F^2 + \|B_{22}\|_F^2 < \|A - X\|_F^2 \quad (15.38)$$

This contradicts the definition Eq. (15.35) of  $X$ , proving that  $B_{12} = 0$ . Similarly, we can prove that  $B_{21} = 0$ . Then

$$\|A - X\|_F^2 = \|B_{11} - \Omega_k\|_F^2 + \|B_{22}\|_F^2 \quad (15.39)$$

Next, try to prove that  $B_{11} = \Omega_k$ . Let

$$Z = Q \begin{bmatrix} B_{11} & 0 \\ 0 & 0 \end{bmatrix} P^T$$

Then  $Z \in M$ , and

$$\|A - Z\|_F^2 = \|B_{22}\|_F^2 \leq \|B_{11} - \Omega_k\|_F^2 + \|B_{22}\|_F^2 = \|A - X\|_F^2 \quad (15.40)$$

From Eq. (15.35), we have  $\|B_{11} - \Omega_k\|_F^2 = 0$ , i.e.,  $B_{11} = \Omega_k$ .

Finally, look at  $B_{22}$ . If the  $(m - k) \times (n - k)$  sub-matrix  $B_{22}$  has the Singular Value Decomposition  $U_1 \Lambda V_1^T$ , then

$$\|A - X\|_F = \|B_{22}\|_F = \|\Lambda\|_F \quad (15.41)$$

Prove that the diagonal elements of  $\Lambda$  are singular values of  $A$ . To do this, let

$$U_2 = \begin{bmatrix} I_k 0 \\ 0 U_2 \end{bmatrix}, V_2 = \begin{bmatrix} I_k 0 \\ 0 V_1 \end{bmatrix}$$

where  $I_k$  is a  $k$ -order unit matrix, and the blocking of  $U_2$  and  $V_2$  coincides with that of  $B$ . Noting the SVD of  $B$  and  $B_{22}$ , we obtain

$$U_2^T Q^T A P V_2 = \begin{bmatrix} \Omega_k & 0 \\ 0 & \Lambda \end{bmatrix} \quad (15.42)$$

$$A = (Q U_2) \begin{bmatrix} \Omega_k & 0 \\ 0 & \Lambda \end{bmatrix} (P V_2)^T \quad (15.43)$$

It can be seen that the diagonal elements of  $\Lambda$  are the singular values of  $A$ . So, there is

$$\|A - X\|_F = \|\Lambda\|_F \geq (\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_n^2)^{\frac{1}{2}} \quad (15.44)$$

It is proved that

$$\|A - X\|_F = (\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_n^2)^{\frac{1}{2}} = \|A - A'\|_F$$

Theorem 15.3 shows that in the set of  $m \times n$  matrices whose rank does not exceed  $k$ , there is an optimal approximate matrix  $X$  in the sense of the Frobenius norm of matrix  $A$ .  $A' = U \Sigma' V^T$  is a matrix that reaches the optimal value.

The compact Singular Value Decomposition and truncated SVD of a matrix are defined above. The compact SVD is lossless compression in the sense of Frobenius norm, and truncated SVD is lossy compression. The rank of the matrix obtained by truncated SVD is  $k$ , which is usually much smaller than the rank  $r$  of the original matrix, so the original matrix is compressed by the low-rank matrix.

### 15.4.3 The Outer Product Expansion of Matrix

The following describes the approximation of matrix  $A$  using the outer product expansion. The Singular Value Decomposition  $U \Sigma V^T$  of matrix  $A$  can also be expressed in the form of outer product. In fact, if the SVD of  $A$  is regarded as the product of matrices  $U \Sigma$  and  $V^T$ , the blocking of  $U \Sigma$  by column vector and the blocking of  $V^T$  by row vector yields

$$U \Sigma = [\sigma_1 u_1, \sigma_2 u_2, \cdots, \sigma_n u_n]$$

$$V^T = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ \vdots \\ v_n^T \end{bmatrix}$$

Then

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_n u_n v_n^T \quad (15.45)$$

Equation (15.45) is called the outer product expansion of matrix  $A$ , where  $u_k v_k^T$  is an  $m \times n$  matrix that is the outer product of the column vector  $u_k$  and the row vector  $v_k^T$ . The element in the  $i$ -th row and the  $j$ -th column is the product of the  $i$ -th element of  $u_k$  and the  $j$ -th element of  $v_k^T$ , i.e.,

$$u_i v_j^T = \begin{bmatrix} u_{1i} \\ u_{2i} \\ \vdots \\ u_{mi} \end{bmatrix} \begin{bmatrix} v_{1j} & v_{2j} & \cdots & v_{nj} \end{bmatrix} = \begin{bmatrix} u_{1i} v_{1j} & u_{1i} v_{2j} & \cdots & u_{1i} v_{nj} \\ u_{2i} v_{1j} & u_{2i} v_{2j} & \cdots & u_{2i} v_{nj} \\ \vdots & \vdots & \ddots & \vdots \\ u_{mi} v_{1j} & u_{mi} v_{2j} & \cdots & u_{mi} v_{nj} \end{bmatrix}$$

The outer product expansion of  $A$  can also be written in the following form

$$A = \sum_{k=1}^n \sigma_k u_k v_k^T \quad (15.46)$$

where  $\sigma_k u_k v_k^T$  is an  $m \times n$  matrix. Equation (15.46) decomposes matrix  $A$  into an ordered weighted sum of matrices.

From the outer product expansion of matrix  $A$ , if the rank of  $A$  is  $n$ , then

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_n u_n v_n^T \quad (15.47)$$

Let the matrix

$$A_{n-1} = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_{n-1} u_{n-1} v_{n-1}^T$$

Then the rank of  $A_{n-1}$  is  $n - 1$ , and matrix  $A_{n-1}$  is the optimal approximate matrix of  $A$  in the sense of Frobenius norm.

Similarly, let the matrix

$$A_{n-2} = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_{n-2} u_{n-2} v_{n-2}^T$$

The rank of  $A_{n-2}$  is  $n - 2$ , and matrix  $A_{n-2}$  is the optimal approximate matrix of  $A$  in the sense of Frobenius norm. And so forth. Generally, let the matrix

$$A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

Then the rank of  $A_k$  is  $k$ , and matrix  $A_k$  is the optimal approximate matrix of  $A$  in the sense of Frobenius norm. The matrix  $A_k$  is the truncated Singular Value Decomposition of  $A$ .

Since the singular value  $\sigma_i$  usually decreases very fast,  $A_k$  can also be a good approximation to  $A$  when  $k$  takes a small value.

**Example 15.6** The rank of matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

given by Example 15.1 is 3. Find the optimal approximation of  $A$  with rank being 2.

### Solution

According to Example 15.3,

$$u_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, u_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, v_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\sigma_1 = 4, \sigma_2 = 3$$

Then we obtain

$$A_2 = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This matrix is taken as the optimal approximation of  $A$ .

## Summary

- The Singular Value Decomposition of matrix means that an  $m \times n$  real matrix  $A$  is expressed as the product of the following three real matrices

$$A = U \sum V^T$$

where  $U$  is an  $m$ -order orthogonal matrix,  $V$  is an  $n$ -order orthogonal matrix, and  $\sum$  is an  $m \times n$  rectangular diagonal matrix

$$\sum = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p), p = \min\{m, n\}$$

Its diagonal elements are non-negative and satisfy the following conditions:

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$$

- Given any real matrix, its Singular Value Decomposition must exist but is not unique.
- Singular Value Decomposition includes compact SVD and truncated SVD. The compact SVD is a Singular Value Decomposition with the same rank as the the original matrix, while the truncated SVD is the one with a lower rank.
- The Singular Value Decomposition has a clear geometric explanation. It corresponds to three continuous linear transformations: a rotation transformation, a scaling transformation, and another rotation transformation. The first and third rotation transformations are based on the orthonormal bases of space.
- Let the SVD of matrix  $A$  be  $A = U \sum V^T$ , then

$$A^T A = U \left( \sum^T \sum \right) V^T$$

$$A A^T = U \left( \sum \sum^T \right) U^T$$

That is to say, the eigendecomposition of symmetric matrices  $A^T A$  and  $A A^T$  can be represented by the SVD matrix of  $A$ .

- The SVD of matrix  $A$  can be obtained by finding the eigenvalues and eigenvectors of matrix  $A^T A$ . The eigenvectors of  $A^T A$  constitute the columns of an orthogonal matrix  $V$ . The singular value  $\sigma_i$  is obtained from the square root of the eigenvalue  $\lambda_j$  of  $A A^T$ , i.e.,

$$\sigma_j = \sqrt{\lambda_j}, \quad j = 1, 2, \dots, n$$

The values of  $\sigma_i$  are arranged in descending order as diagonal elements to form the diagonal matrix  $\sum$ . Find the left singular vector corresponding to the

positive singular value and then the orthonormal bases of the extended  $A^T$  to form the columns of the orthogonal matrix  $U$ .

7. The Frobenius norm of matrix  $A = [a_{ij}]_{m \times n}$  is defined as

$$\|A\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n (a_{ij})^2 \right)^{\frac{1}{2}}$$

In the set of  $m \times n$  matrices with a rank not exceeding  $k$ , there is an optimal approximation matrix  $X$  of matrix  $A$  in the sense of the Frobenius norm. The truncated SVD with rank  $k$  yields matrix  $A_k$  that achieves this optimal value. The Singular Value Decomposition is the optimal approximation of matrix regarding the Frobenius norm, i.e., in the sense of square loss.

8. Any real matrix  $A$  can be expressed by its outer product expansion

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_n u_n v_n^T$$

where  $u_k v_k^T$  is an  $m \times n$  matrix that is the outer product of the column vector  $u_k$  and the row vector  $v_k^T$ ,  $\sigma_k$  is the singular value, and  $u_k$ ,  $v_k^T$  and  $\sigma_k$  are obtained by the SVD of matrix  $A$ .

## Further Reading

To further understand Singular Value Decomposition (SVD), you can refer to linear algebra textbooks, such as references [3, 4], or watch online open courses, such as “MIT 18.06sc linear algebra”, for which the textbook is available in reference [4]. The SVD is usually performed numerically on a computer, and its numerical computation method can be found in references [1, 2]. The SVD introduced in this chapter is defined on the matrix. The SVD can also be extended to the tensor, having two different definitions. See reference [5] for details.

## Exercises

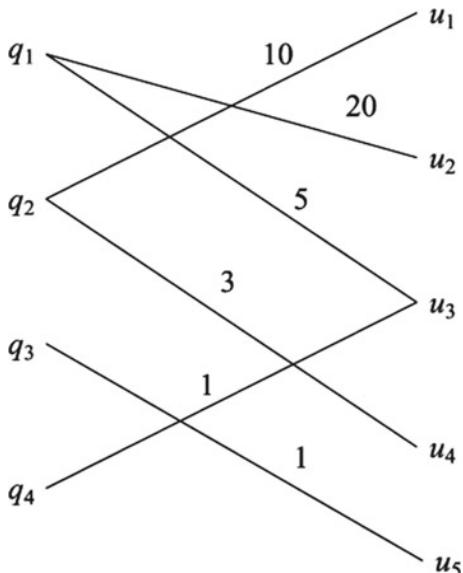
- 15.1. Try to find the Singular Value Decomposition of matrix

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 2 \end{bmatrix}$$

- 15.2. Try to find the Singular Value Decomposition of matrix

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

**Fig. 15.2** Example of search click data



and write down its outer product expansion.

- 15.3. Compare the similarities and differences between the Singular Value Decomposition of matrices and the diagonalization of symmetric matrices.
- 15.4. Prove that any matrix with rank 1 can be represented as the outer product of two vectors, and give an example.
- 15.5. The click data in the search records the query statement submitted by the user during the search, which forms a bipartite graph with the URL of the web page clicked and the number of clicks. Here one node set  $\{q_i\}$  represents the query, the other node set  $\{u_j\}$  represents the URL, the edges represent the click relationship, and the weights on the edges represent the number of clicks. Figure 15.2 is a simplified example of click data, which can be represented by a matrix. Perform Singular Value Decomposition of the matrix, and explain the content of the three obtained matrices.

## References

1. Cline AK, Dhillon IS. Computation of the singular value decomposition, Handbook of linear algebra. CRC Press; 2006.
2. Xu SF. Theory and method of matrix calculation. Peking University Press; 2017.
3. Leon SJ. Linear algebra with applications. Pearson; 2009.
4. Strang G. Introduction to linear algebra. Fourth Edition. Wellesley-Cambridge Press, 2009.
5. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM Rev. 2009;51(3):455–500.

# Chapter 16

## Principal Component Analysis

The principal component analysis (PCA) is a frequently-used unsupervised learning method. PCA uses an orthogonal transformation to convert the observation data represented by linear dependent variables into a few data represented by linear independent variables. Linear independent variables are called the principal components. The number of the principal components is usually less than that of the original variables, so PCA belongs to the dimensionality reduction method. PCA is mainly used to discover the basic structure of data, i.e., the relationship between variables among data. It is a powerful tool for data analysis and is also used for pre-processing other machine learning methods. PCA is a classic method of multivariate statistical analysis. It was first proposed by Pearson in 1901, but only for non-random variables. In 1933, it was extended to random variables by Hotelling.

Section 16.1 of this chapter introduces the basic idea of PCA and describes the definition, theorem, and properties of the overall PCA. Section 16.2 introduces the concept of sample PCA, focusing on the PCA algorithm, including the eigenvalue decomposition method of the covariance matrix and the singular value decomposition method of the data matrix.

### 16.1 Overall Principal Component Analysis

#### 16.1.1 Basic Ideas

In statistical analysis, there may be correlations between data variables, which increases the difficulty of analysis. Therefore, consider replacing the relevant variables with a few irrelevant variables to represent the data, and it is required to be able to retain most of the information in the data.

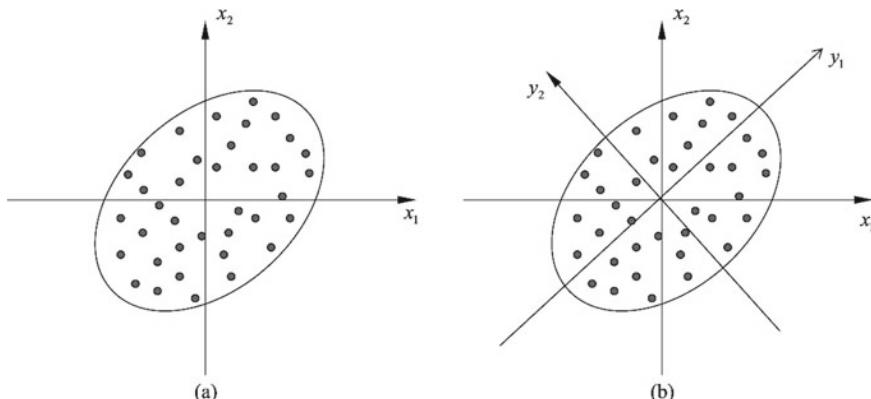
In PCA, we first normalize the given data so that the average value of the data variable is 0 and the variance is 1. After that, the data is orthogonally transformed, and the data originally represented by linear dependent variables are transformed into

data represented by several new linear independent variables through orthogonal transformation. The new variable is the largest sum of variance (information preservation) of the variables in the possible orthogonal transformations, and the variance represents the magnitude of the information of the new variable. The new variables are called the first principal component, the second principal component, and so on in sequence. This is the basic idea of PCA. Through PCA, principal components can be used to approximate the original data, which can be understood as discovering the “basic structure” of the data; the data can also be represented by a few principal components, which can be understood as reducing the dimensionality of the data.

The intuitive explanation of PCA is given below. Points in the real number space (orthogonal coordinate system) represent the samples in the dataset, a coordinate axis of the space represents a variable, and the data obtained after normalization is distributed near the origin. The PCA of the data in the original coordinate system is equivalent to the coordinate system rotation transformation, and the data is projected onto the coordinate axis of the new coordinate system; the first coordinate axis and the second coordinate axis of the new coordinate system respectively represent the first principal components, the second principal components, etc., the square of the coordinate value of the data on each axis represents the variance of the corresponding variable, and this coordinate system has the largest sum of variances on the coordinate axis among all the possible new coordinate systems.

For example, the data is represented by two variables  $x_1$  and  $x_2$ , which exist in a two-dimensional space, and each point represents a sample, as shown in Fig. 16.1a. The data has been normalized, and it can be seen that these data are distributed in an ellipse that is inclined from the bottom left to the top right with the origin as the center. Obviously, the variables  $x_1$  and  $x_2$  in this data are linearly dependent. Specifically, when the value of one of the variables  $x_1$  is known, the prediction of the other variable  $x_2$  is not completely random; vice versa.

PCA performs orthogonal transformation on the data, specifically, the original coordinate system is rotated and transformed, and the data is represented in the new



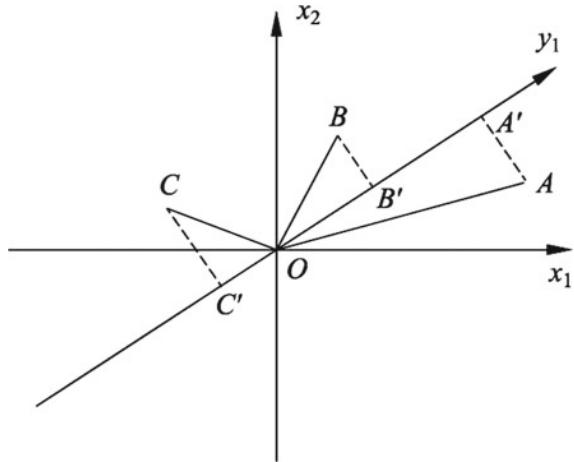
**Fig. 16.1** Example of PCA

coordinate system, as shown in Fig. 16.1b. The data is represented by the variables  $x_1$  and  $x_2$  in the original coordinate system. After the orthogonal transformation, the data is represented by the variables  $y_1$  and  $y_2$  in the new coordinate system. PCA selects the direction with the largest variance (The first principal component) as the first coordinate axis of the new coordinate system, i.e., the axis  $y_1$ , which means selecting the long axis of the ellipse as the first coordinate axis of the new coordinate system; then select the direction orthogonal to the first coordinate axis and the second of the variance (The second principal component) as the second coordinate axis of the new coordinate system, i.e., the axis  $y_2$ , which means that the minor axis of the ellipse is selected as the new coordinate system. In the new coordinate system, the variables  $y_1$  and  $y_2$  in the data are linearly independent. When the value of one of the variables  $y_1$  is known, the prediction of the other variable  $y_2$  is completely random; vice versa. If the PCA only takes the first principal component, that is, the axis  $y_1$  of the new coordinate system, then it is equivalent to projecting the data on the major axis of the ellipse, using this major axis to represent the data, and compressing the data in the two-dimensional space into one-dimensional space.

Let's look at the explanation of the largest variance below. Suppose there are two variables  $x_1$  and  $x_2$ , three sample points  $A$ ,  $B$ , and  $C$ . The samples are distributed in a coordinate system composed of  $x_1$  and  $x_2$  axes, as shown in Fig. 16.2. The coordinate system is rotated and transformed, and a new coordinate axis  $y_1$  is obtained, which represents the new variable  $y_1$ . The sample points  $A$ ,  $B$ , and  $C$  are projected on the axis  $y_1$  to obtain the coordinate values  $A'$ ,  $B'$ ,  $C'$  of the axis  $y_1$ . The sum of squares of coordinate values  $OA^2 + OB^2 + OC^2$  represents the sum of variance of the sample on the variable  $y_1$ . PCA aims to select the variable with the largest variance in the orthogonal transformation as the first principal component, i.e., the axis with the largest sum of squared coordinate values in the rotation transformation. Note that the square sum of the distance between the sample point and the origin in the rotation transformation  $OA^2 + OB^2 + OC^2$  remains unchanged. According to the Pythagorean theorem, the maximum sum of squares  $OA^2 + OB^2 + OC^2$  is equivalent to the minimum sum of squares  $AA'^2 + BB'^2 + CC'^2$  of distance between the sample point and the axis  $y_1$ . Therefore, equivalently, the PCA selects the axis with the smallest sum of squared distances from the sample point in the rotation transformation as the first principal component. The selection of the second principal component, etc., is carried out similarly under the condition that it is orthogonal to the selected coordinate axis.

The PCA performed on the population of data is called the overall PCA, and PCA performed on limited samples is called sample PCA. The former is the basis of the latter. They are introduced separately below.

**Fig. 16.2** The geometric interpretation of principal components



### 16.1.2 Definition and Derivation

Suppose  $x = (x_1, x_2, \dots, x_m)^T$  is an  $m$ -dimensional random variable, and its mean vector is  $\mu$

$$\mu = E(x) = (\mu_1, \mu_2, \dots, \mu_m)^T$$

The covariance matrix is  $\Sigma$

$$\Sigma = \text{cov}(x, x) = E[(x - \mu)(x - \mu)^T]$$

Consider the linear transformation from  $m$ -dimensional random variable  $x$  to  $m$ -dimensional random variable  $y = (y_1, y_2, \dots, y_m)^T$

$$y_i = \alpha_i^T x = \alpha_{1i} x_1 + \alpha_{2i} x_2 + \dots + \alpha_{mi} x_m \quad (16.1)$$

where  $\alpha_i^T = (\alpha_{1i}, \alpha_{2i}, \dots, \alpha_{mi})$ ,  $i = 1, 2, \dots, m$ .

Known from the nature of random variables that

$$E(y_i) = \alpha_i^T \mu, \quad i = 1, 2, \dots, m \quad (16.2)$$

$$\text{var}(y_i) = \alpha_i^T \Sigma \alpha_i, \quad i = 1, 2, \dots, m \quad (16.3)$$

$$\text{cov}(y_i, y_j) = \alpha_i^T \Sigma \alpha_j, \quad i = 1, 2, \dots, m; \quad j = 1, 2, \dots, m \quad (16.4)$$

The definition of the overall principal component is given below.

**Definition 16.1** (*Overall Principal Component*) Given a linear transformation as shown in Eq. (16.1), if they meet the following conditions:

- (1) The coefficient vector  $\alpha_i^T$  is a unit vector, i.e.,  $\alpha_i^T \alpha_i = 1, i = 1, 2, \dots, m$ ;
- (2) The variables  $y_i$  and  $y_j$  are not related to each other, i.e.,  $\text{cov}(y_i, y_j) = 0 (i \neq j)$ ;
- (3) The variable  $y_1$  has the largest variance among all linear transformation of  $x$ ;  $y_2$  has the largest variance among all linear transformation of  $x$  that are not related to  $y_1$ ; Generally,  $y_i$  is the largest variance among all linear transformation of  $x$  that are not related to  $y_1, y_2, \dots, y_{i-1} (i = 1, 2, \dots, m)$ ; At this time,  $y_1, y_2, \dots, y_m$  are called the first principal component, the second principal component, ..., the  $m$ -th principal component, respectively.

Condition (1) in the definition indicates that linear transformation is an orthogonal transformation, and  $\alpha_1, \alpha_2, \dots, \alpha_m$  is a set of orthonormal basis,

$$\alpha_i^T \alpha_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Conditions (2) and (3) give a method of finding the principal component: step 1, among all linear transformations of  $x$

$$\alpha_1^T x = \sum_{i=1}^m \alpha_{i1} x_i,$$

with  $\alpha_1^T \alpha_1 = 1$ , find the one with the largest variance to obtain the first principal component of  $x$ ; step 2, among all linear transformations of  $x$  that are not related to  $\alpha_1^T x$

$$\alpha_2^T x = \sum_{i=1}^m \alpha_{i2} x_i,$$

with  $\alpha_2^T \alpha_2 = 1$ , find the one with the largest variance and obtain the second principal component of  $x$ ; step  $k$ , among all linear transformations of  $x$  unrelated to  $\alpha_1^T x, \alpha_2^T x, \dots, \alpha_{k-1}^T x$

$$\alpha_k^T x = \sum_{i=1}^m \alpha_{ik} x_i$$

with  $\alpha_k^T \alpha_k = 1$ , find the one with the largest variance and get the  $k$ -th principal component of  $x$ ; Continue like this until the  $m$ -th principal component of  $x$  is obtained.

### 16.1.3 Main Properties

First, a theorem on the overall principal component is described. This theorem illustrates the relationship between the overall principal components and the eigenvalues and eigenvectors of the covariance matrix and provides a method for finding the principal component.

**Theorem 16.1** Suppose  $x$  is an  $m$ -dimension random variable,  $\Sigma$  is the covariance matrix, the eigenvalues of  $\Sigma$  are  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m \geq 0$ , respectively, and the unit eigenvector corresponding to the eigenvalue are  $\alpha_1, \alpha_2, \dots, \alpha_m$ , respectively, then the  $k$ -th principal component of  $x$  is

$$y_k = \alpha_k^T x = \alpha_{1k}x_1 + \alpha_{2k}x_2 + \dots + \alpha_{mk}x_m, \quad k = 1, 2, \dots, m \quad (16.5)$$

The variance of the  $k$ -th principal component of  $x$  is

$$\text{var}(y_k) = \alpha_k^T \Sigma \alpha_k = \lambda_k, \quad k = 1, 2, \dots, m \quad (16.6)$$

i.e., the  $k$ -th eigenvalue of the covariance matrix  $\Sigma$ .<sup>1</sup>

**Proof** Use Lagrange multiplier method to find the principal components.

First, find the first principal component  $y_1 = \alpha_1^T x$  of  $x$ , i.e., find the coefficient vector  $\alpha_1$ . According to the definition 16.1, the  $\alpha_1$  of the first principal component is the one maximizing the variance

$$\text{var}(\alpha_1^T x) = \alpha_1^T \Sigma \alpha_1$$

in all linear transformations of  $x$  with  $\alpha_1^T \alpha_1 = 1$ .

To Find the first principal component is to solve the constrained optimization problem:

$$\max_{\alpha_1} \quad \alpha_1^T \Sigma \alpha_1 \quad (16.7)$$

$$\text{s.t. } \alpha_1^T \alpha_1 = 1$$

Define the Lagrange function:

$$\alpha_1^T \Sigma \alpha_1 - \lambda(\alpha_1^T \alpha_1 - 1)$$

<sup>1</sup> If the eigenvalue has multiple roots, the corresponding eigenvectors form a subspace of the  $m$ -dimensional space  $R^m$ . The dimension of the subspace is equal to the number of multiple roots. The unit vector of any orthogonal coordinate system in the subspace can be used as the eigenvector. At this time, the coordinate system is not unique.

where  $\lambda$  is the Lagrange multiplier. Taking the derivative of the Lagrange function with respect to  $\alpha_1$  and setting it to 0, we obtain

$$\Sigma\alpha_1 - \lambda\alpha_1 = 0$$

Therefore,  $\lambda$  is the eigenvalue of  $\Sigma$ ,  $\alpha_1$  is the corresponding unit eigenvector. So, the objective function is

$$\alpha_1^T \Sigma \alpha_1 = \alpha_1^T \lambda \alpha_1 = \lambda \alpha_1^T \alpha_1 = \lambda$$

Suppose  $\alpha_1$  is the unit eigenvector corresponding to the largest eigenvalue  $\lambda_1$  of  $\Sigma$ . Obviously,  $\alpha_1$  and  $\lambda_1$  are the solution of the optimization problem.<sup>2</sup> So,  $\alpha_1^T x$  constitutes the first principal component, and its variance is equal to the maximum eigenvalue of the covariance matrix

$$\text{var}(\alpha_1^T x) = \alpha_1^T \Sigma \alpha_1 = \lambda_1 \quad (16.8)$$

Then find the second principal component  $y_2 = \alpha_2^T x$  of  $x$ . The  $\alpha_2$  of the second principal component is the one maximizing the variance

$$\text{var}(\alpha_2^T x) = \alpha_2^T \Sigma \alpha_2$$

in all linear transformations of  $x$ , under the condition of  $\alpha_2^T \alpha_2 = 1$  and  $\alpha_2^T x$  is not related to  $\alpha_1^T x$ .

To Find the second principal component needs to solve constrained optimization problem.

$$\max_{\alpha_2} \alpha_2^T \Sigma \alpha_2 \quad (16.9)$$

$$\begin{aligned} \text{s.t. } & \alpha_1^T \Sigma \alpha_2 = 0, \quad \alpha_2^T \Sigma \alpha_1 = 0 \\ & \alpha_2^T \alpha_2 = 1 \end{aligned}$$

Note that

$$\alpha_1^T \Sigma \alpha_2 = \alpha_2^T \Sigma \alpha_1 = \alpha_2^T \lambda_1 \alpha_1 = \lambda_1 \alpha_2^T \alpha_1 = \lambda_1 \alpha_1^T \alpha_2$$

and

$$\alpha_1^T \alpha_2 = 0, \quad \alpha_2^T \alpha_1 = 0.$$

Define the Lagrange function

<sup>2</sup> For the convenience of description, the variable and its optimal value are represented by the same symbol here.

$$\alpha_2^T \Sigma \alpha_2 - \lambda(\alpha_2^T \alpha_2 - 1) - \phi \alpha_2^T \alpha_1$$

where  $\lambda$  and  $\phi$  are the Lagrange multiplier. Taking the derivative of the Lagrange function with respect to  $\alpha_2$  and setting it to 0, we obtain

$$2\Sigma\alpha_2 - 2\lambda\alpha_2 - \phi\alpha_1 = 0 \quad (16.10)$$

Multiply the equation by  $\alpha_1^T$  and get

$$2\alpha_1^T \Sigma \alpha_2 - 2\lambda\alpha_1^T \alpha_2 - \phi\alpha_1^T \alpha_1 = 0$$

The first two terms of this formula are 0, and  $\alpha_1^T \alpha_1 = 1$ ,  $\phi = 0$  is derived, so Eq. (16.10) becomes

$$\Sigma\alpha_2 - \lambda\alpha_2 = 0$$

therefore,  $\lambda$  is the eigenvalue of  $\Sigma$ , and  $\alpha_2$  is the corresponding unit eigenvector. Hence, the objective function

$$\alpha_2^T \Sigma \alpha_2 = \alpha_2^T \lambda \alpha_2 = \lambda \alpha_2^T \alpha_2 = \lambda$$

Suppose that  $\alpha_2$  is the unit eigenvector corresponding to the second largest eigenvalue  $\lambda_2$  of  $\Sigma$ , it is obvious that  $\alpha_2$  and  $\lambda_2$  are the solutions of the above optimization problem,<sup>3</sup> so  $\alpha_2^T x$  constitutes the second principal component, and its variance is equal to the second largest eigenvalue of the covariance matrix.

$$\text{var}(\alpha_2^T x) = \alpha_2^T \Sigma \alpha_2 = \lambda_2 \quad (16.11)$$

Generally, the  $k$ -th principal component of  $x$  is  $\alpha_k^T x$ , and  $\text{var}(\alpha_k^T x) = \lambda_k$ , where  $\lambda_k$  is the  $k$ -th eigenvalue of  $\Sigma$ , and  $\alpha_k$  is the corresponding unit eigenvector. It is possible to recursively prove the case of the  $k$ -th principal component starting from the  $k-1$  principal component, which is omitted here.

According to the above method, the first, the second, and up to the  $m-th$  principal components are obtained. The coefficient vectors  $\alpha_1, \alpha_2, \dots, \alpha_m$  are the first, the second, and up to the  $m$ -th unit eigenvectors of  $\Sigma$ , respectively.  $\lambda_1, \lambda_2, \dots, \lambda_m$  are the corresponding eigenvalues, respectively. Moreover, the variance of the  $k$ -th principal component is equal to the  $k$ -th eigenvalue of  $\Sigma$ ,

$$\text{var}(\alpha_k^T x) = \alpha_k^T \Sigma \alpha_k = \lambda_k, \quad k = 1, 2, \dots, m \quad (16.12)$$

Then the theorem is proved.

From theorem 16.1, we obtain

<sup>3</sup> For the convenience of description, the variables and their optimal values are denoted by the same symbol here.

**Deduction 16.1** The necessary and sufficient conditions where the components of  $m$  dimensional random variable  $y = (y_1, y_2, \dots, y_m)^T$  are the first principal component of  $x$  to the  $m$ -th principal component in order are:

- (1)  $y = A^T x$ ,  $A$  is an orthogonal matrix

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1m} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2m} \\ \vdots & \vdots & & \vdots \\ \alpha_{m1} & \alpha_{m2} & \cdots & \alpha_{mm} \end{bmatrix}$$

- (2) The covariance matrix of  $y$  is a diagonal matrix

$$\begin{aligned} \text{cov}(y) &= \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_m) \\ \lambda_1 &\geq \lambda_2 \geq \cdots \geq \lambda_m \end{aligned}$$

where  $\lambda_k$  is the  $k$ -th eigenvalue of  $\Sigma$ ,  $\alpha_k$  is the corresponding unit eigenvector,  $k = 1, 2, \dots, m$ .

In the above proof,  $\lambda_k$  is the  $k$ -th eigenvalue of  $\Sigma$ , and  $\alpha_k$  is the corresponding unit eigenvector, i.e.,

$$\Sigma \alpha_k = \lambda_k \alpha_k, \quad k = 1, 2, \dots, m \quad (16.13)$$

which can be expressed as a matrix

$$\Sigma A = A \Lambda \quad (16.14)$$

Here,  $A = [\alpha_{ij}]_{m \times m}$ ,  $\Lambda$  is a diagonal matrix, and the  $k$ -th diagonal element is  $\lambda_k$ . Since  $A$  is an orthogonal matrix, i.e.,  $A^T A = AA^T = I$ , two formulas can be obtained from Eq. (16.14)

$$A^T \Sigma A = \Lambda \quad (16.15)$$

and

$$\Sigma = A \Lambda A^T \quad (16.16)$$

The properties of the overall principal components are described below:

- (1) The covariance matrix of the overall principal component  $y$  is a diagonal matrix

$$\text{cov}(y) = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_m) \quad (16.17)$$

- (2) The sum of the variance of the overall principal component  $y$  is equal to the sum of the variance of the random variable  $x$ , i.e.,

$$\sum_{i=1}^m \lambda_i = \sum_{i=1}^m \sigma_{ii} \quad (16.18)$$

where  $\sigma_{ii}$  is the variance of the random variable  $x_i$ , i.e., the diagonal element of the covariance matrix  $\Sigma$ . In fact, using Eq. (16.16) and the property of the matrix's trace, we could know that

$$\begin{aligned} \sum_{i=1}^m \text{var}(x_i) &= \text{tr}(\Sigma^T) = \text{tr}(A\Lambda A^T) = \text{tr}(A^T \Lambda A) \\ &= \text{tr}(\Lambda) = \sum_{i=1}^m \lambda_i = \sum_{i=1}^m \text{var}(y_i) \end{aligned} \quad (16.19)$$

- (3) The correlation coefficient  $\rho(y_k, x_i)$  between the  $k$ -th principal component  $y_k$  and the variable  $x_i$  is called factor loading, which represents the correlation between the  $k$ -th principal component  $y_k$  and the variable  $x_i$ . The calculation formula is

$$\rho(y_k, x_i) = \frac{\sqrt{\lambda_k} \alpha_{ik}}{\sqrt{\sigma_{ii}}}, \quad \Lambda k, i = 1, 2, \dots, m \quad (16.20)$$

since

$$\rho(y_k, x_i) = \frac{\text{cov}(y_k, x_i)}{\sqrt{\text{var}(y_k)\text{var}(x_i)}} = \frac{\text{cov}(\alpha_k^T x, e_i^T x)}{\sqrt{\lambda_k} \sqrt{\sigma_{ii}}}$$

where  $e_i$  is the basic unit vector, the  $k$ -th component is 1, and the rest are 0. From the property of covariance

$$\text{cov}(\alpha_k^T x, e_i^T x) = \alpha_k^T \sum e_i = e_i^T \sum \alpha_k = \lambda_k e_i^T \alpha_k = \lambda_k \alpha_{ik}$$

the Formula (16.20) is obtained.

- (4) The factor loading of the  $m$  variables and the  $k$ -th principal component  $y_k$  satisfy

$$\sum_{i=1}^m \sigma_{ii} \rho^2(y_k, x_i) = \lambda_k \quad (16.21)$$

from Eq. (16.20), we have

$$\sum_{i=1}^m \sigma_{ii} \rho^2(y_k, x_i) = \sum_{i=1}^m \lambda_k \alpha_{ik}^2 = \lambda_k \alpha_k^T \alpha_k = \lambda_k$$

(5) The factor loading of the  $m$  principal components and the  $i$ -th variable  $x_i$  satisfies

$$\sum_{k=1}^m \rho^2(y_k, x_i) = 1 \quad (16.22)$$

since  $y_1, y_2, \dots, y_m$  are not related to each other,

$$\rho^2(x_i, (y_1, y_2, \dots, y_m)) = \sum_{k=1}^m \rho^2(y_k, x_i)$$

and because  $x_i$  can be expressed as a linear combination of  $y_1, y_2, \dots, y_m$ , the square of the correlation coefficient between  $x_i$  and  $y_1, y_2, \dots, y_m$  is 1, i.e.,

$$\rho^2(x_i, (y_1, y_2, \dots, y_m)) = 1$$

Therefore, the Eq. (16.22) is obtained.

#### 16.1.4 The Number of Principal Components

The main purpose of PCA is dimensionality reduction, therefore  $k (k \ll m)$  principal components (linearly independent variables) are generally selected to replace  $m$  original variables (linearly dependent variables). In this way, the problem can be simplified and most of the information on the original variables can be retained. The information here refers to the variance of the original variable. Therefore, a theorem is given to show that choosing  $k$  principal components is the best choice.

**Theorem 16.2** For any positive integer  $q$ ,  $1 \ll q \ll m$ , consider the orthogonal linear transformation

$$y = B^T x \quad (16.23)$$

where  $y$  is the  $q$  dimension vector,  $B^T$  is the  $q \times m$  matrix. Let the covariance matrix of  $y$  be

$$\Sigma_y = B^T \Sigma B \quad (16.24)$$

Then the trace  $\text{tr}(\Sigma_y)$  of  $\Sigma_y$  reaches the maximum when  $B = A_q$ , where the matrix  $A_q$  is composed of the first  $q$  columns of the orthogonal matrix  $A$ .

**Proof** Let  $\beta_k$  be the  $k$ -th column of  $B$ . Since the columns of the orthogonal matrix  $A$  constitute the basis of  $m$  dimensional space,  $\beta_k$  can be represented by the columns of  $A$ , i.e.,

$$\beta_k = \sum_{j=1}^m c_{jk} \alpha_j, \quad k = 1, 2, \dots, q$$

Equivalently

$$B = AC \tag{16.25}$$

where  $C$  is  $m \times q$  matrix whose element in the  $j$ -th row and  $k$ -th column is  $c_{jk}$ .

First,

$$B^T \Sigma B = C^T A^T \Sigma A C = C^T \Lambda C = \sum_{j=1}^m \lambda_j c_j c_j^T$$

where  $c_j^T$  is the  $j$ -th line of  $C$ . Therefore

$$\begin{aligned} \text{tr}(B^T \Sigma B) &= \sum_{j=1}^m \lambda_j \text{tr}(c_j c_j^T) \\ &= \sum_{j=1}^m \lambda_j \text{tr}(c_j^T c_j) \\ &= \sum_{j=1}^m \lambda_j c_j^T c_j \\ &= \sum_{j=1}^m \sum_{k=1}^q \lambda_j c_{jk}^2 \end{aligned} \tag{16.26}$$

Second, from Formula (16.25) and the orthogonality of  $A$ , we know that

$$C = A^T B$$

Since  $A$  is orthogonal and the columns of  $B$  are orthogonal,

$$C^T C = B^T A A^T B = B^T B = I_q$$

i.e., the columns of  $C$  are orthogonal. Then

$$\text{tr}(C^T C) = \text{tr}(I_q)$$

$$\sum_{j=1}^m \sum_{k=1}^q c_{jk}^2 = q \quad (16.27)$$

Thus, the matrix  $C$  can be considered as the first  $q$  columns of an  $M$  order orthogonal matrix  $D$ . The rows of the orthogonal matrix  $D$  are also orthogonal, so they satisfy

$$d_j^T d_j = 1, \quad j = 1, 2, \dots, m$$

where  $d_j^T$  is  $j$ -th line of  $D$ . Since the rows of matrix  $D$  include the first  $q$  elements of the row of matrix  $C$ , we have

$$c_j^T c_j \leq 1, \quad j = 1, 2, \dots, m$$

i.e.,

$$\sum_{k=1}^q c_{jk}^2 \leq 1, \quad j = 1, 2, \dots, m \quad (16.28)$$

Note that  $\sum_{k=1}^q c_{jk}^2$  is the coefficient of  $\lambda_j$  in Eq. (16.26). The sum of these coefficients is  $q$  from Eq. (16.27), and it is known from Eq. (16.28) that these coefficients are less than or equal to 1. Since  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_q \geq \dots \geq \lambda_m$ , obviously, when  $c_{jk}$  can be found to satisfy

$$\sum_{k=1}^q c_{jk}^2 = \begin{cases} 1, & j = 1, \dots, q \\ 0, & j = q + 1, \dots, m \end{cases} \quad (16.29)$$

$\sum_{j=1}^m \left( \sum_{k=1}^q c_{jk}^2 \right) \lambda_j$  is the maximum. And when  $B = A_q$ , we have

$$c_{jk} = \begin{cases} 1, & 1 \leq j = k \leq q \\ 0, & \text{else} \end{cases}$$

that satisfies Eq. (16.29). Therefore,  $\text{tr}(\Sigma y)$  reaches the maximum with  $B = A_q$ .

Theorem 16.2 shows that when the linear transformation  $y$  of  $x$  is  $B = A_q$ , The trace  $\text{tr}(\Sigma_y)$  of its covariance matrix  $\Sigma_y$  reaches the maximum, that is to say, when the first  $q$  column of  $A$  is taken and the first  $q$  principal components of  $x$  are taken, the information of the variance of the original variables can be retained to the greatest extent.

**Theorem 16.3** Consider the orthogonal transformation

$$y = B^T x$$

Here  $B^T$  is a  $p \times m$  matrix, and  $A$  and  $\Sigma_y$  are defined as in Theorem 16.2. Then  $\text{tr}(\Sigma_y)$  gets its minimum value when  $B = A_p$ , where the matrix  $A_p$  is composed of the last  $p$  columns of  $A$ .

The proof is similar to Theorem 16.2, which can be proved by interested readers. Theorem 16.3 can be understood as when the last  $P$  columns of  $A$  are discarded, i.e., the last  $p$  principal components of the variable  $x$  are discarded, the information loss in the variance of the original variable is the least.

The above two theorems can be used as the theoretical basis for selecting  $k$  principal components. The method of choosing  $k$  is usually based on the variance contribution rate.

**Definition 16.2** The variance contribution rate of the  $k$ -th principal component  $y_k$  is defined as the ratio of the variance of  $y_k$  to the sum of all variances, which is recorded as  $\eta_k$ .

$$\eta_k = \frac{\lambda_k}{\sum_{i=1}^m \lambda_i} \quad (16.30)$$

The cumulative variance contribution rate of  $k$  principal components  $y_1, y_2, \dots, y_k$  is defined as the ratio of the sum of  $k$  variances to the sum of all variances

$$\sum_{i=1}^k \eta_i = \frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^m \lambda_i} \quad (16.31)$$

Generally,  $k$  is taken to make the cumulative variance contribution rate above the specified percentage, such as more than 70% ~ 80%. The cumulative variance contribution rate reflects the proportion of information reserved by main component, however, it cannot reflect the proportion of information reserved by a certain original variable  $x_i$ . In this case, the contribution rate of  $k$  principal components  $y_1, y_2, \dots, y_k$  to the original variable  $x_i$  is usually used.

**Definition 16.3** The contribution rate of  $k$  principal components  $y_1, y_2, \dots, y_k$  to the original variable  $x_i$  is defined as the square of the correlation coefficient between  $x_i$  and  $(y_1, y_2, \dots, y_k)$ , which is recorded as  $v_i$ .

$$v_i = \rho^2(x_i, (y_1, y_2, \dots, y_k))$$

The calculation formula is as follows:

$$v_i = \rho^2(x_i, (y_1, y_2, \dots, y_k)) = \sum_{j=1}^k \rho^2(x_i, y_j) = \sum_{j=1}^k \frac{\lambda_j \alpha_{ij}^2}{\sigma_{ii}} \quad (16.32)$$

### 16.1.5 The Overall Principal Components of Normalized Variables

In practical problems, different variables may have different dimensions. Directly seeking the principal components sometimes produces unreasonable results. To eliminate this effect, each random variable is often normalized so that its mean is 0 and variance is 1.

Let  $x = (x_1, x_2, \dots, x_m)^T$  be an  $m$ -dimensional random variable,  $x_i$  be the  $i$ -th random variable,  $i = 1, 2, \dots, m$ . Let

$$x_i^* = \frac{x_i - E(x_i)}{\sqrt{\text{var}(x_i)}}, \quad i = 1, 2, \dots, m \quad (16.33)$$

where  $E(x_i)$  and  $\text{var}(x_i)$  are the mean and variance of the random variable  $x_i$ , at which point  $x_i^*$  is the normalized random variable of  $x_i$ .

Obviously, the covariance matrix of normalized random variables is the correlation matrix  $R$ . The PCA is usually performed on the covariance matrix of normalized random variables, i.e., the correlation matrix.

In contrast with the properties of the overall principal components, we can see that the overall principal components of normalized random variables have the following properties:

- (1) The covariance matrix of the principal components of the normalized variable is

$$\Lambda^* = \text{diag}(\lambda_1^*, \lambda_2^*, \dots, \lambda_m^*) \quad (16.34)$$

where  $\lambda_1^* \geq \lambda_2^* \geq \dots \geq \lambda_m^* \geq 0$  is the eigenvalue of the correlation matrix  $R$ .

- (2) The sum of the eigenvalues of the covariance matrix is  $m$

$$\sum_{k=1}^m \lambda_k^* = m \quad (16.35)$$

- (3) The correlation coefficient (factor loading) between the normalized random variable  $x_i^*$  and the principal component  $y_k^*$  is

$$\rho(y_k^*, x_i^*) = \sqrt{\lambda_k^*} e_{ik}^*, \quad k, i = 1, 2, \dots, m \quad (16.36)$$

where  $e_k^* = (e_{1k}^*, e_{2k}^*, \dots, e_{mk}^*)^T$  is the unit eigenvector of the matrix  $R$  corresponding to the eigenvalue  $\lambda_k^*$ .

- (4) The sum of the squares of the correlation coefficients of all normalized random variables  $x_i^*$  and the principal component  $y_k^*$  is equal to  $\lambda_k^*$

$$\sum_{i=1}^m \rho^2(y_k^*, x_i^*) = \sum_{i=1}^m \lambda_k^* e_{ik}^{*2} = \lambda_k^*, \quad k = 1, 2, \dots, m \quad (16.37)$$

- (5) The sum of the squares of the correlation coefficients of the normalized random variable  $x_i^*$  and all principal components  $y_k^*$  is equal to 1

$$\sum_{k=1}^m \rho^2(y_k^*, x_i^*) = \sum_{k=1}^m \lambda_k^* e_{ik}^{*2} = 1, \quad i = 1, 2, \dots, m \quad (16.38)$$

## 16.2 Sample Principal Component Analysis

Section 16.1 describes the overall PCA defined by the sample population. In practical problems, it is necessary to perform PCA on the observation data, which is sample PCA. With the concept of the overall principal component, it is easy to understand the concept of the sample principal component. The sample principal component has the same properties as the overall principal component. Therefore, this section focuses on the algorithm of sample principal components.

### 16.2.1 The Definition and Properties of the Sample Principal Components

Suppose that  $m$ -dimensional random variable  $x = (x_1, x_2, \dots, x_m)^T$  takes  $n$  independent observations,  $x_1, x_2, \dots, x_n$  represents the observation sample, where  $x_j = (x_{1j}, x_{2j}, \dots, x_{mj})^T$  represents the  $j$ -th observation sample,  $x_{ij}$  represents the  $i$ -th variable of the  $j$ -th observation sample,  $j = 1, 2, \dots, n$ . The observation data is represented by the sample matrix  $X$ , denoted as

$$X = [x_1 \ x_2 \ \dots \ x_n] = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad (16.39)$$

Given the sample matrix  $X$ , the sample mean and sample covariance can be estimated. The sample mean vector  $\bar{x}$  is

$$\bar{x} = \frac{1}{n} \sum_{j=1}^n x_j \quad (16.40)$$

The sample covariance matrix  $S$  is

$$S = [s_{ij}]_{m \times m}$$

$$s_{ij} = \frac{1}{n-1} \sum_{k=1}^n (x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j), \quad i, j = 1, 2, \dots, m \quad (16.41)$$

where  $\bar{x}_i = \frac{1}{n} \sum_{k=1}^n x_{ik}$  is the sample mean of the  $i$ -th variable, and  $\bar{x}_j = \frac{1}{n} \sum_{k=1}^n x_{jk}$  is the sample mean of the  $j$ -th variable.

The sample correlation matrix  $R$  is

$$R = [r_{ij}]_{m \times m}, \quad r_{ij} = \frac{s_{ij}}{\sqrt{s_{ii}s_{jj}}}, \quad i, j = 1, 2, \dots, m \quad (16.42)$$

Define the linear transformation of the  $m$ -dimensional vector  $x = (x_1, x_2, \dots, x_m)^T$  to the  $m$ -dimensional vector  $y = (y_1, y_2, \dots, y_m)^T$

$$y = A^T x \quad (16.43)$$

with

$$A = [a_1 \ a_2 \ \dots \ a_m] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{bmatrix}$$

$$a_i = (a_{1i}, a_{2i}, \dots, a_{mi})^T, \quad i = 1, 2, \dots, m$$

Consider any linear transformation of Eq. (16.43)

$$y_i = a_i^T x = a_{1i}x_1 + a_{2i}x_2 + \dots + a_{mi}x_m, \quad i = 1, 2, \dots, m \quad (16.44)$$

where  $y_i$  is the  $i$ -th variable of the  $m$ -dimensional vector  $y$ , corresponding to the sample  $x_1, x_2, \dots, x_n$  with capacity  $n$ ;  $\bar{y}_i$ , the sample mean of  $y_i$  is

$$\bar{y}_i = \frac{1}{n} \sum_{j=1}^n a_i^T x_j = a_i^T \bar{x} \quad (16.45)$$

where  $\bar{x}$  is the sample mean of the random vectors  $x$

$$\bar{x} = \frac{1}{n} \sum_{j=1}^n x_j$$

The sample variance  $\text{var}(y_i)$  of  $y_i$  is

$$\begin{aligned}\text{var}(y_i) &= \frac{1}{n-1} \sum_{j=1}^n (a_i^T x_j - a_i^T \bar{x})^2 \\ &= a_i^T \left[ \frac{1}{n-1} \sum_{j=1}^n (x_j - \bar{x})(x_j - \bar{x})^T \right] a_i = a_i^T S a_i\end{aligned}\quad (16.46)$$

For any two linear transformations  $y_i = a_i^T x$ ,  $y_k = a_k^T x$ , the sample covariance of  $y_i$ ,  $y_k$  corresponding to samples  $x_1, x_2, \dots, x_n$ , with capacity  $n$  is

$$\text{cov}(y_i, y_k) = a_i^T S a_k \quad (16.47)$$

The definition of the sample principal components is now given.

**Definition 16.4** (Sample Principal Components) Given the sample matrix  $X$ .  $y_1 = a_1^T x$ , the first sample principal component is the linear transformation of  $x$  maximizing the sample variance  $a_1^T S a_1$  of  $a_1^T x_j$  ( $j = 1, 2, \dots, n$ ) with  $a_1^T a_1 = 1$ ;  $y_2 = a_2^T x$ , the second sample principal component is the linear transformation of  $x$  maximizing the sample variance  $a_2^T S a_2$  of  $a_2^T x_j$  ( $j = 1, 2, \dots, n$ ) under the condition that  $a_1^T a_2 = 1$  and  $a_1^T S a_2$ , the sample covariance of  $a_2^T x_j$  and  $a_1^T x_j$  ( $j = 1, 2, \dots, n$ ), equals 0; generally, the  $i$ -th sample principal component  $y_i = a_i^T x$  is the linear transformation of  $x$  maximizing the sample variance  $a_i^T S a_i$  of  $a_i^T x_j$  ( $j = 1, 2, \dots, n$ ) under the condition that  $a_i^T a_i = 1$  and the sample covariance  $a_k^T S a_i$  of  $a_k^T x_j$  ( $k < i$ ,  $j = 1, 2, \dots, n$ ) equals 0.

The sample principal components have the same properties as the overall principal components. This is easily seen from the definition of sample principal components. It is sufficient to replace the overall covariance matrix  $\Sigma$  with the sample covariance matrix  $S$ . Theorem 16.2 for the overall principal component and Theorem 16.3 for the sample principal component Theorem 16.2 and Theorem 16.3 still hold for sample principal components. The properties of the sample principal components are not repeated.

When using sample principal components, it is generally assumed that the sample data are normalized, i.e., the sample matrix is transformed as follows:

$$x_{ij}^* = \frac{x_{ij} - \bar{x}_i}{\sqrt{s_{ii}}}, \quad i = 1, 2, \dots, m; \quad j = 1, 2, \dots, n \quad (16.48)$$

where

$$\bar{x}_i = \frac{1}{n} \sum_{j=1}^n x_{ij}, \quad i = 1, 2, \dots, m$$

$$s_{ii} = \frac{1}{n-1} \sum_{j=1}^n (x_{ij} - \bar{x}_i)^2, \quad i = 1, 2, \dots, m$$

For convenience, the normalized variable  $x_{ij}^*$  is still denoted as  $x_{ij}$  in the following, and the normalized sample matrix is still denoted as  $X$ . At this point, the sample covariance matrix  $S$  is the sample correlation matrix  $R$ .

$$R = \frac{1}{n-1} XX^T \quad (16.49)$$

The sample covariance matrix  $S$  is an unbiased estimation of the overall covariance matrix  $\Sigma$ , the sample correlation matrix  $R$  is an unbiased estimation of the overall correlation matrix. The eigenvalues and eigenvectors of  $S$  are the maximum likelihood estimates of the eigenvalues and eigenvectors of  $\Sigma$ . The eigenvalues and eigenvectors of  $R$  are the maximum likelihood estimates of the eigenvalues and eigenvectors of  $\Sigma$ . This issue is not discussed in this book, but the interested reader is referred to books on multivariate statistics, e.g., literature [1].

### 16.2.2 Eigenvalue Decomposition Algorithm of Aorrelation Matrix

The traditional PCA is performed through the eigenvalue decomposition of the covariance matrix or correlation matrix of the data. Nowadays, the commonly used method is performed by Singular Value Decomposition (SVD) of the data matrix. The eigenvalue decomposition of the covariance matrix or correlation matrix of the data is described first.

Given the sample matrix  $X$ , PCA is performed using the sample covariance matrix or the eigenvalue decomposition of the sample correlation matrix of the data. The specific steps are as follows:

- (1) Normalize the observable data according to Eq. (16.48) to obtain the normalized data matrix, which is still denoted by  $X$ .
- (2) Based on the normalized data matrix, calculate the sample correlation matrix  $R$ .

$$R = [r_{ij}]_{m \times m} = \frac{1}{n-1} XX^T$$

where

$$r_{ij} = \frac{1}{n-1} \sum_{l=1}^n x_{il} x_{jl}, \quad i, j = 1, 2, \dots, m$$

- (3) Find the  $k$  eigenvalues of the sample correlation matrix  $R$  and the corresponding  $k$  unit eigenvectors.

Solve the characteristic equation of  $R$

$$|R - \lambda I| = 0$$

to obtain for the  $m$  eigenvalues of  $R$

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$$

Find the number of principal components  $k$  whose variance contribution  $\sum_{i=1}^k \eta_i$  reaches a predetermined value.

Find the unit eigenvectors corresponding to the first  $k$  eigenvalues.

$$a_i = (a_{1i}, a_{2i}, \dots, a_{mi})^T, \quad i = 1, 2, \dots, k$$

- (4) Find the  $k$  sample principal components.

Linear transformation with  $k$  unit eigenvectors as coefficients is implemented to find the  $k$  sample principal components

$$y_i = a_i^T x, \quad i = 1, 2, \dots, k \quad (16.50)$$

- (5) The correlation coefficients  $\rho(x_i, y_j)$  between the  $k$  principal components  $y_j$  and the original variable  $x_i$ , and the contribution  $v_i$  of the  $k$  principal components to the original variable  $x_i$  are calculated.  
(6) Calculate  $k$  principal component values for  $n$  samples

The normalized sample data are substituted into the  $k$  principal components Eq. (16.50) to obtain the principal component values of the  $n$  samples. The  $i$ -th principal component value of the  $j$ -th sample  $x_j = (x_{1j}, x_{2j}, \dots, x_{mj})^T$  is

$$y_{ij} = (a_{1i}, a_{2i}, \dots, a_{mi})(x_{1j}, x_{2j}, \dots, x_{mj})^T = \sum_{l=1}^m a_{li} x_{lj}$$

$$i = 1, 2, \dots, m, \quad j = 1, 2, \dots, n$$

The results obtained from the PCA can be used as input to other machine learning methods. For example, sample points can be clustered by projecting them into a space with principal components as axes and then applying a clustering algorithm.

The following is an example of the PCA method.

**Example 16.1** Suppose there are  $n$  students taking exams in four courses, and the students' test scores are treated as random variables, and the test score data are normalized to obtain the sample correlation matrix  $R$ , which is listed in Table 16.1.

Try to implement PCA on the data.

**Table 16.1** Sample correlation matrix  $R$ 

Courses	Language	Foreign Languages	Mathematics	Physical
Language	1	0.44	0.29	0.33
Foreign Languages	0.44	1	0.35	0.32
Mathematics	0.29	0.35	1	0.60
Physics	0.33	0.32	0.60	1

**Solution** Let the variables  $x_1, x_2, x_3, x_4$  denote the grades of Language, foreign languages, mathematics, and Physics, respectively. Eigenvalue decomposition is performed on sample correlation matrix to obtain the eigenvalues of the correlation matrix, and sequence them in a descending order.

These eigenvalues are the variance contribution of each principal component. Assuming that the cumulative variance contribution of the principal components is required to be greater than 75%, it is sufficient to take only the first two principal components, i.e.,  $k = 2$ , because

$$\frac{\lambda_1 + \lambda_2}{\sum_{i=1}^4 \lambda_i} = 0.76$$

The unit eigenvectors corresponding to the eigenvalues  $\lambda_1, \lambda_2$  are found and listed in Table 16.2. The last column in the table is the variance contribution of the principal components.

From this, the first and second principal components are obtained according to Eq. (16.50).

$$\begin{aligned} y_1 &= 0.460x_1 + 0.476x_2 + 0.523x_3 + 0.537x_4 \\ y_2 &= 0.574x_1 + 0.486x_2 - 0.476x_3 - 0.456x_4 \end{aligned}$$

which is the result of PCA. The variables  $y_1$  and  $y_2$  denote the first and second principal components.

Next, the factor loadings of the first and second principal components and the contribution of the first and the second principal components to the variable  $x_i$  are derived from the eigenvalues and unit eigenvectors. The results are listed in Table 16.3.

As can be seen in Table 16.3, the factor loadings  $\rho(y_1, x_i)$ ,  $i = 1, 2, 3, 4$  corresponding to the first principal component  $y_1$  are all positive, indicating that  $y_1$  can be

**Table 16.2** Variance contribution of unit eigenvectors and principal components

Project	$x_1$	$x_2$	$x_3$	$x_4$	Variance contribution rate
$y_1$	0.460	0.476	0.523	0.537	0.543
$y_2$	0.574	0.486	-0.476	-0.456	0.218

**Table 16.3** Factor loadings and contribution rates of principal components

Projects	$x_1$	$x_2$	$x_3$	$x_4$
$y_1$	0.678	0.701	0.770	0.791
$y_2$	0.536	0.453	-0.444	-0.425
The contribution of $y_1, y_2$ to $x_i$	0.747	0.697	0.790	0.806

improved by improving the performance in each course, i.e., the first principal component  $y_1$  reflects the overall student performance. It can also be seen that the factor loadings have similar values, and  $\rho(y_1, x_4)$  has the largest value, which indicates that the physical grade is the most important part of the overall grade.

The second principal component  $y_2$  corresponds to the factor loadings  $\rho(y_2, x_i), i = 1, 2, 3, 4$  with positive and negative values, positive for language and foreign language and negative for mathematics and physics, indicating that any increase in performance in arts can increase  $y_2$ , while any increase in performance in science can decrease  $y_2$ , i.e., the second principal component  $y_2$  reflects the relationship between students' performance in liberal arts and science.

The factor loadings of the primary variables  $x_1, x_2, x_3, x_4$  (for language, foreign language, mathematics, and physics, respectively) and the principal components  $y_1, y_2$  (for overall performance, and liberal arts vs. science performance, respectively) are represented in a plane coordinate system in Fig. 16.3. The relationship between the variables can be seen in the figure. Four primary variables are clustered into two categories; language and foreign language with similar factor loadings are in one category, and mathematics and physics are in one category, with the former reflecting the performance of liberal arts courses and the latter reflecting the performance of science courses.

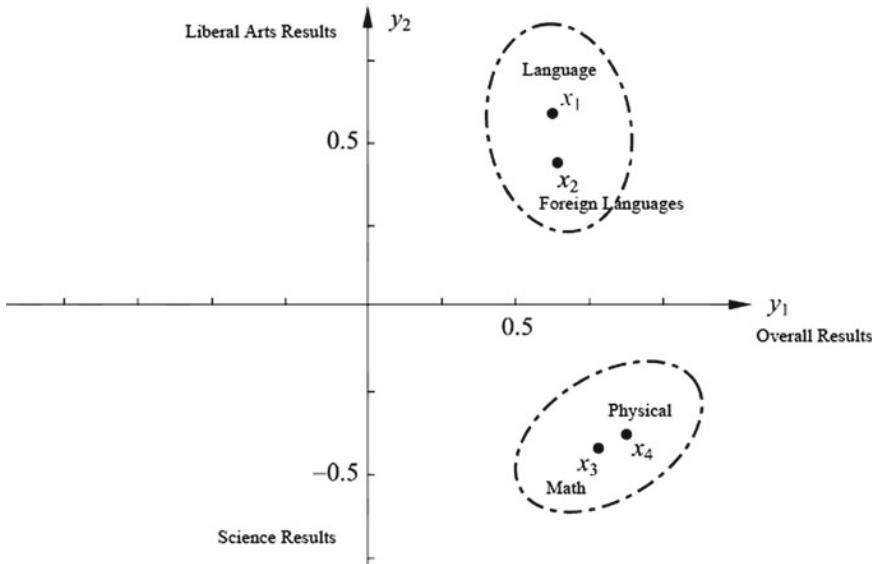
### 16.2.3 Singular Value Decomposition Algorithm for Data Matrix

Given the sample matrix  $X$ , PCA is performed using data matrix singular value decomposition. The specific procedure is as follows. Here it is assumed that there are  $k$  principal components.

According to Eq. (15.19), for an  $m \times n$  real matrix  $A$ , suppose its rank is  $r$  and  $0 < k < r$ , the truncated singular value decomposition of the matrix  $A$  can be performed

$$A \approx U_k \Sigma_k V_k^T$$

where  $U_k$  is an  $m \times k$  matrix,  $V_k$  is an  $n \times k$  matrix,  $\Sigma_k$  is a  $k$ -order diagonal matrix;  $U_k, V_k$  are obtained by taking the first  $k$  columns of the matrix  $U, V$  of the complete singular value decomposition of  $A$ , respectively, and  $\Sigma_k$  is obtained by taking the first



**Fig. 16.3** Distribution of factor loadings

$k$  diagonal elements of the matrix  $\Sigma$  of the complete singular value decomposition of  $A$ .

Define a new  $n \times m$  matrix  $X'$

$$X' = \frac{1}{\sqrt{n-1}} X^T \quad (16.51)$$

Each column of  $X'$  has a mean value of zero. It is not difficult to know that

$$\begin{aligned} X'^T X' &= \left( \frac{1}{\sqrt{n-1}} X^T \right)^T \left( \frac{1}{\sqrt{n-1}} X^T \right) \\ &= \frac{1}{n-1} X X^T \end{aligned} \quad (16.52)$$

That is,  $X'^T X'$  is equal to the covariance matrix  $S_X$  of  $X$ .

$$S_X = X'^T X' \quad (16.53)$$

PCA can be attributed to finding the eigenvalues and the corresponding unit eigenvectors of the covariance matrix  $S_X$ , so the problem is transformed into finding the eigenvalues and the corresponding unit eigenvectors of the matrix  $X'^T X'$ .

Suppose that the truncated singular value decomposition of  $X'$  is  $X' = U \Sigma V^T$ , then the column vector of  $V$  is the unit eigenvector of  $S_X = X'^T X'$ . Therefore, the column vectors of  $V$  constitute the orthogonal right-angle coordinate system of

the principal components of  $X$ . Thus, finding the principal component of  $X$  can be achieved by finding the singular value decomposition of  $X'$ . The specific algorithm is as follows.

**Algorithm 16.1 (PCA algorithm)** Input:  $m \times n$  sample matrix  $X$  whose elements in each row have mean zero;

Output:  $k \times n$  sample principal component matrix  $Y$ .

Parameters: Number of principal components  $k$ .

- (1) Construct a new  $n \times m$  matrix

$$X' = \frac{1}{\sqrt{n-1}} X^T$$

The mean value of each column of  $X'$  is zero.

- (2) The truncated singular value decomposition of matrix  $X'$  yields

$$X' = U \Sigma V^T$$

There are  $k$  singular values and singular vectors. The product of the matrices  $V^T$  and  $X$  constitutes the sample principal component matrix.

- (3) Find the  $k \times n$  sample principal component matrix

$$Y = V^T X$$

## Summary

1. Suppose that  $x$  is an  $m$ -dimensional random variable with mean  $\mu$  and covariance matrix  $\Sigma$ .

Consider the linear transformation from an  $m$ -dimensional random variable  $x$  to an  $m$ -dimensional random variable  $y$

$$y_i = \alpha_i^T x = \sum_{k=1}^m \alpha_{ki} x_k, i = 1, 2, \dots, m$$

Among them,  $\alpha_i^T = (\alpha_{1i}, \alpha_{2i}, \dots, \alpha_{mi})$ .

The linear transformation is called the overall principal component if it satisfies the following conditions:

- (1)  $\alpha_i^T \alpha_i = 1, i = 1, 2, \dots, m$ ;
- (2)  $\text{cov}(y_i, y_j) = 0 (i \neq j)$ ;
- (3) The variable  $y_1$  is the one with the largest variance among all linear transformations of  $x$ ;  $y_2$  is the one with the largest variance among all linear transformations of  $x$  that are uncorrelated with  $y_1$ . In general,  $y_i$  is the

largest variance among all linear transformations of  $x$  that are uncorrelated with  $y_1, y_2, \dots, y_{i-1}$ , ( $i = 1, 2, \dots, m$ ); then  $y_1, y_2, \dots, y_m$  are called the first principal component, the second principal component, ..., and the  $m$ -th principal component of  $x$ .

2. Suppose  $x$  is an  $m$ -dimensional random variable whose covariance matrix is  $\Sigma$  and the eigenvalues of  $\Sigma$  are  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m \geq 0$ , and the unit eigenvectors corresponding to the eigenvalues are  $\alpha_1, \alpha_2, \dots, \alpha_m$ , then the  $i$ -th principal component of  $x$  can be written as

$$y_i = \alpha_i^T x = \sum_{k=1}^m \alpha_{ki} x_k, i = 1, 2, \dots, m$$

Besides, the variance of the  $i$ -th principal component of  $x$  is the  $i$ -th eigenvalue of the covariance matrix  $\Sigma$ , i.e.,

$$\text{var}(y_i) = \alpha_i^T \Sigma \alpha_i = \lambda_i$$

3. The principal components have the following properties:

The covariance matrix of the principal component  $y$  is a diagonal matrix

$$\text{cov}(y) = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_m)$$

The sum of the variances of the principal component  $y$  is equal to the sum of the variances of the random variable  $x$

$$\sum_{i=1}^m \lambda_i = \sum_{i=1}^m \sigma_{ii}$$

where  $\sigma_{ii}$  is the variance of  $x_i$ , i.e., the diagonal elements of covariance matrix  $\Sigma$ .

The correlation coefficient  $\rho(y_k, x_i)$  between principal component  $y_k$  and variable  $x_i$  is called the factor loading, which represents the correlation between the  $k$ -th principal component  $y_k$  and the variable  $x_i$ , i.e., the contribution of  $y_k$  to  $x_i$ .

$$\rho(y_k, x_i) = \frac{\sqrt{\lambda_k} \alpha_{ik}}{\sqrt{\sigma_{ii}}}, \quad k, i = 1, 2, \dots, m$$

4. Sample PCA is the PCA based on sample covariance matrix.

The sample matrix is given as

$$X = [x_1 \ x_2 \ \cdots \ x_n] = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

where  $x_j = (x_{1j}, x_{2j}, \dots, x_{mj})^T$  is the  $j$ -th independent observation sample of  $x$ ,  $j = 1, 2, \dots, n$ .

The sample covariance matrix is

$$S = [s_{ij}]_{m \times m}, \quad s_{ij} = \frac{1}{n-1} \sum_{k=1}^n (x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)$$

$$i = 1, 2, \dots, m, \quad j = 1, 2, \dots, m$$

where  $\bar{x}_i = \frac{1}{n} \sum_{k=1}^n x_{ik}$ .

Given the sample data matrix  $X$ , consider the linear transformation of vector  $x$  to  $y$ .

$$y = A^T x$$

Here,

$$A = [a_1 \ a_2 \ \cdots \ a_m] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{bmatrix}$$

If the linear transformation satisfies the following conditions, it is called sample principal component. The first principal component  $y_1 = a_1^T x$  of a sample is a linear transformation of  $x$  maximizing the sample variance  $a_1^T S a_1$  of  $a_1^T x_j$  ( $j = 1, 2, \dots, n$ ) under the condition of  $a_1^T a_1 = 1$ ; The second principal component  $y_2 = a_2^T x$  is a linear transformation of  $x$  maximizing the sample variance  $a_2^T S a_2$  of  $a_2^T x_j$  ( $j = 1, 2, \dots, n$ ) under the conditions that  $a_2^T a_2 = 1$  and the sample covariance  $a_1^T S a_2$  of  $a_2^T x_j$  and  $a_1^T x_j$  ( $j = 1, 2, \dots, n$ ) equals 0; Generally, the  $i$ -th principal component  $y_i = a_i^T x$  of a sample is a linear transformation of  $x$  maximizing the sample variance  $a_i^T S a_i$  of  $a_i^T x_j$  ( $j = 1, 2, \dots, n$ ) under the condition that  $a_i^T a_i = 1$  and the sample variance  $a_k^T S a_i$  of  $a_i^T x_j$  and  $a_k^T x_j$  ( $k < i, j = 1, 2, \dots, n$ ) equals 0.

5. There are two main PCA methods, which are implemented by eigenvalue decomposition of the correlation matrix or singular decomposition of the sample matrix.

- (1) Eigenvalue decomposition algorithm of correlation matrix. For the  $m \times n$  sample matrix  $X$ , find the sample correlation matrix

$$R = \frac{1}{n-1} X X^T$$

Then  $k$  eigenvalues of the sample correlation matrix and the corresponding unit eigenvector are obtained to construct the orthogonal matrix

$$V = (v_1, v_2, \dots, v_k)$$

Each column of  $V$  corresponds to a principal component, and the  $k \times n$  sample principal component matrix is obtained

$$Y = V^T X$$

- (2) The Singular Value Decomposition (SVD) algorithm of matrix  $X$ . For a  $m \times n$  sample matrix  $X$

$$X' = \frac{1}{\sqrt{n-1}} X^T$$

The truncated singular value decomposition of the matrix  $X'$  is carried out, and  $k$  singular values and singular vectors are retained to obtain

$$X' = U S V^T$$

Each column of  $V$  corresponds to a principal component, and  $k \times n$  sample principal component matrix  $Y$  is obtained

$$Y = V^T X$$

## Further Reading

For further understanding of PCA, please refer to the literature [1–4]. PCA can be performed implicitly in high-dimensional space by the kernel method, called kernel PCA [5]. PCA is an analysis method about the correlation between a group of variables, and canonical correlation analysis is an analysis method about the correlation between two variable groups [6]. In recent years, robust PCA has been proposed. It is an extension of PCA and suitable for basic structure discovery of severely damaged data [7].

## Exercises

- 16.1 Perform a Principal Components Analysis on the following sample data:

$$X = \begin{bmatrix} 2 & 3 & 3 & 4 & 5 & 7 \\ 2 & 4 & 5 & 5 & 6 & 8 \end{bmatrix}$$

- 16.2 Prove that the sample covariance matrix  $S$  is an unbiased estimation of the variance of the overall covariance matrix  $\Sigma$ .
- 16.3 Let  $X$  be the data normalization sample matrix, then the principal components are equivalent to solving the following optimization problem.

$$\begin{aligned} \min_L \|X - L\|_F \\ s.t. \operatorname{rank}(L) \leq k \end{aligned}$$

Here  $F$  is Frobenius norm, and  $k$  is the number of principal components.  
Why?

## References

1. Fang KT. Practical multivariate statistical analysis. East China Normal University Press; 1989.
2. Xia SW. Introduction to systems engineering. Tsinghua University publishing house; 1995.
3. Jolliffe IT. Principal component analysis, 2nd edn. Wiley; 2002.
4. Shlens J. A tutorial on Principal component analysis. arXiv preprint arXiv: 1404.1100, 2014.
5. Schölkopf B, Smola A, Müller K-R. Kernel principal component analysis. Artificial Neural Networks—ICANN'97. Springer; 1997. p. 583–588.
6. Hardoon DR, Szedmak S, Shawe-Taylor J. Canonical correlation analysis: an overview with application to learning methods. Neural Comput. 2004;16(12):2639–64.
7. Candes EJ, Li XD, Ma Y, et al. Robust principal component analysis? J ACM (JACM). 2011;58(3):11.

# Chapter 17

## Latent Semantic Analysis

Latent Semantic Analysis (LSA) is an unsupervised learning method mainly used for the topic analysis of the text. It is characterized by discovering topic-based semantic relationships between text and words through matrix factorization. LSA was proposed by Deerwester et al. in 1990 and was originally applied to textual information retrieval. It is also known as latent semantic indexing (LSI), which is also widely used in recommender systems, image processing, bio-informatics, and other fields.

In text information processing, the traditional approach represents the semantic content of texts in terms of word vectors and the semantic similarity between texts in terms of the metric of word vector space. LSA aims to solve the problem that the traditional method cannot accurately represent semantics, and tries to discover potential topics from a large amount of text data. The semantic content of texts is represented by the topic vector, and the semantic similarity between texts is represented more accurately by the metric of topic vector space. This is also the basic idea of topic modeling.

LSA uses a non-probabilistic topic analysis model. Specifically, the set of texts is represented as a word-text matrix. A singular value decomposition is performed on the word-text matrix to obtain the topic vector space and the representation of texts in the topic vector space. Singular value decomposition (SVD), the matrix decomposition method introduced in Chap. 15, is characterized by the orthogonality of the decomposed matrices.

Non-negative matrix factorization (NMF) is another method of factorization of matrices, which is characterized by the non-negativity of the decomposed matrices. After the publication of Lee and Sheung's paper [1] in 1999, NMF has attracted much attention and widespread application. NMF can also be used for topic modeling.

Section 17.1 introduces the word vector space model and the topic vector space model, pointing out the need for LSA. Section 17.2 describes the SVD algorithm for LSA. Section 17.3 describes the NMF algorithm.

## 17.1 Word Vector Space and Topic Vector Space

### 17.1.1 Word Vector Space

A core problem of text information processing, such as text information retrieval and text data mining, is to represent the semantic content of texts and perform semantic similarity calculations between texts. The simplest way is to use the vector space model (VSM), which is also known as the word vector space model. The basic idea of the VSM is that, given a text, the “semantics” of the text is represented by a vector, each dimension of which corresponds to a word, and its value is the frequency or weight of the word in the text; the basic assumption is that the occurrence of all words in a text indicates the semantic content of the text; each text in the text set is represented as a vector, existing in a vector space; the metric of vector space, such as inner product or normalized inner product, represents the “semantic similarity” between texts.

For example, the task of text information retrieval is to help the user find the most relevant text to the query when the user raises a query and presents it in a sorted form. The simplest approach is to use a word vector space model that represents the query and text as a vector of words and calculates the inner product of the query vector and the text vector as the semantic similarity by which the text is sorted. Here, the query is viewed as a pseudo-text, and the semantic similarity between the query and the text indicates the relevance of the query to the text.

A strict definition is given below. Given a set  $D = \{d_1, d_2, \dots, d_n\}$  containing  $n$  texts, and a set  $W = \{w_1, w_2, \dots, w_m\}$  with  $m$  words occurring in all texts. The data on the occurrence of words in the text is represented by a word-document matrix, denoted as  $X$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad (17.1)$$

This is an  $m \times n$  matrix, and the element  $x_{ij}$  represents the frequency or weight of the occurrence of word  $w_i$  in the text  $d_j$ . The word-text matrix is a sparse matrix, that is because there are many kinds of words, while the kinds of words appearing in each text are usually small.

The weights are usually expressed in term frequency-inverse document frequency (TF-IDF), which is defined as

$$TFIDF_{ij} = \frac{tf_{ij}}{tf_j} \log \frac{df}{df_i}, \quad i = 1, 2, \dots, m; \quad j = 1, 2, \dots, n \quad (17.2)$$

where  $\text{tf}_{ij}$  is the frequency of word  $w_i$  appearing in text  $d_j$ ,  $\text{tf}_j$  is the sum of the frequencies of all words appearing in text  $d_j$ ,  $\text{df}_i$  is the number of texts containing word  $w_i$ , and  $\text{df}$  is the number of all texts in text set  $D$ . Intuitively, the more frequently a word appears in a text, the more important that word is in that text; the fewer texts a word appears in the entire text collection, the more the word can express the characteristics of the text in which it is located, and the higher the importance; the TF-IDF of a word in a text is the product of two importance levels, indicating the combined importance.

The single-word vector space model uses the information from the word-text matrix directly. The  $j - th$  column vector  $x_j$  of the word-text matrix represents the text  $d_j$

$$X = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{mj} \end{bmatrix}, \quad j = 1, 2, \dots, n \quad (17.3)$$

where  $x_{ij}$  is the weight of word  $w_i$  in text  $d_j$ ,  $i = 1, 2, \dots, m$ . The higher the weight, the higher the importance of the word in that text. At this point, the matrix  $X$  can also be written as  $X = [x_1 \ x_2 \ \cdots \ x_n]$ .

The inner product or normalized inner product (i.e., cosine) of two word vectors represents the semantic similarity between the corresponding texts. Therefore, the similarity between the text  $d_i$  and  $d_j$  is

$$x_i \cdot x_j, \frac{x_i \cdot x_j}{\|x_i\| \|x_j\|} \quad (17.4)$$

where  $\cdot$  denotes the inner product of vectors, and  $\|\cdot\|$  denotes the vector norm.

Intuitively, the more words appear together in two texts, the more similar their semantic contents are. At this time, the more the corresponding non-zero dimension of the word vector, the larger the inner product (the values of the word vector elements are all non-negative), which means that the two texts are more similar in semantic content. Although this model is simple, it can well represent the semantic similarity between texts, which is close to people's judgment of semantic similarity and can meet the needs of applications to a certain extent. It is still widely used in the fields of text information retrieval and text data mining and can be considered a basic principle of text information processing. Note that the semantic similarity of two texts is not determined by whether one or two words appear in both texts, but by the "pattern" of all words appearing together in both texts.

The advantage of the single-word vector space model is the simplicity of the model and the high computational efficiency. Because word vectors are usually sparse, the inner product of two vectors only needs to be calculated in the same non-zero dimension. It requires very little computation and can be done efficiently. The single-word vector space model also has certain limitations, which are reflected in that

	$d_1$	$d_2$	$d_3$	$d_4$
airplane	2			
aircraft		2		
computer			1	
apple			2	3
fruit				1
produce	1	2	2	1

**Fig. 17.1** Word-text matrix example

the inner product similarity may not accurately express the semantic similarity of two texts. Since words in natural languages have polysemy and synonymy, i.e., the same word can represent multiple semantics, and multiple words can represent the same semantics, the similarity calculation based on word vectors has the problem of inaccuracy.

Figure 17.1 gives an example. Word-text matrix, each row represents a word, each column represents a text, and each element of the matrix represents the frequency of the word in the text, with frequency 0 omitted. The text  $d_1$  is not very similar to  $d_2$  in the single-word vector space model. Although the content of the two texts is similar, this is because the synonyms “airplane” and “aircraft” are treated as two separate words, and the word vector space model does not take into account the synonymy of the words. In this case, accurate similarity calculation cannot be performed. On the other hand, the text  $d_3$  is somewhat similar to  $d_4$ , even though the contents of the two texts are not similar. This is because the word “apple” has multiple meanings and can mean “apple computer” and “fruit”. The word vector space model does not consider the multiple meanings of words, and it is unable to calculate the accurate similarity in this case.

### 17.1.2 Topic Vector Space

The semantic similarity of the two texts can be reflected in their topic similarity. There is no strict definition for the so-called topic, it just refers to the content or subject discussed in the text. A text generally contains several topics. If the topics of the two texts are similar, then the semantics of the two texts should also be similar. A topic can be represented by several semantically related words, synonyms (e.g., “airplane” and “aircraft”) can represent the same topic, and polysemous words (e.g., “apple”) can represent different topics. In this way, the topic-based model can solve the problems of the above-mentioned word-based model.

It is conceivable to define a topic vector space model. Given a text, use a vector in the topic space to represent the text, each component of the vector corresponds to a topic, and its value is the weight of the topic in the text. The inner product or normalized inner product of two vectors represents the semantic similarity of the corresponding two texts. Note that the number of topics is usually much smaller than the number of words, and the topic vector space model is more abstract. In fact, LSA is just the method of constructing topic vector space (i.e., the topic analysis method). The word vector space model and topic vector space model can complement each other, and they can be used at the same time in practice.

### 17.1.2.1 Topic Vector Space

Given a text set  $D = \{d_1, d_2, \dots, d_n\}$  and a corresponding word set  $W = \{w_1, w_2, \dots, w_m\}$ . The word-text matrix  $X$  can be obtained.  $X$  constitutes the original word vector space, and each column is the representation of a text in the word vector space.

$$X = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \dots & & \dots \\ x_{m1} & \dots & x_{mn} \end{bmatrix} \quad (17.5)$$

Matrix  $X$  can also be written as  $X = [x_1 \ x_2 \ \dots \ x_n]$ .

Assume that all texts contain a total of  $k$  topics, each topic is represented by an  $m$ -dimensional vector defined on the word set  $W$ , called topic vector, namely

$$t_l = \begin{bmatrix} t_{1l} \\ t_{2l} \\ \vdots \\ t_{ml} \end{bmatrix}, \quad l = 1, 2, \dots, k \quad (17.6)$$

Among them,  $t_{il}$  is the weight of the word  $w_i$  in the topic  $i = 1, 2, \dots, m$ , the greater the weight, the higher the importance of the word in the topic. These  $k$  topic vectors  $t_1, t_2, \dots, t_k$  are expanded into a topic vector space, the dimension is  $k$ . Note that the topic vector space  $T$  is a subspace of the word vector space  $X$ .

The topic vector space  $T$  can also be expressed as a matrix, called the word-topic matrix, namely

$$T = \begin{bmatrix} t_{11} & t_{12} & \dots & t_{1k} \\ t_{21} & t_{22} & \dots & t_{2k} \\ \vdots & \vdots & & \vdots \\ t_{m1} & t_{m2} & \dots & t_{mk} \end{bmatrix} \quad (17.7)$$

The matrix  $T$  can also be written as  $T = [t_1 \ t_2 \ \cdots \ t_k]$ .

### 17.1.2.2 Representation of Text in Topic Vector Space

Now consider the text  $d_j$  of the text set  $D$ , represented by a vector  $x_j$  in the word vector space, project  $x_j$  into the topic vector space  $T$  to obtain a vector  $y_j$  in the topic vector space.  $y_j$  is a  $k$ -dimensional vector, and its expression for

$$y_j = \begin{bmatrix} y_{1j} \\ y_{2j} \\ \vdots \\ y_{kj} \end{bmatrix}, \quad j = 1, 2, \dots, n \quad (17.8)$$

Among them,  $y_{lj}$  is the weight of the text  $d_j$  on the topic  $t_l$ ,  $l = 1, 2, \dots, k$ . The larger the weight, the higher the importance of the topic in the text.

Matrix  $Y$  represents the appearance of topics in the text, called topic-document matrix, namely

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & & \vdots \\ y_{k1} & y_{k2} & \cdots & y_{kn} \end{bmatrix} \quad (17.9)$$

The matrix  $Y$  can also be written as  $Y = [y_1 \ y_2 \ \cdots \ y_n]$ .

### 17.1.2.3 Linear Transformation from Word Vector Space to Topic Vector Space

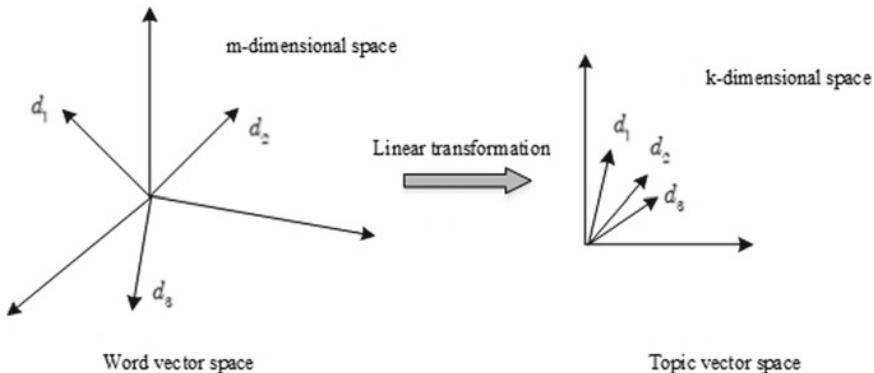
In this way, the text vector  $x_j$  in the word vector space can be approximated by its vector  $y_j$  in the topic space, specifically by the linear combination of  $k$  topic vectors with  $y_j$  as the coefficient.

$$x_j \approx y_{1j}t_1 + y_{2j}t_2 + \cdots + y_{kj}t_k =, \quad j = 1, 2, \dots, n \quad (17.10)$$

Therefore, the word-text matrix  $X$  can be approximately expressed as the product of the word-topic matrix  $T$  and the topic-text matrix  $Y$ . This is LSA.

$$X \approx TY \quad (17.11)$$

Intuitively, LSA is to transform the text representation in the word vector space into a representation in the topic vector space through linear transformation, as shown in



**Fig. 17.2** The text representation in the word vector space is transformed into a topic space representation through a linear transformation

$$m \begin{bmatrix} X \\ \vdots \\ n \end{bmatrix} \approx m \begin{bmatrix} T \\ \vdots \\ k \end{bmatrix} \times k \begin{bmatrix} Y \\ \vdots \\ n \end{bmatrix}$$

**Fig. 17.3** LSA is realized by matrix factorization. Word-text matrix  $X$  can be approximately expressed as the product of word topic matrix  $T$  and topic-text matrix  $Y$

Fig. 17.2. This linear transformation is embodied in the form of matrix factorization (17.11). Figure 17.3 schematically shows the matrix factorization for LSA.

In the original word vector space, the similarity between two texts  $d_i$  and  $d_j$  can be represented by the inner product of the corresponding vectors, i.e.,  $x_i \cdot x_j$ . After LSA, in the topic vector space, the similarity between two texts  $d_i$  and  $d_j$  can be represented by the inner product of the corresponding vectors, namely  $y_i \cdot y_j$ .

To perform LSA, it is necessary to determine the content of two parts at the same time. One is the topic vector space  $T$ , two is the representation  $Y$  of text in topic space. To make the product of the two approximate the original matrix data, and this result is completely obtained from the information of topic-text matrix.

## 17.2 Latent Semantic Analysis Algorithm

In LSA, SVD is used to decompose the word-text matrix. The left matrix is used as the topic vector space, and the product of the diagonal matrix and the right matrix is used as the representation of the text in the topic vector space.

### 17.2.1 Matrix Singular Value Decomposition Algorithm

#### 17.2.1.1 Word-Text Matrix

Given the text set  $D = \{d_1, d_2, \dots, d_n\}$  and the word set  $W = \{w_1, w_2, \dots, w_m\}$ . The LSA first lists these data into a word-text matrix

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad (17.12)$$

This is a  $m \times n$  matrix, element  $x_{ij}$  represents the frequency or weight of word  $w_i$  in text  $d_j$ .

#### 17.2.1.2 Truncated Singular Value Decomposition

The LSA uses the number of topics  $k$  to conduct truncated singular value decomposition for word-text matrix  $X$ :

$$X \approx U_k \sum_k V_k^T = [u_1 \ u_2 \ \cdots \ u_k] \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \end{bmatrix} \quad (17.13)$$

where  $k \leq n \leq m$ ,  $U_k$  is  $m \times k$  matrix, whose columns are composed of the first  $k$  mutually orthogonal left singular vectors of  $X$ , and  $\sum_k$  is a diagonal square matrix of order  $k$ , the diagonal elements are the first  $k$  largest singular values.  $V_k$  is  $n \times k$  matrix, whose columns are composed of the first  $k$  orthogonal right singular vectors of  $X$ .

### 17.2.1.3 Topic Vector Space

In the truncated singular value decomposition (17.13) of the word-text matrix  $X$ , each column vector  $u_1, u_2, \dots, u_k$  of the matrix  $U_k$  represents a topic, which is called topic vector. The  $k$  topic vectors are expanded into a subspace.

$$U_k = [u_1 \ u_2 \ \cdots \ u_k]$$

It is called topic vector space.

### 17.2.1.4 Topic Space Representation of Text

With topic vector space, we consider the representation of text in topic space. The Eq. (17.13) is denoted as

$$\begin{aligned} X &= [x_1 \ x_2 \ \cdots \ x_n] \approx U_k \sum_k V_k^T \\ &= [u_1 \ u_2 \ \cdots \ u_k] \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & 0 & \\ & 0 & \ddots & \\ & & & \sigma_k \end{bmatrix} \begin{bmatrix} v_{11} & v_{21} & \cdots & v_{n1} \\ v_{12} & v_{22} & \cdots & v_{n2} \\ \vdots & \vdots & & \vdots \\ v_{1k} & v_{2k} & \cdots & v_{nk} \end{bmatrix} \\ &= [u_1 \ u_2 \ \cdots \ u_k] \begin{bmatrix} \sigma_1 v_{11} & \sigma_1 v_{21} & \cdots & \sigma_1 v_{n1} \\ \sigma_2 v_{12} & \sigma_2 v_{22} & \cdots & \sigma_2 v_{n2} \\ \vdots & \vdots & & \vdots \\ \sigma_k v_{1k} & \sigma_k v_{2k} & \cdots & \sigma_k v_{nk} \end{bmatrix} \quad (17.14) \end{aligned}$$

where

$$u_l = \begin{bmatrix} u_{1l} \\ u_{2l} \\ \vdots \\ u_{ml} \end{bmatrix}, \quad l = 1, 2, \dots, k$$

According to Eq. (17.14), the vector  $x_j$  of the  $j - th$  column of matrix  $X$  satisfies

$$x_j \approx U_k \left( \sum_k V_k^T \right)_j$$

$$\begin{aligned}
&= [u_1 \ u_2 \ \cdots \ u_k] \begin{bmatrix} \sigma_1 v_{j1} \\ \sigma_2 v_{j2} \\ \vdots \\ \sigma_k v_{jk} \end{bmatrix} \\
&= \sum_{l=1}^k \sigma_l v_{jl} u_l, \quad j = 1, 2, \dots, n
\end{aligned} \tag{17.15}$$

where  $(\sum_k V_k^T)_j$  is  $j-th$  column vector of the matrix  $(\sum_k V_k^T)_j$ . Equation (17.15) is an approximate expression of the text  $d_j$ , which is composed of a linear combination of  $k$  topic vectors  $u_l$ . Each column vector of matrix  $(\sum_k V_k^T)$

$$\begin{bmatrix} \sigma_1 v_{11} \\ \sigma_2 v_{12} \\ \vdots \\ \sigma_2 v_{12} \end{bmatrix}, \begin{bmatrix} \sigma_1 v_{21} \\ \sigma_2 v_{22} \\ \vdots \\ \sigma_k v_{2k} \end{bmatrix}, \dots, \begin{bmatrix} \sigma_1 v_{n1} \\ \sigma_2 v_{n2} \\ \vdots \\ \sigma_k v_{nk} \end{bmatrix}$$

It is the representation of a text in the topic vector space.

In conclusion, the LSA can be done by SVD of word-text matrix

$$X \approx U_k \sum_k V_k^T = U_k \left( \sum_k V_k^T \right) \tag{17.16}$$

Get the topic space  $U_k$  and the text representation in the topic space  $(\sum_k V_k^T)$ .

## 17.2.2 Examples

Here is an example of LSA.<sup>1</sup> Suppose there are 9 texts and 11 words, and the word text matrix  $X$  is  $11 \times 9$  matrix, the elements of the matrix are the frequency of words in the text, expressed as follows:

Index words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
Book			1	1					
Dads						1			1
Dummies		1						1	
Estate				1			1		1

(continued)

<sup>1</sup> <http://www.puffinwarellc.com/index.php/news-and-articles/articles/33-latent-semantic-analysis-tutorial.html?showall=1>.

(continued)

Index words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
Guide	1					1			
Investing	1	1	1	1	1	1	1	1	1
Market	1		1						
Real							1		1
Rich						2			1
Stock	1		1					1	
Value				1	1				

Perform LSA. Implement truncated singular value decomposition of the matrix. Assuming that the number of topics is 3, the truncated singular value decomposition result of the matrix is

	0	0.15	-0.27	0.04									
Dads	0.24	0.38	-0.09										
Dummies	0.13	-0.17	0.07										
Estate	0.18	†	0.45										
Guide	0.22	0.09	-0.46	3.91	0	0							
Investing	0.74	-0.21	0.21	*	0	2.61	0						
Market	0.18	-0.30	-0.28	*	0	0	2.00						
Real	0.18	0.19	0.45	*									
Rich	0.36	0.59	-0.34										
Stock	0.25	-0.42	0.28										
Value	0.12	-0.14	0.23										

\*

	T1	T2	T3	T4	T5	T6	T7	T8	T9
	0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
*	-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
	-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0.00	0.34

It can be seen that the left matrix  $U_3$  has 3 column vectors (left singular vectors). The values of the first column vector  $u_1$  are all positive, and the values of the second column vector  $u_2$  and the third column vector  $u_3$  are either positive or negative. The elements of the diagonal matrix  $\Sigma_3$  in the middle are 3 singular values (positive values) from large to small. The right matrix is  $V_3^T$ , and its transposed matrix  $V_3$  also has 3 column vectors (right singular vectors). The values of the first column vector  $v_1$  are also positive, and the values of the second column vector  $v_2$  and the third column vector  $v_3$  are positive or negative.

Now, multiply  $\Sigma_3$  and  $V_3^T$ , the whole becomes the form of the product of two matrices

$$\begin{aligned} X &\approx U_3 \left( \sum_3 V_3^T \right) \\ &= \begin{bmatrix} 0.15 & -0.27 & 0.04 \\ 0.24 & 0.38 & -0.09 \\ 0.13 & -0.17 & 0.07 \\ 0.18 & 0.19 & 0.45 \\ 0.22 & 0.09 & -0.46 \\ 0.74 & -0.21 & 0.21 \\ 0.18 & -0.30 & 0.28 \\ 0.18 & 0.19 & 0.45 \\ 0.36 & 0.59 & -0.34 \\ 0.25 & -0.42 & -0.28 \\ 0.12 & -0.14 & 0.23 \end{bmatrix} \\ &\quad \begin{bmatrix} 1.37 & 0.86 & 1.33 & 1.02 & 0.86 & 1.92 & 1.09 & 1.13 & 1.72 \\ -0.84 & -0.39 & -1.20 & -0.63 & -0.37 & 1.44 & 0.18 & -0.81 & 1.15 \\ -0.82 & 0.28 & -0.32 & 0.50 & 0.44 & -1.02 & 1.10 & 0.00 & 0.68 \end{bmatrix} \end{aligned}$$

The matrix  $U_3$  has 3 column vectors, representing 3 topics, and matrix  $U_3$  represents topic vector space. The matrix  $(\sum_3 V_3^T)$  has 9 column vectors, representing 9 texts, and the matrix  $(\sum_3 V_3^T)$  is the representation of the text collection in the topic vector space.

## 17.3 Non-negative Matrix Factorization Algorithm

The NMF can also be used for topic analysis. The NMF is conducted on the word-text matrix, take the left matrix as the topic vector space and the right matrix as the representation of the text in the topic vector space. Note that the word-text matrix is usually non-negative.

### 17.3.1 Non-negative Matrix Factorization

If all the elements of a matrix are non-negative, the matrix is called non-negative, and if  $X$  is a non-negative matrix, it is recorded as  $X \geq 0$ .

Given a non-negative matrix  $X \geq 0$ , find two non-negative matrices  $W \geq 0$  and  $H \geq 0$  such that

$$H \approx WH \quad (17.17)$$

That is, the decomposition of a non-negative matrix  $X$  into the product of two non-negative matrices  $W$  and  $H$  is called NMF. Because it is difficult to achieve the complete equality of  $WH$  and  $X$ , it is only required that  $WH$  and  $X$  are approximately equal.

Suppose that the non-negative matrix  $X$  is an  $m \times n$  matrix, and the non-negative matrices  $W$  and  $H$  are  $m \times k$  matrix and  $k \times n$  matrix, respectively. Assuming  $k < \min(m, n)$ , i.e.,  $W$  and  $H$  are smaller than the original matrix  $X$ , so the NMF is a compression of the original data.

From Eq. (17.17), the  $j - th$  column vector  $x_j$  of matrix  $X$  satisfies

$$\begin{aligned} x_j &\approx Wh_j \\ &= [w_1 \ w_2 \ \cdots \ w_k] \begin{bmatrix} h_{1j} \\ h_{2j} \\ \vdots \\ h_{kj} \end{bmatrix} \\ &= \sum_{l=1}^k h_{lj} w_l, \quad j = 1, 2, \dots, n \end{aligned} \quad (17.18)$$

where  $h_j$  is the  $j - th$  column of matrix  $H$ ,  $w_l$  is the  $l - th$  column of matrix  $W$ ,  $h_{lj}$  is the  $l - th$  element of  $h_j$ ,  $l = 1, 2, \dots, k$ .

Equation (17.18) shows that the  $j - th$  column  $x_j$  of the matrix  $X$  can be approximated by the linear combination of  $k$  columns  $w_l$  of the matrix  $W$ , and the coefficient of the linear combination is the element of the  $j - th$  column  $h_j$  of the matrix  $H$ . Here the column vector of matrix  $W$  is a set of the basis, and the column vector of matrix  $H$  is the linear combination coefficient.  $W$  is called the basis matrix and  $H$  is called the coefficient matrix. NMF aims to use fewer basis vectors and coefficient vectors to represent larger data matrices.

### 17.3.2 Latent Semantic Analysis Model

Given a  $m \times n$  non-negative word-text matrix  $X \geq 0$ . Assuming that the text set contains a total of  $k$  topics, perform NMF on  $X$ . That is, find the non-negative  $m \times k$  matrix  $W \geq 0$  and  $k \times n$  matrix  $H \geq 0$ , such that

$$X \approx WH \quad (17.19)$$

Let  $W = [w_1 \ w_2 \ \cdots \ w_k]$  be the topic vector space,  $w_1, w_2, \dots, w_k$  denote the  $k$  topics of the text set, let  $H = [h_1 \ h_2 \ \cdots \ h_n]$  be the text in the topic vector space,  $h_1, h_2, \dots, h_n$  represents  $n$  texts in the text collection. This is the LSA model based on NMF.

NMF has a very intuitive explanation. Both topic vectors and text vectors are non-negative, corresponding to “pseudo probability distribution”, and the linear combination of vectors means that the local superposition constitutes the whole.

### 17.3.3 Formalization of Non-negative Matrix Factorization

NMF can be formalized as an optimization problem. First define the loss function or cost function.

The first loss function is the square loss. Suppose two non-negative matrices  $A = [a_{ij}]_{m \times n}$  and  $B = [b_{ij}]_{m \times n}$ , the square loss function is defined as

$$\|A - B\|^2 = \sum_{i,j} (a_{ij} - b_{ij})^2 \quad (17.20)$$

Its lower bound is 0, and it reaches the lower bound if and only when  $A = B$ .

Another loss function is divergence. Suppose two non-negative matrices  $A = [a_{ij}]_{m \times n}$  and  $B = [b_{ij}]_{m \times n}$ , the divergence loss function is defined as

$$D(A||B) = \sum_{i,j} \left( a_{ij} \log \frac{a_{ij}}{b_{ij}} - a_{ij} + b_{ij} \right) \quad (17.21)$$

Its lower bound is also 0, and it reaches the lower bound if and only when  $A = B$ .  $A$  and  $B$  are asymmetrical. When  $\sum_{i,j} a_{ij} = \sum_{i,j} b_{ij} = 1$ , the divergence loss function degenerates to Kullback-Leiber divergence or relative entropy. At this time,  $A$  and  $B$  are probability distributions.

Then define the following optimization problem.

The objective function s.t.  $W, H \geq 0$  is the minimization of  $W$  and  $H$ , which satisfies the constraint condition  $W, H \geq 0$ , i.e.,

$$\min_{W,H} \|X - WH\|^2 \quad (17.22)$$

$$\text{s.t. } W, H \geq 0$$

or, the objective function  $D(X\|WH)$  is the minimization of  $W$  and  $H$ , which satisfies the constraint condition  $W, H \geq 0$ , i.e.,

$$\min_{W,H} D(X\|WH) \quad (17.23)$$

$$\text{s.t. } W, H \geq 0$$

### 17.3.4 Algorithm

Consider solving the optimization problem (17.22) and problem (17.23). Since the objective function  $\|X - WH\|^2$  and  $D(X\|WH)$  are only convex functions for one of the variables  $W$  and  $H$ , not for both variables at the same time. Therefore, it is difficult to find the global optimum (minimum value). The local optimum (minimal value) can be found by numerical optimization. The gradient descent method is easier to implement, but the convergence speed is slow. The conjugate gradient method converges fast, but the implementation is more complicated. Lee and Seung proposed a new optimization algorithm based on the “multiplication update rule”, which alternately updates  $W$  and  $H$ . The theoretical basis is the following theorem.

**Theorem 17.1** *The Square Loss  $\|X - WH\|^2$  is non-increasing for the following multiplication update rules*

$$H_{lj} \leftarrow H_{lj} \frac{(W^T X)_{lj}}{(W^T WH)_{lj}} \quad (17.24)$$

$$W_{il} \leftarrow W_{il} \frac{(X H^T)_{il}}{(W H H^T)_{il}} \quad (17.25)$$

*The update of the function does not change if and only if  $W$  and  $H$  are the stable points of the squared loss function.*

**Theorem 17.2** *Divergence loss  $D(X - WH)$  is non-increasing for the following multiplication update rules.*

$$H_{lj} \leftarrow H_{lj} \frac{\sum_i [W_{il} X_{ij} / (WH)_{ij}]}{\sum_i W_{il}} \quad (17.26)$$

$$W_{il} \leftarrow W_{il} \frac{\sum_j [H_{lj} X_{ij} / (WH)_{ij}]}{\sum_j H_{lj}} \quad (17.27)$$

The update of the function does not change if and only if  $W$  and  $H$  are the stable points of the divergence loss function.

Theorem 17.1 and 17.2 give the multiplication update rules. The proof of the theorem can be found in literature [2].

Now describe the algorithm of NFM. Only introduce the algorithm of the first problem (17.22), the algorithm of the second problem (17.23) is similar.

The optimization objective function is  $\|X - WH\|^2$ . For the convenience of multiplying the objective function by 1/2, the optimal solution is the same as the original problem, denoted as

$$J(W, H) = \frac{1}{2} \|X - WH\|^2 = \frac{1}{2} \sum [X_{ij} - (WH)_{ij}]^2$$

Use gradient descent method to solve the problem. First, find the gradient of the objective function

$$\begin{aligned} \frac{\partial J(W, H)}{\partial W_{il}} &= - \sum_j [X_{ij} - (WH)_{ij}] H_{lj} \\ &= -[(XH^T)_{il} - (WHH^T)_{il}] \end{aligned} \quad (17.28)$$

Similarly, we can obtain

$$\frac{\partial J(W, H)}{\partial H_{lj}} = -[(W^T X)_{lj} - (W^T WH)_{lj}] \quad (17.29)$$

Then obtain the update rule of the gradient descent method. From Eqs. (17.28) and (17.29), we have

$$W_{il} = W_{il} + \lambda_{il} [(XH^T)_{il} - (WHH^T)_{il}] \quad (17.30)$$

$$H_{lj} = H_{lj} + \mu_{lj} [(W^T X)_{lj} - (W^T WH)_{lj}] \quad (17.31)$$

where  $\lambda_{il}$  and  $\mu_{lj}$  are the step length. Select

$$*, \mu_{lj} = \frac{H_{lj}}{(W^T WH)_{lj}} \quad (17.32)$$

i.e., get the multiplication update rule

$$W_{il} = W_{il} \frac{(XH^T)_{il}}{(WHH^T)_{il}}, \quad i = 1, 2, \dots, m; \quad l = 1, 2, \dots, k \quad (17.33)$$

$$H_{lj} = H_{lj} \frac{(W^T X)_{lj}}{(W^T WH)_{lj}}, \quad i = 1, 2, \dots, m; \quad l = 1, 2, \dots, k \quad (17.34)$$

Select initial matrix  $W$  and  $H$  as non-negative matrix to ensure that the iterative process and the resulting matrices  $W$  and  $H$  are both non-negative.

The following describes the iterative algorithm for matrix non-negative decomposition based on the multiplication update rule. The algorithm alternately iterates  $W$  and  $H$ , and each iteration normalizes the column vector of  $W$  to make the base vector to unit vector.

### Algorithm 17.1 (Iterative Algorithm for Non-negative Matrix Factorization)

Input: Word-text matrix  $X \geq 0$ , the number of topics in the text set  $k$ , and the maximum number of iterations  $t$ ;

Output: Topic matrix  $W$ , text representation matrix  $H$ .

(1) Initialization

$W \geq 0$ , and normalize each column of  $W$ ;  
 $H \geq 0$ ;

(2) Iteration

Perform the following steps for the number of iterations from 1 to  $t$ :

- (a) Update the elements of  $W$ , i.e., for  $l$  from 1 to  $k$ , for  $i$  from 1 to  $m$ , update  $W_{il}$  according to Eq. (17.33);
- (b) Update the elements of  $H$ , i.e., for  $l$  from 1 to  $k$ , for  $i$  from 1 to  $n$ , update  $H_{lj}$  according to Eq. (17.33);

### Summary

1. The word vector space model expresses the semantic content of the text by word vectors. Take word-text matrix  $X$  as input, where each row corresponds to a word, each column corresponds to a text, and each element represents the frequency or weight of the word in the text (such as TF-IDF).

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

The word vector space model assumes that each column vector of this matrix is a word vector, representing a text, and the inner product or normalized inner product of two word vectors represents the semantic similarity between the texts.

2. The Topic Vector Space model Expresses the Semantic Content of the Text Through topic Vectors. Suppose There is a Topic -Text Matrix

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & & \vdots \\ y_{k1} & y_{k2} & \cdots & y_{kn} \end{bmatrix}$$

Each row corresponds to a topic, each column corresponds to a text, and each element represents the weight of the topic in the text. The topic vector space model assumes that each column vector of this matrix is a topic vector, representing a text. And the inner product or normalized inner product of two topic vectors represents the semantic similarity between the texts. Assume that there is a word-topic matrix  $T T$

$$T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1k} \\ t_{21} & t_{22} & \cdots & t_{2k} \\ \vdots & \vdots & & \vdots \\ t_{m1} & t_{m2} & \cdots & t_{mk} \end{bmatrix}$$

where each row corresponds to a word, each column corresponds to a topic, and each element represents the weight of the word in the topic.

Given a word-topic matrix  $X$

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

The goal of LSA is to find a suitable word-topic matrix  $T$  and topic-text matrix  $Y$ , and approximate the word-text matrix  $X$  as the product of  $T$  and  $Y$ .

$$X \approx TY$$

Equivalently, LSA converts the representation  $X$  of the text in the word vector space through a linear transformation  $T$  into the representation  $Y$  in the topic vector space.

The key of the LSA is to perform the above matrix factorization (topic analysis) on the word-text matrix.

3. The algorithm of LSA is SVD. The truncated singular value decomposition is conducted on the word-text matrix to get

$$X \approx U_k \Sigma_k V_k^T = U_k (\Sigma_k V_k^T)$$

The matrix  $U_k$  represents the topic space, and the matrix  $(\Sigma_k V_k^T)$  is the representation of the text in the topic space.

NMF can also be used for topic analysis. The NMF approximately decomposes the non-negative word-text matrix into the product of two non-negative matrices  $W$  and  $H$ , and obtain

$$X \approx WH$$

The matrix  $W$  represents the topic space, and the matrix  $H$  is the representation of the text in the topic space.

NMF can be expressed as the following optimization problem:

$$\min_{W,H} \|X - WH\|^2$$

$$s.t. \quad W, H \geq 0$$

The algorithm of NMF is iterative. The iterative algorithm of the multiplication update rule alternately updates  $W$  and  $H$ . The essence is the gradient descent method. The iterative process and the resulting matrices  $W$  and  $H$  are guaranteed to be non-negative by defining a special step size and a non-negative initial value.

## Further Reading

The literature [3] is the original paper on LSA, and a related introduction is also available in the literature [4], which focuses on LSA based on matrix SVD. The LSA based on NMF can be found in the literature [1, 2, 5]. There is also a method based on sparse matrix factorization [6]. The latter two methods can be implemented through parallel computing, which greatly improves computational efficiency.

## Exercises

- 17.1 Try to Perform the LSA on the Example in Fig. 17.1 and Observe the Results.
- 17.2 Give the algorithm of Non-negative Matrix Decomposition (LSA) when the loss function is a divergence loss.
- 17.3 Give the computational complexity of two algorithms for LSA, including the SVD and NMF.
- 17.4 List the similarities and Differences Between Latent Semantic Analysis (LSA) and Principal Component Analysis (PCA).

## References

1. Lee DD, Seung HS. Learning the parts of objects by non-negative matrix factorization. *Nature*. 1999;401(6755):788–91.
2. Lee D D, Seung H S. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 2001: 556–562.
3. Deerwester S, Dumais ST, Furnas GW, et al. Indexing by latent semantic analysis. *J Am Soc Inf Sci*. 1990;41(6):391–407.
4. Landauer T K. Latent semantic analysis. In: *Encyclopedia of Cognitive Science*, Wiley, 2006.
5. Xu W, Liu X, Gong Y. Document clustering based on non-negative matrix factorization. *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003: 267–273.
6. Wang Q, Xu J, Li H, et al. Regularized latent semantic Indexing. *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2011: 685–694.

# Chapter 18

## Probabilistic Latent Semantic Analysis

Probabilistic latent semantic analysis (PLSA), also called probabilistic latent semantic indexing (PLSI), is an unsupervised learning method for topic analysis of text collections using probabilistic generative models. The most important feature of the model is that topics are represented by hidden variables; the whole model represents the process of text-generating-topics and topics-generating-words, thus obtaining word-text co-occurrence data; it is assumed that each text is determined by a topic distribution and each topic is determined by a word distribution.

Inspired by latent semantic analysis (LSA), PLSA was proposed by Hofmann in 1999. PLAS is based on probabilistic models, while LSA is based on non-probabilistic models. PLSA was initially used in text data mining and later extended to other fields.

In this chapter, the models of PLSA, including generative and co-occurrence models, are described in Sect. 18.1. The learning strategies and algorithms of PLSA models are then introduced in Sect. 18.2.

### 18.1 Probabilistic Latent Semantic Analysis Model

First, we describe the intuitive explanation of probabilistic latent semantic analysis. Probabilistic latent semantic analysis models include generative models and equivalent co-occurrence models. We first introduce the generative model, then introduce the co-occurrence model, and finally explain the properties of the model.

#### 18.1.1 Basic Ideas

Given a set of texts, each text discusses several topics, and each topic is represented by several words. Probabilistic latent semantic analysis of the text collection can

	doc 1	doc 2	doc 3	doc 4	
word 1	2	2	4	3	
word 2	2	1	5	3	
word 3	1	1	2	0	
word 4	0	1	2	1	

**Fig. 18.1** Visual explanation of probabilistic latent semantic analysis (see color chart in Appendix)

discover the topic of each text and the words of each topic. The topic is latent, which cannot be directly observed from the data.

The text set is converted into text-word co-occurrence data, which is specifically represented as a word-text matrix. Figure 18.1 shows an example of a word-text matrix (see the color diagram in Appendix before the text for details). Each row corresponds to a word, each column corresponds to a text, and each element represents the number of times the word appears in the text. A topic represents semantic content. The text data is generated based on the following probabilistic model (co-occurrence model): first, there is the probability distribution of the topic, then there is the conditional probability distribution of the text under the given conditions of the topic, and the conditional probability distribution of the words under the given conditions of the topic. PLSA is to discover topics represented by latent variables, i.e., latent semantics. Intuitively, the words with similar semantics and the texts with similar semantics are grouped into the same “soft category”, and the topics represent such soft categories. Assuming that there are 3 latent topics, the red, green, and blue boxes in the figure each represents a topic.

### 18.1.2 Generative Model

Suppose there is a word set  $W = \{w_1, w_2, \dots, w_M\}$ , where  $M$  is the number of words; Text (index) set  $D = \{d_1, d_2, \dots, d_N\}$ , where  $N$  is the number of texts; Topic set  $Z = \{z_1, z_2, \dots, z_K\}$ , where  $K$  is the preset number of topics. The random variable  $w$  is selected from the word set; The random variable  $d$  takes the value from the text set, and the random variable  $z$  takes the value from the topic set. Probability distribution  $P(d)$ , conditional probability distribution  $P(z|d)$  and conditional probability distribution  $P(w|z)$  belong to multinomial distribution, where  $P(d)$  represents

the probability of generating text  $d$ ,  $P(z|d)$  represents the probability of generating topic  $z$  from text  $d$ , and  $P(w|z)$  represents the probability of generating word  $w$  from topic  $z$ .

Each text  $d$  has its own topic probability distribution  $P(z|d)$ , and each topic  $z$  has its own word probability distribution  $P(w|z)$ ; in other words, the content of a text is determined by its related topics, and the content of a topic is determined by its related words.

The generative model generates text word co-occurrence data through the following steps:

- (1) According to the probability distribution  $P(d)$ , a text  $d$  is randomly selected from the text (index) set to generate  $N$  texts; For each text, do the following;
- (2) Given the text  $d$ , a topic  $z$  is randomly selected from the topic set according to the conditional probability distribution  $P(z|d)$  to generate a total of  $L$  topics, where  $L$  is the length of the text;
- (3) Under the given condition of topic  $z$ , according to the conditional probability distribution  $P(w|z)$ , a word  $w$  is randomly selected from the word set.

Note that for the convenience of narration, it is assumed that the texts are of equal length, which is not necessary in reality.

In the generative model, word variable  $w$  and text variable  $d$  are observation variables, and topic variable  $z$  is hidden variable. That is to say, what the model generates is a set of word-topic-text triples  $(w, z, d)$ , but what it observes is a set of word text triples  $(w, d)$ , and the observation data is expressed as the form of word-text matrix  $T$ , and the rows of matrix  $T$  represent words, the columns represent text, and the elements represent the number of occurrences of the word text pairs  $(w, d)$ .

From the process of data generation, we can deduce that the generation probability of text-word co-occurrence data  $T$  is the product of the generation probability of all word text pairs  $(w, d)$ :

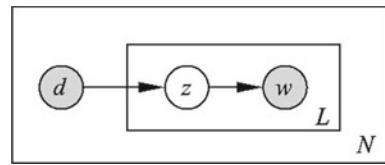
$$P(T) = \prod_{(w,d)} P(w, d)^{n(w,d)} \quad (18.1)$$

Here  $n(w, d)$  denotes the number of occurrences of  $(w, d)$ , and the total number of occurrences of word-text pairs is  $N \times L$ . The generation probability of each word-text pairs  $(w, d)$  is determined by the following formula:

$$\begin{aligned} P(w, d) &= P(d)P(w|d) \\ &= P(d) \sum_z P(w, z|d) \\ &= P(d) \sum_z P(z|d)P(w|z) \end{aligned} \quad (18.2)$$

Formula (18.2) is the definition of generative model.

**Fig. 18.2** Generative model of probabilistic latent semantic analysis



The generative model assumes that the word  $w$  and the text  $d$  are independent under the given condition of topic  $z$ , i.e.,

$$P(w, z|d) = P(z|d)P(w|z) \quad (18.3)$$

The generative model belongs to probabilistic directed graph model, which can be represented by directed graph, as shown in Fig. 18.2. In the figure, the solid circle represents the observed variable, the hollow circle represents the hidden variable, the arrow represents the probability dependence, the box represents multiple repetitions, and the number in the box represents the number of repetitions. The text variable  $d$  is an observation variable, the topic variable  $z$  is an implicit variable, and the word variable  $w$  is an observation variable.

### 18.1.3 Co-occurrence Model

A co-occurrence model equivalent to the above generative model can be defined.

The generation probability of text-word co-occurrence data  $T$  is the product of the generation probabilities of all word-text pairs  $(w, d)$ :

$$P(T) = \prod_{(w,d)} P(w, d)^{n(w,d)} \quad (18.4)$$

The probability of each word-text pairs  $(w, d)$  is determined by the following formula:

$$P(w, d) = \sum_{z \in Z} P(z)P(w|z)P(d|z) \quad (18.5)$$

Formula (18.5) is the definition of co-occurrence model. It is easy to verify that the generative model (18.2) and the co-occurrence model (18.5) are equivalent.

The co-occurrence model assumes that the word  $w$  and the text  $d$  are conditionally independent when the topic  $z$  is given, i.e.,

$$P(w, d|z) = P(w|z)P(d|z) \quad (18.6)$$

**Fig. 18.3** Co-occurrence model of probabilistic latent semantic model

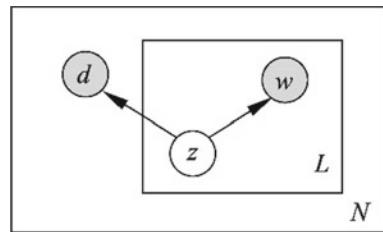


Figure 18.3 shows the co-occurrence model. In the figure, the text variable  $d$  is an observation variable, the word variable  $w$  is an observation variable, and the topic variable  $z$  is an implicit variable. Figure 18.1 is an intuitive explanation of the co-occurrence model.

Although the generative model and the co-occurrence model are equivalent in the sense of the probability formula, they have different properties. The generative model describes the process of text-word co-occurrence data generation, and the co-occurrence model describes the pattern of text-word co-occurrence data. In the generative model (18.2), the word variable  $w$  and the text variable  $d$  are asymmetric, while in the co-occurrence model (18.5), the word variable  $w$  and the text variable  $d$  are symmetric; so the former is also called asymmetric model, and the latter is also called symmetric model. Because of the different forms of the two models, the forms of their learning algorithms are also different.

## 18.1.4 Model Properties

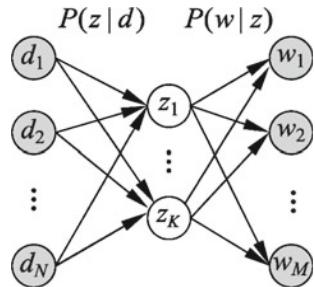
### 18.1.4.1 Model Parameters

If the co-occurrence probability  $P(w|d)$  of words and texts is defined directly, the number of model parameters is  $O(M \cdot N)$ , where  $M$  is the number of words and  $N$  is the number of texts. The number of parameters of the generative model and co-occurrence model of probabilistic latent semantic analysis is  $O(M \cdot K + N \cdot K)$ , where  $K$  is the number of topics. In reality,  $K \ll M$ , so probabilistic latent semantic analysis makes the data more concise through the topic, which reduces the possibility of over fitting in the learning process. Figure 18.4 shows the relationship among text, topic and word in the model.

### 18.1.4.2 Geometric Interpretation of the Model

The geometric explanation of the generative model is given below. The probability distribution  $P(w|d)$  represents the probability that the text  $d$  generates the word  $w$ ,

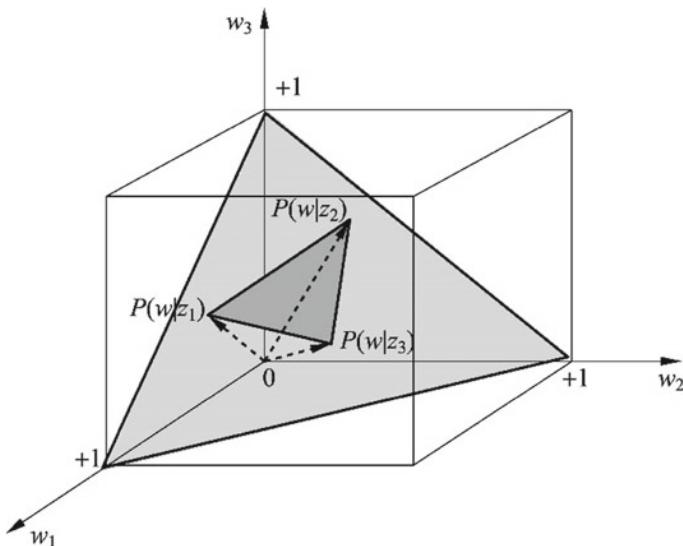
**Fig. 18.4** The relationship between text, topic, and words in probabilistic latent semantic analysis



$$\sum_{i=1}^M P(w_i|d) = 1, \quad 0 \leq P(w_i|d) \leq 1, \quad i = 1, \dots, M$$

It can be represented by the points in the  $(M - 1)$  simplex in the  $M$ -dimensional space. Figure 18.5 shows the situation in three-dimensional space. Each point on the simplex represents a distribution  $P(w|d)$  (distribution parameter vector), all distributions  $P(w|d)$  (distribution parameter vector) are on the simplex, call this  $(M - 1)$  simplex the word simplex.

From Eq. (18.2), it can be seen that the text probability distribution in  $P(w|d)$  the probabilistic latent analysis model (generative model) has the following relationship established:



**Fig. 18.5** Word simplex and topic simplex

$$P(w|d) = \sum_z P(z|d)P(w|z) \quad (18.7)$$

Here the probability distribution  $P(w|z)$  represents the probability of topic  $z$  generating word  $w$ .

The probability distribution  $P(w|z)$  also exists in the  $(M - 1)$  simplex in the  $M$ -dimensional space. If there are  $K$  topics, then there are  $K$  probability distributions  $P(w|z_k)$ ,  $k = 1, 2, \dots, K$ , represented by  $K$  points on the  $(M - 1)$  simplex (refer to Fig. 18.5). Take these  $K$  points as vertices to form a  $(K - 1)$  simplex, which is called topic simplex. The topic simplex is a sub-simplex of the word simplex. Refer to Fig. 18.5.

From Eq. (18.7), the distribution  $P(w|d)$  of the text in the generative model can be represented by the linear combination of the distribution  $P(w|z_k)$ ,  $k = 1, 2, \dots, K$  of  $K$  topics, and the corresponding point of the text is in the  $(K - 1)$  topic simplex formed by the points of  $K$  topics. This is the geometric interpretation of the generative model. Note that usually  $K \ll M$ , the probabilistic latent semantic model exists in a relatively small parameter space. Figure 18.5 shows the situation when  $M = 3$  and  $K = 3$ . When  $K = 2$ , the topic simplex is a line segment, and when  $K = 1$ , the topic simplex is a point.

#### 18.1.4.3 Relationship with Latent Semantic Analysis

The probabilistic latent semantic analysis model (co-occurrence model) can be described in the framework of the latent semantic analysis model. Figure 18.6 shows the latent semantic analysis. The word-text matrix is subjected to singular value decomposition to obtain  $X = U\Sigma V^T$ , where  $U$  and  $V$  are orthogonal matrices, and  $\Sigma$  are non-negative descending diagonal matrices (refer to Chap. 17).

The co-occurrence model (18.5) can also be expressed in the form of the product of three matrices. In this way, the correspondence between probabilistic latent semantic analysis and latent semantic analysis can be seen clearly. The following is the matrix product form of the co-occurrence model:

$$\begin{matrix} X \\ M \times N \end{matrix} = \begin{matrix} U \\ M \times K \end{matrix} \begin{matrix} \Sigma \\ K \times K \end{matrix} \begin{matrix} V^T \\ K \times N \end{matrix}$$

**Fig. 18.6** The relationship between probabilistic latent semantic analysis and latent semantic analysis

$$\begin{aligned}
X' &= U'\Sigma'V'\mathbf{T} \\
X' &= [P(w, d)]_{M \times N} \\
U' &= [P(w|z)]_{M \times K} \\
\Sigma' &= [P(z)]_{K \times K} \\
V' &= [P(d|z)]_{N \times K}
\end{aligned} \tag{18.8}$$

The matrices  $U'$  and  $V'$  in the probabilistic latent semantic analysis model (18.8) are non-negative and normalized, representing the conditional probability distribution, while the matrices  $U$  and  $V$  in the latent semantic analysis model are orthogonal, not necessarily non-negative, and do not mean Probability distributions.

## 18.2 Algorithms for Probabilistic Latent Semantic Analysis

The probabilistic latent semantic analysis model is a model containing hidden variables, it uses EM algorithm for learning. This section introduces the EM algorithm for generative model learning.

The EM algorithm is an iterative algorithm. Each iteration includes two alternate steps: E-step, seeking expectation; M-step, seeking maximum. M-Step is to calculate the  $Q$  function, which is the expectation of the log-likelihood function of complete data on the conditional distribution of incomplete data. The M-step is to maximize the  $Q$  function and update the model parameters. See Chap. 9 for details. The EM algorithm for generating the model is described below.

Suppose the word set is  $W = \{w_1, w_2, \dots, w_M\}$ , the text set is  $D = \{d_1, d_2, \dots, d_N\}$ , and the topic set is  $Z = \{z_1, z_2, \dots, z_K\}$ . Given word-text co-occurrence data  $T = \{n(w_i, d_j)\}$ ,  $i = 1, 2, \dots, M$ ,  $j = 1, 2, \dots, N$ , the goal is to estimate the parameters of the probabilistic latent semantic analysis model (generative model). If maximum likelihood estimation is used, the log-likelihood function is

$$\begin{aligned}
L &= \sum_{i=1}^M \sum_{j=1}^N n(w_i, d_j) \log P(w_i, d_j) \\
&= \sum_{i=1}^M \sum_{j=1}^N n(w_i, d_j) \log \left[ \sum_{k=1}^K P(w_i|z_k) P(z_k|d_j) \right]
\end{aligned}$$

However, the model contains hidden variables, and the optimization of the log-likelihood function cannot be solved analytically. In this case, the EM algorithm is used. The core of applying the EM algorithm is to define the  $Q$  function.

E-step: Calculate the  $Q$  function

The  $Q$  function is the expectation of the log-likelihood function of complete data on the conditional distribution of incomplete data. For the generative model of probabilistic latent semantic analysis, the  $Q$  function is

$$Q = \sum_{k=1}^K \left\{ \sum_{j=1}^N n(d_j) [\log P(d_j) + \sum_{i=1}^M \frac{n(w_i, d_j)}{n(d_j)} \log P(w_i|z_k) P(z_k|d_j)] \right\} P(z_k|w_i, d_j) \quad (18.9)$$

where  $n(d_j) = \sum_{i=1}^M n(w_i, d_j)$  represents the number of words in the text, and  $d_j$ ,  $n(w_i, d_j)$  represents the number of times the word  $w_i$  appears in the text  $d_j$ . The conditional probability distribution  $P(z_k|w_i, d_j)$  represents incomplete data and is a known variable. The product of the conditional probability distribution  $P(w_i|z_k)$  and  $P(z_k|d_j)$  represents complete data and is an unknown variable.

Since the estimation of  $P(d_j)$  can be obtained directly from the data, only the estimation of  $P(w_i|z_k)$  and  $P(z_k|d_j)$  are considered here, and the  $Q$  function can be simplified to the function  $Q'$ .

$$Q' = \sum_{i=1}^M \sum_{j=1}^N n(w_i, d_j) \sum_{k=1}^K P(z_k|w_i, d_j) \log [P(w_i|z_k) P(z_k|d_j)] \quad (18.10)$$

$P(z_k|w_i, d_j)$  in the  $Q'$  function can be calculated according to the Bayesian formula

$$P(z_k|w_i, d_j) = \frac{P(w_i|z_k) P(z_k|d_j)}{\sum_{k=1}^K P(w_i|z_k) P(z_k|d_j)} \quad (18.11)$$

where  $P(z_k|d_j)$  and  $P(w_i|z_k)$  are obtained from the previous iteration.

**M-step: Maximize the Q function**

Solving for the extreme value of the Q function by constrained optimization, when  $P(z_k|d_j)$  and  $P(w_i|z_k)$  are variables. Since the variables  $P(w_i|z_k)$ ,  $P(z_k|d_j)$  form a probability distribution that satisfies the constraint

$$\begin{aligned} \sum_{i=1}^M P(w_i|z_k) &= 1, \quad k = 1, 2, \dots, K \\ \sum_{k=1}^K P(z_k|d_j) &= 1, \quad j = 1, 2, \dots, N \end{aligned}$$

Applying the Lagrange method, introduce Lagrange multipliers  $\tau_k$  and  $\rho_j$  and define the Lagrange function  $\Lambda$

$$\Lambda = Q' + \sum_{k=1}^K \tau_k \left( 1 - \sum_{i=1}^M P(w_i|z_k) \right) + \sum_{j=1}^N \rho_j \left( 1 - \sum_{k=1}^K P(z_k|d_j) \right)$$

The Lagrange function  $\Lambda$  is obtained by taking the partial derivatives of  $P(w_i|z_k)$  and  $P(z_k|d_j)$  respectively, and making them equal to 0. the system of equations.

$$\begin{aligned} & \sum_{j=1}^N n(w_i, d_j) P(z_k|w_i, d_j) - \tau_k P(w_i|z_k), \quad i = 1, 2, \dots, M; \quad k = 1, 2, \dots, K \\ & \sum_{i=1}^M n(w_i, d_j) P(z_k|w_i, d_j) - \rho_j P(z_k|d_j), \quad j = 1, 2, \dots, N; \quad k = 1, 2, \dots, K \end{aligned}$$

Solving the system of equations yields the parameter estimation equation for M steps.

$$P(w_i|z_k) = \frac{\sum_{j=1}^N n(w_i, d_j) P(z_k|w_i, d_j)}{\sum_{m=1}^M \sum_{j=1}^N n(w_m, d_j) P(z_k|w_m, d_j)} \quad (18.12)$$

$$P(z_k|d_j) = \frac{\sum_{i=1}^M n(w_i, d_j) P(z_k|w_i, d_j)}{n(d_j)} \quad (18.13)$$

The following algorithms are summarized:

**Algorithm 18.1** (*EM algorithm for parameter estimation of probabilistic latent semantic models*)

**Input:** Let the set of words be  $W = \{w_1, w_2, \dots, w_M\}$ , the set of texts be  $D = \{d_1, d_2, \dots, d_N\}$ , and the set of topics is  $Z = \{z_1, z_2, \dots, z_K\}$ , and the co-occurrence data  $\{n(w_i, d_j)\}$ ,  $i = 1, 2, \dots, M$ ,  $j = 1, 2, \dots, N$ ;

**Output:**  $P(w_i|z_k)$  and  $P(z_k|d_j)$ .

1. Set the initial values of the parameters  $P(w_i|z_k)$  and  $P(z_k|d_j)$ .
2. Iteratively perform the following E steps and M steps until convergence.

E-step:

$$P(z_k|w_i, d_j) = \frac{P(w_i|z_k) P(z_k|d_j)}{\sum_{k=1}^K p(w_i|z_k) P(z_k|d_j)}$$

M-step:

$$P(w_i|z_k) = \frac{\sum_{j=1}^N n(w_i, d_j) P(z_k|w_i, d_j)}{\sum_{m=1}^M \sum_{j=1}^N n(w_m, d_j) P(z_k|w_m, d_j)}$$

$$P(z_k|d_j) = \frac{\sum_{i=1}^M n(w_i, d_j) P(z_k|w_i, d_j)}{n(d_j)}$$

## Summary

1. Probabilistic latent semantic analysis is a method for thematic analysis of text sets using probabilistic generative models. PLSA is inspired by latent semantic analysis, and the two can be linked by matrix factorization.

Given a set of texts, PLSA can be performed to obtain the conditional topic generation for each text probability distribution and the conditional probability distribution of words generated by each topic.

Models for PLSA are generative models and equivalent co-occurrence models. Its learning strategy is the maximum likelihood estimation of observation data, and its learning algorithm is the EM algorithm.

2. The generative model represents the process of text-generating-topics and topics-generating-words to obtain word-text co-occurrence data. Assume that each text is determined by a topic distribution, and each topic is determined by a word distribution. The word variable  $w$  and text variable  $d$  are observed variables, and topic variable  $z$  is the hidden variable. The generative model is defined as follows:

$$P(T) = \prod_{(w,d)} P(w, d)^{n(w,d)}.$$

$$P(w, d) = P(d)P(w|d) = P(d) \sum_z P(z|d)P(w|z)$$

3. The co-occurrence model describes the pattern of co-occurrence data possession of text words. The co-occurrence model is defined as follows::

$$P(T) = \prod_{(w,d)} P(w, d)^{n(w,d)}$$

$$P(w, d) = \sum_{z \in Z} P(z)P(w|z)P(d|z)$$

6. The number of parameters of the PLSA model is  $O(M \cdot K + N \cdot K)$ . In reality,  $K \ll M$ . Thus, PLSA achieves data compression by a more concise representation of the data through topics.
7. The probability distribution  $P(w|d)$  in the model can be represented by the simplexes in the parameter space. The  $M$ -dimensional parameter space is represented by the word simplexes. The word simplex in the parameter space represents

the distribution of all possible texts, and the topic simplex in it represents the distribution of all possible texts under K topic mimics represent the distribution of all possible texts under K topic definitions. topic simplexes are subsets of word simplexes, which represent the latent semantic space.

8. The learning of PLSA usually adopts the EM algorithm. The parameters  $P(w|z)$  and  $P(z|d)$  of the model are learned by iteratively, while  $P(d)$  can be derived directly statistically.

### Further Reading

The original literature on probabilistic latent semantic analysis is available in [1–3]. In the literature [4], the authors discuss probabilistic latent semantic analysis in relation to non-negative matrix factorization.

### Exercises

- 18.1 Prove that the generative model is equivalent to the co-occurrence model.
- 18.2 Derive the EM algorithm for the co-occurrence model.
- 18.3 Perform probabilistic latent semantic analysis on the following text dataset.

Index words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
Book			1	1					
Dads						1			1
Dummies		1						1	
Estate							1		1
Guide	1					1			
Investing	1	1	1	1	1	1	1	1	1
Market	1		1						
Real							1		1
Rich						2			1
Stock	1		1					1	
Value				1	1				

### References

1. Hofmann T. Probabilistic latent semantic analysis. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence. 1999. p. 289–96.
2. Hofmann T. Probabilistic latent semantic indexing. In: Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval. 1999. p. 50–7.

3. Hofmann T. Unsupervised learning by probabilistic latent semantic analysis. *Mach Learn.* 2001;42(1):177–96.
4. Ding C, Li T, Peng W. On the equivalence between non-negative matrix factorization and probabilistic latent semantic Indexing. *Comput Stat Data Anal.* 2008;52(8):3913–27.

# Chapter 19

## Markov Chain Monte Carlo Method

The Monte Carlo method, also known as the statistical simulation method, is a method of approximate numerical computation by random sampling from a probabilistic model. Markov Chain Monte Carlo (MCMC) is a Monte Carlo method using the Markov chain as the probabilistic model. MCMC method constructs a Markov chain so that its stationary distribution is the distribution to be sampled. It first performs a random walk based on the Markov chain to generate a sequence of samples and then uses samples of the stationary distribution to perform approximate numerical computation.

The Metropolis–Hastings algorithm is the most basic MCMC method. Metropolis et al. proposed the original algorithm in 1953, and Hastings generalized it to its present form in 1970. Gibbs sampling, a simpler and more widely used MCMC method, was proposed by S. Geman and D. Geman in 1984.

MCMC method is applied to problems such as estimation of the probability distribution, approximate computation of definite integrals, approximate solution of optimization problems, etc. Especially, it is applied to the learning and inference of probabilistic models in machine learning and is an important computational method for machine learning.

This chapter first introduces the general Monte Carlo method in Sect. 19.1, Markov chains in Sect. 19.2, then describes the general approach to Markov Chain Monte Carlo in Sect. 19.3, and finally the Metropolis–Hastings algorithm and Gibbs sampling in Sects 19.4 and 19.5, respectively.

### 19.1 Monte Carlo Method

This section describes the application of the general Monte Carlo method to random sampling, mathematical expectation estimation, and definite integral computation. Markov Chain Monte Carlo method is one of the Monte Carlo methods.

### 19.1.1 Random Sampling

The machine learning aim to make inferences about the characteristics of probability distributions based on data. The problem to be solved by the Monte Carlo method is to assume that the definition of the probability distribution is known, then obtain random samples of probability distribution through sampling, and analyze the feature of probability distribution through the obtained random samples. For example, the empirical distribution is obtained from the sample to estimate the overall distribution; or the sample mean is calculated from the sample to estimate the overall expectation. Therefore, the core of the Monte Carlo method is random sampling.

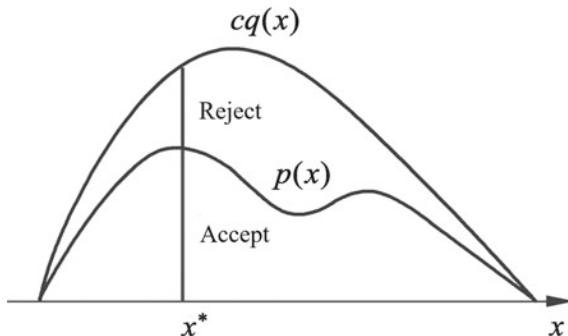
The general Monte Carlo methods are the direct sampling method, accept-reject sampling method, and importance sampling method. Accept-reject sampling method and importance sampling method are suitable for cases where the probability density function is complex (for example, the density function contains multiple variables, the variables are not independent of each other, and the density function is complicated in form) and cannot be sampled directly.

The accept-reject sampling method is introduced here. Suppose that there is a random variable  $x$ , taking the value  $x \in \mathcal{X}$ , whose probability density function is  $p(x)$ . The objective is to obtain a random sample of this probability distribution in order to perform an analysis of this probability distribution.

The basic idea of the accept-reject method is as follows. Assume that  $p(x)$  can not be directly sampled. Find a distribution that can be sampled directly, called the proposal distribution. Assume that  $q(x)$  is the probability density function of the proposal distribution, and that there are  $c$  times  $q(x)$  that must be greater than or equal to  $p(x)$ , where  $c > 0$ , as shown in Fig. 19.1 (see the color chart in Appendix earlier in the text for details). Sampling according to  $q(x)$  assumes that the result is  $x^*$ , and then randomly decides whether to accept  $x^*$  in proportion to  $\frac{p(x^*)}{cq(x^*)}$ . Intuitively, those falling within the range of  $p(x^*)$  are accepted (green) and those falling outside the range of  $p(x^*)$  are rejected (red). The accept-reject method samples the covered area (or covered volume) of  $p(x)$  in proportion to the covered area (or covered volume) of  $cq(x)$ .

The specific algorithm of the accept-reject method is as follows.

**Fig. 19.1** Accept-reject sampling method (see color chart in Appendix)



**Algorithm 19.1** (*Accept-reject method*)

Input: the probability density function  $p(x)$  of the target probability distribution of the sample;

Output: random samples of the probability distribution  $x_1, x_2, \dots, x_n$ .

Parameters: number of samples  $n$ .

- (1) Choose the probability distribution with probability density function  $q(x)$  as the proposed distribution so that it satisfies  $cq(x) \geq p(x)$  for any  $x$ , where  $c > 0$ .
- (2) The sample  $x^*$  is obtained by random sampling according to the proposed distribution  $q(x)$ , and then  $u$  is obtained by sampling according to the uniform distribution in the range  $(0, 1)$ .
- (3) If  $u \leq \frac{p(x^*)}{cq(x^*)}$ , then  $x^*$  is taken as the sampling result; otherwise, go back to step (2).
- (4) until  $n$  random samples are obtained, end.

The advantage of the accept-reject method is that it is easy to implement and the disadvantage is that it may not be efficient. If the covered volume of  $p(x)$  is a low proportion of the covered volume of  $cq(x)$ , it will result in a high proportion of rejections and inefficient sampling. Note that sampling is generally done in high-dimensional space, and even if  $p(x)$  is close to  $cq(x)$ , the difference between the two covered volumes may be large (unlike what we intuit in three-dimensional space).

### 19.1.2 Mathematical Expectation Estimate

General Monte Carlo methods, such as direct sampling, accept-reject sampling, and importance sampling, can also be used for estimation of mathematical expectation. Suppose there is a random variable  $x$ , taking values  $x \in \mathcal{X}$ , with probability density function  $p(x)$  and  $f(x)$  as a function defined on  $\mathcal{X}$ . The objective is to find the mathematical expectation  $E_{p(x)}[f(x)]$  of the function  $f(x)$  with respect to the density function  $p(x)$ .

For this problem, the Monte Carlo method draws  $n$  samples  $x_1, x_2, \dots, x_n$  independently according to the probability distribution  $p(x)$ . For example, use the above sampling method, and then calculate the sample mean  $\hat{f}_n$  of function  $f(x)$ .

$$\hat{f}_n = \frac{1}{n} \sum_{i=1}^n f(x_i) \quad (19.1)$$

as an approximation to the mathematical expectation  $E_{p(x)}[f(x)]$ .

It follows from the law of large numbers that when the sample size increases, the sample mean converges with probability 1 to the mathematical expectation that:

$$\hat{f}_n \rightarrow E_{p(x)}[f(x)], n \rightarrow \infty \quad (19.2)$$

This gives the approximate computation of the mathematical expectation:

$$E_{p(x)}[f(x)] \approx \frac{1}{n} \sum_{i=1}^n f(x_i) \quad (19.3)$$

### 19.1.3 Integral Computation

The general Monte Carlo methods can also be used for the approximate computation of definite integrals, called Monte Carlo integration. Suppose there is a function  $h(x)$  and the objective is to compute the integral of this function

$$\int_x h(x) dx$$

If one can decompose the function  $h(x)$  into the form of the product of a function  $f(x)$  and a probability density function  $p(x)$ , then there is

$$\int_x h(x) dx = \int_x f(x)p(x) dx = E_{p(x)}[f(x)] \quad (19.4)$$

So the integral of function  $h(x)$  can be expressed as the mathematical expectation of function  $f(x)$  with respect to probability density function  $p(x)$ . In fact, given a probability density function  $p(x)$ , Eq. (19.4) could be obtained when  $f(x) = \frac{h(x)}{p(x)}$ . In other words, the integral of any function can be expressed in the form of mathematical expectation of a function. The mathematical expectation of the function can be estimated by the sample mean of the function. Therefore, we can use the sample mean to approximate the integral. This is the basic idea of Monte Carlo integration.

$$\int_x h(x) dx = E_{p(x)}[f(x)] \approx \frac{1}{n} \sum_{i=1}^n f(x_i) \quad (19.5)$$

**Example 19.1**<sup>1</sup> Find the solution for  $\int_0^1 e^{-x^2/2} dx$  using Monte Carlo integration.

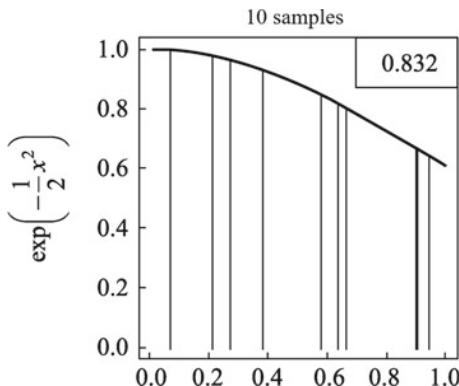
**Solution** Let  $f(x) = e^{-x^2/2}$

$$p(x) = 1 (0 < x < 1)$$

---

<sup>1</sup> Examples 19.1 and 19.2 come from Jarad Niemi.

**Fig. 19.2** Example of Monte Carlo integration



That is to say, suppose the random variable  $x$  follows uniform distribution in the  $(0, 1)$  interval.

Using the Monte Carlo integration method, as shown in Fig. 19.2, 10 random samples  $x_1, x_2, \dots, x_{10}$  are selected according to the uniform distribution in the  $(0, 1)$  interval. Calculate the function mean  $\hat{f}_{10}$  of the sample.

$$\hat{f}_{10} = \frac{1}{10} \sum_{i=1}^{10} e^{-x_i^2/2} = 0.832$$

Which is the approximation of integration. The larger the number of random samples, the more accurate the calculation.

**Example 19.2** Find the solution for  $\int_{-\infty}^{\infty} x \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-x^2}{2}\right) dx$  using Monte Carlo integration.

**Solution** Let  $f(x) = x$

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-x^2}{2}\right)$$

$p(x)$  is the density function of the standard normal distribution.

Monte Carlo integration method is used to sample  $x_1, x_2, \dots, x_n$  in the interval  $(-\infty, \infty)$  according to the standard normal distribution, take the average value to get the required integral value. When the sample increases, the integral value tends to 0.

The Markov Chain Monte Carlo method introduced in this chapter is also suitable for the case that the probability density function is complex and cannot be sampled directly. It aims to solve the problem that the sampling efficiency of general Monte Carlo methods, such as acceptance rejection sampling method and importance sampling method, is not high. The sampling samples in the general Monte

Carlo method are independent, while the sampling samples in the MCMC method are not independent, and the sample sequence forms a Markov chain.

## 19.2 Markov Chain

This section first gives the definition of Markov chain, and then introduces some properties of Markov chain. MCMC method uses these properties.

### 19.2.1 Basic Definition

**Definition 19.1** (*Markov chain*) Consider a sequence  $X = \{X_0, X_1, \dots, X_t, \dots\}$  of a random variable, where  $X_t$  represents the random variable at time  $t$ ,  $t = 0, 1, 2, \dots$ . The value collection of every random variable  $X_t$  ( $t = 0, 1, 2, \dots$ ), which is also called state space and denoted as  $S$ . Random variables can be discrete or continuous. The sequence of the above random variables constitutes a stochastic process.

Suppose that the random variable  $X_0$  at time 0 follows the probability distribution  $P(X_0) = \pi_0$ , which is called initial state distribution. There is a conditional distribution  $P(X_t|X_{t-1})$  between the random variable  $X_t$  at a certain time  $t \geq 1$  and the random variable  $X_{t-1}$  at the previous time. If  $X_t$  only depends on  $X_{t-1}$  and does not depend on the past random variables  $\{X_0, X_1, \dots, X_{t-2}\}$ , this property is called Markov property, i.e.,

$$P(X_t|X_0, X_1, \dots, X_{t-1}) = P(X_t|X_{t-1}), \quad t = 1, 2, \dots \quad (19.6)$$

The random sequence  $X = \{X_0, X_1, \dots, X_t, \dots\}$  with Markov property is called Markov chain or Markov process. The conditional probability distribution  $P(X_t|X_{t-1})$  is called the transition probability distribution of Markov chain. The transition probability distribution determines the properties of Markov chain.

The intuitive explanation of Markov nature is that “the future only depends on the present (assuming that the present is known), and has nothing to do with the past”. This assumption is reasonable in many applications.

If the transition probability distribution  $P(X_t|X_{t-1})$  is independent of  $t$ , i.e.,

$$P(X_{t+s}|X_{t-1+s}) = P(X_t|X_{t-1}), \quad t = 1, 2, \dots; s = 1, 2, \dots \quad (19.7)$$

The Markov chain is called time homogeneous Markov chain. The Markov chains mentioned in this book are time homogeneous.

The above definition is a first-order Markov chain, which can be extended to  $n$ -order Markov chain, satisfying  $n$ -order Markov property

$$P(X_t | X_0 X_1 \dots X_{t-2} X_{t-1}) = P(X_t | X_{t-n} \dots X_{t-2} X_{t-1}) \quad (19.8)$$

This book mainly considers the first order Markov chain. It is easy to verify that the  $n$ -order Markov chain can be transformed into the first-order Markov chain.

## 19.2.2 Discrete-Time Markov Chain

### 19.2.2.1 Transition Probability Matrix and State Distribution

Discrete-time Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$ , random variable  $X_t (t = 0, 1, 2, \dots)$  is defined in discrete space  $S$ , and the transition probability distribution can be expressed by matrix.

If the Markov chain is in state  $j$  at time  $(t - 1)$  and moves to state  $i$  at time  $t$ , the transition probability is recorded as

$$p_{ij} = (X_t = i | X_{t-1} = j), \quad i = 1, 2, \dots; \quad j = 1, 2, \dots \quad (19.9)$$

Satisfying

$$p_{ij} \geq 0, \quad \sum_i p_{ij} = 1$$

The transfer probability  $p_{ij}$  of Markov chain can be expressed by matrix, i.e.,

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots \\ p_{21} & p_{22} & p_{23} & \cdots \\ p_{31} & p_{32} & p_{33} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \quad (19.10)$$

The transition probability matrix  $P$  satisfies the condition  $p_{ij} \geq 0, \sum_i p_{ij} = 1$ . The matrix satisfying these two conditions is called stochastic matrix. Note that the sum of the elements of the matrix column is 1.

Consider the probability distribution of the Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  at moment  $t (t = 0, 1, 2, \dots)$ , called the state distribution at moment  $t$ , denoted as

$$\pi(t) = \begin{bmatrix} \pi_1(t) \\ \pi_2(t) \\ \vdots \end{bmatrix} \quad (19.11)$$

where  $\pi_i(t)$  represents the probability  $P(X_t = i)$  of state  $i$  at time  $t$ ,

$$\pi_i(t) = P(X_t = i), \quad i = 1, 2, \dots$$

In particular, the initial state distribution of a Markov chain can be expressed as

$$\pi(t) = \begin{bmatrix} \pi_1(0) \\ \pi_2(0) \\ \vdots \end{bmatrix} \quad (19.12)$$

where  $\pi_i(0)$  represents the probability  $P(X_0 = i)$  that the state at time 0 is  $i$ . In general, only one component of the vector of the initial distribution  $\pi(0)$  is 1, and all the remaining components are 0, indicating that the Markov chain starts from a specific state.

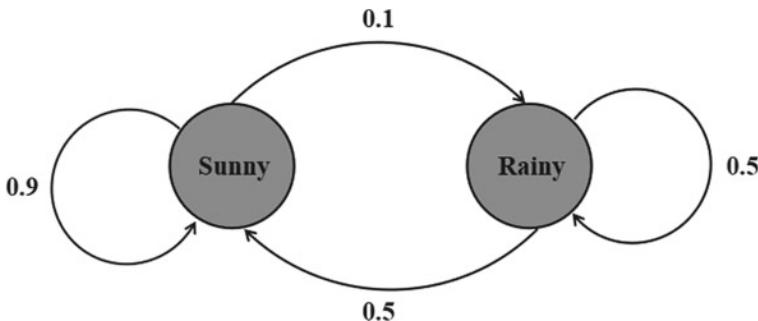
A directed graph can represent Markov chains in finite discrete states. The node represents the state, the edge represents the transition between states, and the numerical value on the edge represents the transition probability. Starting from an initial state, a sequence of states can be produced by randomly jumping (or randomly shifting) between states according to probabilities defined by directed edges. Markov chains are models that depict transitions between states over time, assuming that the future transition states depend only on the present state and have nothing to do with the past state.

The following is a simple example to give an intuitive explanation of Markov chains. Suppose we observe the weather in a certain place, and the order of the day is “Fine, Rain, Sunny, Sunny, Sunny, Rain, Sunny...”, with a certain pattern. A Markov chain can characterize the process. Suppose the change of weather has Markov property, i.e., tomorrow’s weather only depends on today’s weather, and has nothing to do with yesterday’s and the previous weather. This hypothesis is empirically sound or at least approximates reality. To be specific, for example, if it is sunny today, there is a 0.9 chance that it will be sunny tomorrow and a 0.1 chance that it will be rainy. If it’s a rainy day today, the probability of a sunny day tomorrow is 0.5, and the probability of a rainy day is also 0.5. Figure 19.3 shows this Markov chain. Based on this Markov chain, starting from an initial state and randomly shifting between states over time, the weather sequence can be generated, and the weather can be predicted.

Let’s look at an example of a Markov chain application. Language model is often used in natural language processing and speech processing, which is an  $n$ -order Markov chain based on thesaurus. For example, in English speech recognition, the speech model produces two candidates: “How to recognize speech” or “How to wreck a nice beach,”<sup>2</sup> to determine which is more likely. Obviously the former is more likely from a semantic point of view, and the language model can help to make this judgment.

---

<sup>2</sup> The English pronunciation of these two sentences is similar, but the meaning of the latter is unexplainable.



**Fig. 19.3** Example of Markov chains

Think of a statement as a sequence of words  $w_1 w_2 \dots w_s$ , and the goal is to compute its probability. The same statement is rarely repeated more than once in the corpus, so it is difficult to estimate the probability of each statement directly from the corpus. The language model can solve this problem by combining the probabilities of local word sequences to compute the probabilities of global word sequences.

Assuming that each word depends only on the word that precedes it, i.e., the sequence of words has Markov properties, then we can define a first-order Markov chain, i.e., a language model, and compute the probability of the sentence as follows

$$\begin{aligned}
 P(w_1 w_2 \dots w_s) &= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \dots P(w_i|w_1 w_2 \dots w_{i-1}) \\
 &\quad \dots P(w_s|w_1 w_2 \dots w_{s-1}) \\
 &= P(w_1)P(w_2|w_1)P(w_3|w_2) \dots P(w_i|w_{i-1}) \dots P(w_s|w_{s-1})
 \end{aligned}$$

The third equation here is based on the Markov chain hypothesis. In this Markov chain, the state space is a word list, and the generation of words in one position only depends on the words in the previous position, not on the words in the previous position. The above is a first-order Markov chain, which can be extended to  $n$ -order Markov chains in general.

Learning a language model is equivalent to determining the transfer probability value in a Markov chain. If a sufficient corpus is available, the transfer probability can be estimated directly from the corpus. Intuitively, once “wreck a nice” appears, the probability of “beach” following it is very low, so the second statement should have a smaller probability and the first statement is more likely from a language model perspective.

The state distribution of Markov chain  $X$  at time  $t$  can be determined by the state distribution at time  $(t - 1)$  and the transition probability distribution

$$\pi(t) = P\pi(t - 1) \tag{19.13}$$

This is because

$$\begin{aligned}
\pi_i(t) &= P(X_t = i) \\
&= \sum_m P(X_t = i | X_{t-1} = m) P(X_{t-1} = m) \\
&= \sum_m p_{im} \pi_m(t-1)
\end{aligned}$$

The state distribution of a Markov chain at time  $t$  can be obtained by recursion. In fact, the recursion of Eq. (19.13),

$$\pi(t) = P\pi(t-1) = P(P\pi(t-2)) = P^2\pi(t-2)$$

yields

$$\pi(t) = P^t\pi(0) \quad (19.14)$$

Here,  $P^t$  is called the  $t$ -step transition probability matrix,

$$P_{ij}^t = P(X_t = i | X_0 = j)$$

It represents the  $t$  step transition probability of starting from state  $j$  at time 0 and reaching state  $i$  at time  $t$ .  $P^t$  is also a random matrix. Equation (19.14) shows that the state distribution of a Markov chain is determined by the initial distribution and the transition probability distribution.

For the Markov chain in Fig. 19.3, the transfer matrix is

$$P = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix}$$

If the first day is sunny, the weather probability distribution (initial state distribution) is as follows:

$$\pi(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

According to this Markov chain model, the probability distribution (state distribution) of the weather for the next day, the third day and thereafter can be computed.

$$\begin{aligned}
\pi(1) &= P\pi(0) = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} \\
\pi(2) &= P^2\pi(0) = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix}^2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}
\end{aligned}$$

### 19.2.2.2 Stationary Distribution

**Definition 19.2 (stationary distribution)** Suppose Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$ , its state space is  $S$ , and the transition probability matrix is  $P = (p_{ij})$ , if there is a distribution on the state space  $S$ .

$$\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \end{bmatrix}$$

Make

$$\pi = P\pi \quad (19.15)$$

Then  $\pi$  is called a Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  is a stationary distribution.

Intuitively, if the stationary distribution of the Markov chain exists, then the stationary distribution is used as the initial distribution, and the random state transition is carried out facing the future. The state distribution at any time after that is the stationary distribution.

**Lemma 19.1** Given a Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$ , the state space is  $S$ , and the transition probability matrix is  $P = (p_{ij})$ , then the distribution  $\pi = (\pi_1, \pi_2, \dots)^T$ . The necessary and sufficient condition for  $T$  to be a stationary distribution of  $X$  is that  $\pi = (\pi_1, \pi_2, \dots)^T$  is the solution of the following equations:

$$x_i = \sum_j p_{ij}x_j, \quad i = 1, 2, \dots \quad (19.16)$$

$$x_i \geq 0, \quad i = 1, 2, \dots \quad (19.17)$$

$$\sum_i x_i = 1 \quad (19.18)$$

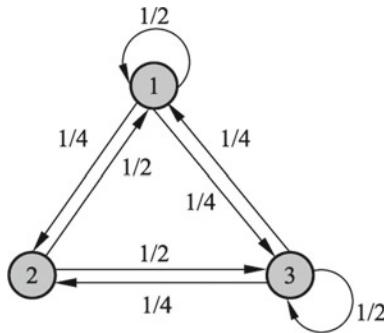
**Proof** Necessity. Assuming that  $\pi = (\pi_1, \pi_2, \dots)^T$  is a stationary distribution, which obviously satisfies Eqs. (19.17) and (19.18). Also

$$\pi_i = \sum_j p_{ij}\pi_j, \quad i = 1, 2, \dots$$

That is  $\pi = (\pi_1, \pi_2, \dots)^T$  satisfies Formula (19.16).

Adequacy. From Eqs. (19.17) and (19.18), we know that  $\pi = (\pi_1, \pi_2, \dots)^T$  is a probability distribution. Hypothesis  $\pi = (\pi_1, \pi_2, \dots)^T$  is the distribution of  $X_t$ , then

**Fig. 19.4** Markov chain example



$$P(X_t = i) = \pi_i = \sum_j p_{ij} \pi_j = \sum_j p_{ij} P(X_{t-1} = j), \quad i = 1, 2, \dots$$

$\pi = (\pi_1, \pi_2, \dots)^T$  is also  $X_{t-1}$  distribution. In fact this holds for any  $t$ . So  $\pi = (\pi_1, \pi_2, \dots)^T$  is the stationary distribution of the Markov chain.

Lemma 19.1 Gives a method to find the stationary distribution of Markov chains.

**Example 19.3** With the Markov chain shown in Fig. 19.4, its transition probability distribution is as follows.

$$P = \begin{bmatrix} 1/2 & 1/2 & 1/4 \\ 1/4 & 0 & 1/4 \\ 1/4 & 1/2 & 1/2 \end{bmatrix}$$

Find its stationary distribution.

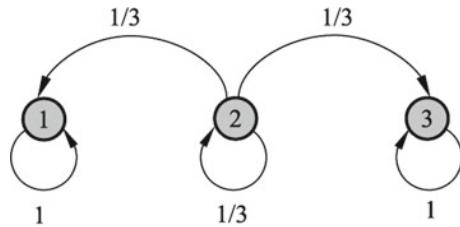
**Solution** Suppose the stationary distribution is  $\pi = (x_1, x_2, x_3)^T$ , then from Eqs. (19.16) to (19.18)

$$\begin{aligned} x_1 &= \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{4}x_3 \\ x_2 &= \frac{1}{4}x_1 + \frac{1}{4}x_3 \\ x_3 &= \frac{1}{4}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 \\ x_1 + x_2 + x_3 &= 1 \\ x_i &\geq 0, \quad i = 1, 2, 3 \end{aligned}$$

Solve the system of equations and get the unique stationary distribution

$$\pi = (2/5 \ 1/5 \ 2/5)^T$$

**Fig. 19.5** Markov chain example



**Example 19.4** With the Markov chain shown in Fig. 19.5, its transition probability distribution is as follows, and find its stationary distribution.

$$P = \begin{bmatrix} 1 & 1/3 & 0 \\ 0 & 1/3 & 0 \\ 0 & 1/3 & 1 \end{bmatrix}$$

**Solution** The stationary distribution of this Markov chain is not unique,  $\pi = (3/4 \ 0 \ 1/4)^T$ ,  $\pi = (2/3 \ 0 \ 1/3)^T$  etc. are all stationary distributions.

A Markov chain may have a unique stationary distribution, an infinite number of stationary distributions, or no stationary distributions.<sup>3</sup>

### 19.2.3 Continuous-Time Markov Chain

The continuous-time Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$ , the random variable  $X_t (t = 0, 1, 2, \dots)$  is defined in the continuous-time space  $S$ , and the transition probability distribution is defined by Probability transition kernel or transition kernel (transition kernel) said.

Let  $\mathbb{S}$  be the continuous-time space, for any  $x \in \mathbb{S}$ ,  $A \subset \mathbb{S}$ , the transition core  $P(x, A)$  is defined as

$$P(x, A) = \int_A p(x, y) dy \quad (19.19)$$

where  $p(x, \bullet)$  is the probability density function, satisfying  $p(x, \bullet) \geq 0$ ,  $P(x, \mathbb{S}) = \int_{\mathbb{S}} p(x, y) dy = 1$ . The transition core  $P(x, A)$  represents the transition probability from  $x \sim A$

$$P(X_t = A | X_{t-1} = x) = P(x, A) \quad (19.20)$$

---

<sup>3</sup> When the discrete-time Markov chain has infinite states, there may not be a stationary distribution.

Sometimes the probability density function  $p(x, \bullet)$  is also called the transition kernel.

If the probability distribution on the state space  $\mathbb{S}$  of the Markov chain  $\pi(x)$  satisfies the condition

$$\pi(y) = \int p(x, y)\pi(x)dx, \quad \forall y = \mathbb{S} \quad (19.21)$$

Then the distribution  $\pi(x)$  is called the stationary distribution of the Markov chain. Equivalently,

$$\pi(A) = \int p(x, A)\pi(x)dx, \quad \forall A = \mathbb{S} \quad (19.22)$$

Or abbreviated

$$\pi = P\pi \quad (19.23)$$

### 19.2.4 Properties of Markov Chain

The following introduces the properties of discrete-time Markov chain. It can be naturally extended to continuous-time Markov chain.

#### 19.2.4.1 Irreducible

**Definition 19.3** (*irreducible*) Let the Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  and the state space be  $\mathbb{S}$ . for any state  $i, j \in \mathbb{S}$ , if there is a time  $t(t > 0)$  satisfying

$$P(X_t = i | X_0 = j) > 0 \quad (19.24)$$

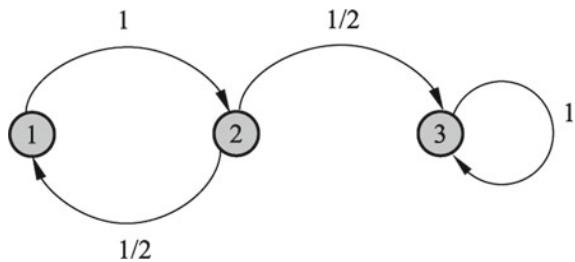
In other words, if time 0 starts from state  $j$  and the probability of time  $t$  reaching state  $i$  is greater than 0, then the Markov chain  $X$  is irreducible, otherwise it is reducible.

Intuitively, an irreducible Markov chain can start from any state and reach any state after a long time. The Markov chain in Example 19.3 is irreducible, and the Markov chain in Example 19.5 is reducible.

**Example 19.5** The Markov chain shown in Fig. 19.6 is reducible.

**Solution** With the transition probability matrix

**Fig. 19.6** Markov chain example



$$\begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 1 & 0 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix}$$

we obtain the stationary distribution  $\pi = (0 \ 0 \ 1)^T$ . After the Markov chain is transferred to state 3, it circulates and jumps on that state. It can not reach state 1 and state 2, and finally stays in state 3.

#### 19.2.4.2 Aperiodic

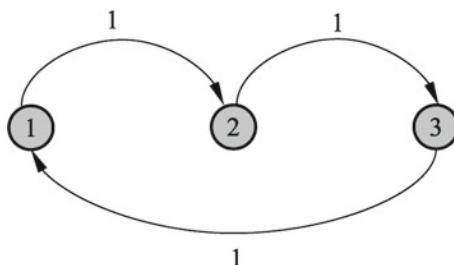
**Definition 19.4 (aperiodic)** Let the Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  and the state space be  $\mathbb{S}$ . For any state  $i \in \mathbb{S}$ , if time 0 starts from state  $i$  and the greatest common divisor of all time length  $\{t : P(X_t = i | X_0 = i) > 0\}$  of time  $t$  returning to state  $i$  is 1, then the Markov chain  $X$  is called aperiodic, Otherwise, Markov chain is called periodic.

Intuitively, a aperiodic Markov chain does not have a state. The time from this state to this state is periodic. The Markov chain in Example 19.3 is aperiodic, and the Markov chain in Example 19.6 is periodic.

**Example 19.6** The Markov chain shown in Fig. 19.7 is periodic.

**Solution** Transition probability matrix

**Fig. 19.7** Markov chain example



$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Its stationary distribution is  $\pi = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)^T$ . This Markov chain starts from each state, and the time to return to the state is a multiple of 3,  $\{3, 6, 9\}$ . It has periodicity, and the probability of finally staying in each state is  $\frac{1}{3}$ .

**Theorem 19.2** *Irreducible and aperiodic finite state Markov chains have unique stationary distribution.*

#### 19.2.4.3 Positive Recurrent

**Definition 19.5 (positive recurrent)** Let the Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  and the state space be  $\mathbb{S}$ . for any state  $i, j \in \mathbb{S}$ , the probability  $p_{ij}^t$  is defined as the probability that time 0 starts from state  $j$  and time  $t$  first transfers to state  $i$ , that is,  $p_{ij}^t = P(X_t = i, X_s \neq i, s = 1, 2, \dots, t-1 | X_0 = j)$ ,  $t = 1, 2, \dots$ . Markov chain  $X$  is said to be positive recurrent if  $\lim_{t \rightarrow \infty} p_{ij}^t > 0$  is satisfied for all States  $i$  and  $j$ .

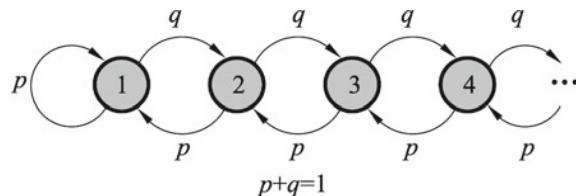
Intuitively, for a positive recurrent Markov chain, if any state starts from any other state, when the time tends to infinity, the probability of the first transition to this state is not zero. The Markov chain in Example 19.7 is positive recurrent or not positive recurrent according to different conditions.

**Example 19.7** the infinite state Markov chain shown in Fig. 19.8 is positive recurrent when  $p > q$ , and not positive recurrent when  $p \leq q$ .

**Solution** Transition probability matrix

$$\begin{bmatrix} p & p & 0 & 0 & \dots \\ q & 0 & p & 0 & \dots \\ 0 & q & 0 & p & \dots \\ 0 & 0 & q & 0 & \dots \\ \vdots & \vdots & & \ddots & \end{bmatrix}$$

**Fig. 19.8** Markov chain example



When  $p > q$ , the stationary distribution is

$$\pi_i = \left(\frac{q}{p}\right)^i \left(\frac{p-q}{p}\right), \quad i = 1, 2, \dots$$

When time tends to infinity, the probability of transferring to any state is not zero, and the Markov chain is positive recurrent.

When  $p \leq q$ , there is no stationary distribution and the Markov chain is not positive recurrent.

**Theorem 19.3** *An irreducible, non-periodic and normally returning Markov chain has a unique stationary distribution.*

#### 19.2.4.4 Ergodic Theorem

The following describes the ergodic theorem of Markov chain.

**Theorem 19.4** (ergodic theorem) *Let the Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  and the state space be  $\mathbb{S}$ . If the Markov chain  $X$  is irreducible, aperiodic and positive recurrent, then the Markov chain has a unique stationary distribution  $\pi = (\pi_1, \pi_2, \dots)^T$ . And the limit distribution of transition probability is the stationary distribution of Markov chain*

$$\lim_{t \rightarrow \infty} P(X_t = i | X_0 = j) = \pi_i, \quad i = 1, 2, \dots; \quad j = 1, 2, \dots \quad (19.25)$$

If  $f(X)$  is a function defined in the state space,  $E_\pi[|f(X)|] < \infty$ , then

$$P\left\{\hat{f}_t \rightarrow E_\pi[f(X)]\right\} = 1 \quad (19.26)$$

where,

$$\hat{f}_t = \frac{1}{t} \sum_{s=1}^t f(x_s)$$

$E_\pi[f(X)] = \sum_i f(i)\pi_i$  is the mathematical expectation of  $f(X)$  with respect to a stationary distribution  $\pi = (\pi_1, \pi_2, \dots)^T$ , expressed by Eq. (19.26)

$$\hat{f}_t \rightarrow E_\pi[f(X)], \quad t \rightarrow \infty \quad (19.27)$$

Holds almost everywhere or holds with probability 1.

An intuitive explanation of the ergodic theorem: For a Markov chain that satisfies the corresponding conditions, when the time tends to infinity, the state distribution

of the Markov chain approaches a stationary distribution, and the sample mean of the function of the random variable converges to mathematical expectation of the function with probability 1. The sample mean can be thought of as the time mean, while the mathematical expectation is the space mean. The ergodic theorem actually expresses the meaning of ergodicity: when time tends to infinity, the time mean is equal to the space mean. The three conditions of the ergodic theorem: irreducible, non-periodic, and positive recurrent, ensure that the probability of reaching any state when time tends to infinity is not zero.

In theory, it is not known how many iterations can the state distribution of a Markov chain be close to a stationary distribution. In the actual application of the ergodic theorem, a sufficiently large integer  $m$  is selected. After  $m$  iterations, the state distribution is considered to be a stationary distribution. At this time, calculate the mean value from the  $m + 1$  iteration to the  $n$  iteration, that is

$$\hat{E}f = \frac{1}{n-m} \sum_{i=m+1}^n f(x_i) \quad (19.28)$$

is called the ergodic mean.

#### 19.2.4.5 Reversible Markov Chain

**Definition 19.6 (Reversible Markov Chain)** Set Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$ , the state space is  $\mathbb{S}$ , the transition probability matrix is  $P$ , if there is a state distribution  $\pi = (\pi_1, \pi_2, \dots)^T$ , for any state  $i, j \in \mathbb{S}$ , for any time  $t$  satisfies

$$P(X_t = i | X_{t-1} = j)\pi_j = P(X_{t-1} = j | X_t = i)\pi_i, \quad i, j = 1, 2, \dots \quad (19.29)$$

or abbreviated as

$$p_{ij}\pi_j = p_{ji}\pi_i, \quad i, j = 1, 2, \dots \quad (19.30)$$

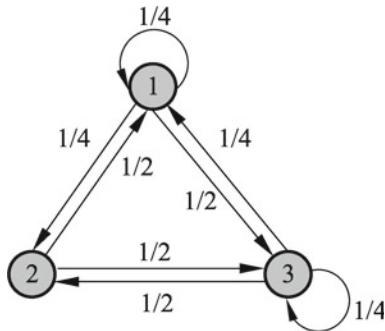
Then this Markov chain  $X$  is called a reversible Markov chain, and Formula (19.30) is called the detailed balance equation.

Intuitively, if there is a reversible Markov chain, the stationary distribution of the Markov chain is used as the initial distribution to perform random state transition. Whether it is facing the future or facing the past, the state distribution at any moment is the stationary distribution. The Markov chain in Example 19.3 is reversible, and the Markov chain in Example 19.8 is irreversible.

**Example 19.8** The Markov chain shown in Fig. 19.9 is irreversible.

**Solution** Transition probability matrix

**Fig. 19.9** Markov chain example



$$\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/4 & 0 & 1/2 \\ 1/2 & 1/2 & 1/4 \end{bmatrix}$$

stationary distribution  $\pi = (8/25 \ 7/25 \ 2/5)^T$ . Does not satisfy the detailed stationary equation.

**Theorem 19.5** (detailed balance equation) *The state distribution  $\pi$  that satisfies the detailed balance equation is the stationary distribution of the Markov chain, which is*

$$P\pi = \pi$$

**Proof** In fact

$$(P\pi)_i = \sum_j p_{ij}\pi_j = \sum_j p_{ji}\pi_i = \pi_i \sum_j p_{ji} = \pi_i, \quad i = 1, 2, \dots \quad (19.31)$$

Theorem 19.5 states that a reversible Markov chain must have a unique stationary distribution, and a sufficient condition (not a necessary condition) for a Markov chain to have a stationary distribution is given. In other words, the reversible Markov chain satisfies the condition of ergodic theorem 19.4

## 19.3 Markov Chain Monte Carlo Method

### 19.3.1 Basic Ideas

Suppose the goal is to randomly sample a probability distribution, or to find the mathematical expectation of a function about the probability distribution. Traditional Monte Carlo methods can be used, such as accept-reject methods, importance

sampling methods, and Markov Chain Monte Carlo methods. The MCMC method is more suitable for the situations where the random variable is multivariate, the density function is non-standard, and the components of the random variable are not independent.

Assuming that a multivariate random variable  $x$  satisfies  $x \in \mathcal{X}$ , its probability density function is  $p(x)$ ,  $f(x)$  is a function defined on  $x \in \mathcal{X}$ , and the goal is to obtain a sample set of probability distribution  $p(x)$ , and find the mathematical expectation of the function  $E_{p(x)}[f(x)]$ .

The MCMC method is used to solve this problem. The basic idea is: define a Markov chain  $X = \{X_0, X_1, \dots, X_t, \dots\}$  which satisfies the ergodic theorem on the state space  $\mathcal{S}$  of the random variable  $x$ , so that its stationary distribution is the sampling target distribution  $p(x)$ . Then perform a random walk on this Markov chain, and get a sample at each moment. According to the ergodic theorem, when time goes to infinity, the distribution of the sample approaches a stationary distribution, and the mean value of the sample's function approaches the mathematical expectation of the function. Therefore, when the time is long enough (the time is greater than a certain positive integer  $m$ ), the sample set  $\{x_{m+1}, x_{m+2}, \dots, x_n\}$  obtained by random walk in the following time (the time is less than or equal to a certain positive integer  $n$ ,  $n > m$ ) is the sampling result of the target probability distribution, and the obtained function mean (ergodic mean) is the mathematical expectation value to be computed:

$$\hat{E} f = \frac{1}{n-m} \sum_{i=m+1}^n f(x_i) \quad (19.32)$$

the time period up to time  $m$  is called the combustion period.

How to construct a specific Markov chain becomes the key to this method. For continuous variables, we need to define the transition kernel function; for discrete variables, we need to define the transition matrix. One method is to define a special transition kernel function or transition matrix and construct a reversible Markov chain, which can guarantee the validity of the ergodic theorem. Commonly used MCMC methods include Metropolis–Hastings algorithm and Gibbs sampling.

Since this Markov chain satisfies the ergodic theorem, the starting point of the random walk does not affect the results obtained, that is, starting from different starting points, they will all converge to the same stationary distribution.

The judgment of the convergence of the MCMC method is usually empirical. For example, a random walk is performed on the Markov chain to check whether the ergodic mean has converged. Specifically, samples are taken at regular intervals, and after multiple samples are obtained, the traversal average is computed. When the computed average is stable, it is considered that the Markov chain has converged. For another example, perform multiple random walks in parallel on the Markov chain, and compare whether the ergodic mean of each random walk is close to the same.

In the sample sequence obtained in the MCMC method, adjacent sample points are related, rather than independent. Therefore, when an independent sample is needed, random sampling can be performed again in the sample sequence, for example, a

sample is taken at regular intervals, and the sub-sample set obtained in this way is regarded as an independent sample set.

The MCMC method is easier to implement than the accept-reject method, because only the Markov chain needs to be defined, not the recommendation distribution. Generally speaking, the MCMC method is more efficient than the accept-reject method. There is not a large number of rejected samples, although the samples in the burning period are also discarded.

### 19.3.2 Basic Steps

According to the above discussion, the Markov Chain Monte Carlo method can be summarized as the following three steps:

- (1) First, construct a Markov chain that satisfies the ergodic theorem on the state space  $\mathbb{S}$  of the random variable  $x$ , and make its stationary distribution as the target distribution  $p(x)$ ;
- (2) Starting from a certain point  $x_0$  in the state space, use the constructed Markov chain to perform a random walk to generate the sample sequence  $x_0, x_1, \dots, x_l, \dots$
- (3) Apply the ergodic theorem of Markov chain, determine the positive integers  $m$  and  $n$ , ( $m < n$ ), get the sample set  $\{x_{m+1}, x_{m+2}, \dots, x_n\}$ , find the mean value of the function  $f(x)$  (ergodic mean)

$$\hat{E}f = \frac{1}{n-m} \sum_{i=m+1}^n f(x_i) \quad (19.33)$$

It is the computation formula of the MCMC method.

Here are several important questions:

- (1) How to define a Markov chain to ensure that the conditions of the MCMC method are established.
- (2) How to determine the number of convergence steps  $m$  to ensure the unbiasedness of sample sampling.
- (3) How to determine the number of iteration steps  $n$  to ensure the accuracy of the ergodic mean calculation.

### 19.3.3 Markov Chain Monte Carlo Method and Machine Learning

Markov Chain Monte Carlo method plays an important role in machine learning, especially Bayesian learning. The main reason is that the MCMC method can be used in the learning and reasoning of probabilistic models.

Assuming that the observed data is represented by a random variable  $y \in \mathcal{Y}$  and the model is represented by a random variable  $x \in \mathcal{X}$ , Bayesian learning computes the posterior probability of the model under the given data condition through Bayes' theorem, and selects the model with the largest posterior probability.

Posterior probability is

$$p(x|y) = \frac{p(x)p(y|xy)}{\int_X p(y|x')p(x')dx'} \quad (19.34)$$

Three integral operations are often required in Bayesian learning: normalization, marginalization, and mathematical expectation.

Normalization calculation is required in the computation of posterior probability:

$$\int_X p(y|x')p(x')dx' \quad (19.35)$$

If there is a hidden variable  $z \in \mathcal{Z}$ , the computation of the posterior probability requires marginalization calculation:

$$p(x|y) = \int_{\mathcal{Z}} p(x, z|y)dz \quad (19.36)$$

If there is a function  $f(x)$ , the mathematical expectation of the posterior probability distribution of the function can be calculated:

$$E_{P(x|y)}[f(x)] = \int_{\mathcal{X}} f(x)p(x|y)dx \quad (19.37)$$

When the observation data and the model are very complicated, the above integral computation becomes difficult. The Markov Chain Monte Carlo method provides a general and effective solution for these computations.

## 19.4 Metropolis–Hasting Algorithm

This section describes the Metropolis–Hastings algorithm, which is a representative algorithm of Markov Chain Monte Carlo method.

## 19.4.1 Fundamental Concepts

### 19.4.1.1 Markov Chain

Suppose the probability distribution to be sampled is  $p(x)$ . The Metropolis–Hastings algorithm uses a Markov chain with a transition core  $p(x, x')$ :

$$p(x, x') = q(x, x')\alpha(x, x') \quad (19.38)$$

where  $q(x, x')$  and  $\alpha(x, x')$  are called proposal distribution and acceptance distribution, respectively.

Proposal distribution  $q(x, x')$  is the transition core of another Markov chain, and  $q(x, x')$  is irreducible, that is, its probability value is always non-zero, and it is a distribution that is easy to sample. Acceptance distribution  $\alpha(x, x')$  is

$$\alpha(x, x') = \min \left\{ 1, \frac{p(x')q(x', x)}{p(x)q(x, x')} \right\} \quad (19.39)$$

At this time, the transfer core  $p(x, x')$  can be written as

$$p(x, x') = \begin{cases} q(x, x'), & p(x')q(x', x) \geq p(x)q(x, x') \\ q(x', x) \frac{p(x')}{p(x)}, & p(x')q(x', x) < p(x)q(x, x') \end{cases} \quad (19.40)$$

The random walk on the Markov chain with the transfer core  $p(x, x')$  is performed in the following way. If at time  $(t - 1)$  it is in state  $x$ , i.e.,  $x_{t-1} = x$ , first, according to the suggested distribution  $q(x, x')$  generate a candidate state  $x'$ , and then according to the acceptance distribution  $\alpha(x, x')$  determine whether to accept the state  $x'$ . Accept  $x'$  with probability  $\alpha(x, x')$ , decide to transition to state  $x'$  at time  $t$ , and with probability  $1 - \alpha(x, x')$  reject  $x'$  and decide to stay in state  $x'$  at time  $t$ . Specifically, a random number  $u$  is drawn from the uniform distribution on the interval  $(0, 1)$  to determine the state at time  $t$ .

$$x_t = \begin{cases} x', & u \leq \alpha(x, x') \\ x, & u > \alpha(x, x') \end{cases}$$

It can be proved that the Markov chain whose transition kernel is  $p(x, x')$  is a reversible Markov chain (satisfying the ergodic theorem), and its stationary distribution is  $p(x)$ , which is the target distribution to be sampled. In other words, this is a concrete realization of Markov Chain Monte Carlo method.

**Theorem 19.6** *The Markov chain formed by the transfer core (19.38)–(19.40) is reversible, i.e.,*

$$p(x)p(x, x') = p(x')p(x', x) \quad (19.41)$$

and  $p(x)$  is the stationary distribution of the Markov chain.

**Proof** If  $x \neq x'$ , then Formula (19.41) is obviously true.

Suppose  $x \neq x'$ , then

$$\begin{aligned} p(x)p(x, x') &= p(x')q(x, x') \min\left\{1, \frac{p(x')q(x', x)}{p(x)q(x, x')}\right\} \\ &= \min\{p(x)q(x, x'), p(x')q(x', x)\} \\ &= p(x')q(x', x) \min\left\{\frac{p(x)q(x', x)}{p(x)q(x', x)}, 1\right\} \\ &= p(x')p(x', x) \end{aligned}$$

Equation (19.41) holds.

From Eq. (19.41),

$$\begin{aligned} \int p(x)p(x, x')dx &= \int p(x')p(x', x)dx \\ &= p(x') \int p(x', x)dx \\ &= p(x') \end{aligned}$$

According to the definition of stationary distribution (19.21),  $p(x)$  is the stationary distribution of Markov chain.

#### 19.4.1.2 Recommended Distribution

It is suggested that the distribution  $q(x, x')$  has many possible forms, here introduce two common forms.

The first form assumes that the proposed distribution is symmetric, i.e., for any  $x$  and  $x'$

$$q(x, x') = q(x', x) \quad (19.42)$$

Such a recommended distribution is called Metropolis selection, and it is also the recommended distribution originally adopted by the Metropolis–Hastings algorithm. At this time, the acceptance distribution  $\alpha(x, x')$  is simplified to

$$\alpha(x, x') = \min\left\{1, \frac{p(x')}{p(x)}\right\} \quad (19.43)$$

A special case of Metropolis selection is that  $q(x, x')$  takes the conditional probability distribution  $p(x'|x)$ , which is defined as a multivariate normal distribution. The mean is  $x$  and the covariance matrix is a constant matrix.

Another special case of Metropolis selection is to set  $q(x, x') = q(|x - x'|)$ , then the algorithm is called the random walk Metropolis algorithm. e.g.,

$$q(x, x') \propto \exp\left(-\frac{(x' - x)^2}{2}\right)$$

The feature of Metropolis selection is that when  $x'$  is close to  $x$ , the probability value of  $q(x, x')$  is high, otherwise the probability value of  $q(x, x')$  is low. The probability of state transition is greater in nearby points.

The second form is called independent sampling. Assuming that  $q(x, x')$  has nothing to do with the current state  $x$ , i.e.,  $q(x, x') = q(x')$ . The computation of the recommended distribution is carried out according to  $q(x')$  independent sampling. At this time, the acceptance distribution  $\alpha(x, x')$  can be written as

$$\alpha(x, x') = \min\left\{1, \frac{w(x')}{w(x)}\right\} \quad (19.44)$$

where  $w(x') = p(x')/q(x')$ ,  $w(x) = p(x)/q(x)$ .

Independent sampling is simple to implement, but may be slow to converge. The distribution close to the target distribution  $p(x)$  is usually selected as the recommended distribution  $q(x)$ .

#### 19.4.1.3 Full Conditional Distribution

The target distribution of the Markov Chain Monte Carlo method is usually a multivariate joint probability distribution  $p(x) = p(x_1, x_2, \dots, x_k)$ , where  $x = (x_1, x_2, \dots, x_k)^T$  is a  $k$ -dimensional random variable. If all  $k$  variables in the conditional probability distribution  $p(x_I|x_{-I})$  appear, where  $x_I = \{x_i, i \in I\}$ ,  $x_{-I} = \{x_i, i \notin I\}$ ,  $I \subset K = \{1, 2, \dots, k\}$ , then this conditional probability distribution is called a full conditional distribution.

The fully conditional distribution has the following properties: for any  $x \in \chi$  and any  $I \subset K$ , there are

$$p(x_I|x_{-I}) = \frac{p(x)}{\int p(x)dx_I} \propto p(x) \quad (19.45)$$

Moreover, for any  $x, x' \in \chi$  and any  $I \subset K$ , we have

$$\frac{p(x'_I|x'_{-I})}{p(x_I|x_{-I})} = \frac{p(x')}{p(x)} \quad (19.46)$$

In the Metropolis–Hastings algorithm, the property (19.46) can be used to simplify the computation and improve the efficiency. Specifically, the ratio of the fully conditional distribution probability  $\frac{p(x'_I|x'_{-I})}{p(x_I|x_{-I})}$  is used to calculate the ratio of the joint probability  $\frac{p(x')}{p(x)}$ , while the former is easier to calculate.

**Example 19.9** Let the density function of the joint probability distribution of  $x_1$  and  $x_2$  be

$$p(x_1, x_2) \propto \exp\left\{-\frac{1}{2}(x_1 - 1)^2(x_2 - 1)^2\right\}$$

Find its full conditional distribution.

**Solution** By the definition of full conditional distribution

$$\begin{aligned} p(x_1|x_2) &\propto p(x_1, x_2) \\ &\propto \exp\left\{-\frac{1}{2}(x_1 - 1)^2(x_2 - 1)^2\right\} \\ &\propto N(1, (x_2 - 1)^{-2}) \end{aligned}$$

Here  $N(1, (x_2 - 1)^{-2})$  is a normal distribution, the mean value is 1, and the variance is  $(x_2 - 1)^{-2}$ . At this time,  $x_1$  is a variable and  $x_2$  is a parameter. Similarly obtain

$$\begin{aligned} p(x_2|x_1) &\propto p(x_1, x_2) \\ &\propto \exp\left\{-\frac{1}{2}(x_2 - 1)^2(x_1 - 1)^2\right\} \\ &\propto N(1, (x_1 - 1)^{-2}) \end{aligned}$$

## 19.4.2 Metropolis–Hastings Algorithm

### Algorithm 19.2 (Metropolis–Hastings algorithm)

Input: the density function  $p(x)$  and the function  $f(x)$  of the sampling target distribution;

Output: random samples  $x_{m+1}, x_{m+2}, \dots, x_n$  of  $p(x)$ , function sample mean  $f_{mn}$ ;  
Parameters: the number of convergence steps  $m$ , the number of iteration steps  $n$ .

- (1) Choose an initial value  $x_0$  arbitrarily.
- (2) Execute cyclically for  $i = 1, 2, \dots, n$ 
  - (a) Set state  $x_{i-1} = x$ , and randomly select a candidate state  $x'$  according to the suggested distribution  $q(x, x')$ .

(b) Calculate the probability of acceptance

$$\alpha(x, x') = \min \left\{ 1, \frac{p(x')q(x', x)}{p(x)q(x, x')} \right\}$$

(c) Randomly draw a number  $u$  from the interval  $(0, 1)$  according to a uniform distribution.

If  $u \leq \alpha(x, x')$ , the state  $x_i = x'$ ; otherwise, the state  $x_i = x$ .

(3) Get the sample set  $\{x_{m+1}, x_{m+2}, \dots, x_n\}$ .

$$f_{mn} = \frac{1}{n-m} \sum_{i=m+1}^n f(x_i)$$

### 19.4.3 The Single-Component Metropolis–Hastings Algorithm

In the Metropolis–Hastings algorithm, it is usually necessary to sample the distribution of multivariate variables. Sometimes it is difficult to sample the distribution of multivariate variables. The conditional distribution of each variable of the multivariate can be sampled separately in order to realize a sampling of the entire multivariate. This is the single-component Metropolis–Hastings algorithm.

Assume that the state of the Markov chain is represented by  $k$ -dimensional random variables

$$x = (x_1, x_2, \dots, x_k)^T$$

where  $x_j$  represents the  $j$ -th component of the random variable  $x$ ,  $j = 1, 2, \dots, k$ , and  $x^{(i)}$  represents the state of the Markov chain at time  $i$

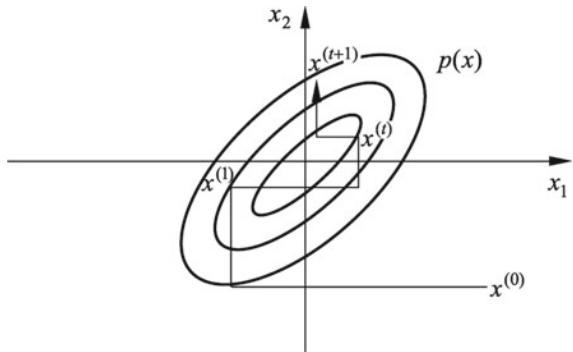
$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)})^T, \quad i = 1, 2, \dots, n$$

where  $x_j^{(i)}$  is the  $j$ -th component of the random variable  $x^{(i)}$ ,  $j = 1, 2, \dots, k$ .

In order to generate a sample set  $\{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$  with a capacity of  $n$ , the single-component Metropolis–Hastings algorithm implements one iteration of the Metropolis–Hastings algorithm by the following  $k$  iterations.

At the end of the  $(i-1)$ -th iteration, the value of component  $x_j$  is  $x_j^{(i-1)}$ . In the  $j$ -th step of the  $i$ -th iteration, the component  $x_j$  is updated according to the metropolis Hastings algorithm to obtain its new value  $x_j^{(i)}$ . First, the candidate value  $x_j'^{(i)}$  of component is  $x_j$  generated by sampling from the proposed distribution  $q(x_j'^{(i)}, x_j | x_{-j}^{(i)})$ , where  $x_{-j}^{(i)}$  represents all the values of  $F$  except for  $x_j^{(i-1)}$  after the

**Fig. 19.10** An example of single component Metropolis–Hastings algorithm



( $j - 1$ )-th step of the  $i$ -th iteration, namely

$$x_{-j}^{(i)} = (x_1^{(i)}, \dots, x_{j-1}^{(i)}, x_{j+1}^{(i-1)}, \dots, x_k^{(i-1)})^T$$

where component  $1, 2, \dots, j - 1$  has been updated. Then, according to the acceptance probability

$$\alpha(x_j^{(i-1)}, x_j^{(i)} | x_{-j}^{(i)}) = \min \left\{ 1, \frac{p(x_j^{(i)} | x_{-j}^{(i)}) q(x_j^{(i)}, x_j^{(i-1)} | x_{-j}^{(i)})}{p(x_j^{(i-1)} | x_{-j}^{(i)}) q(x_j^{(i-1)}, x_j^{(i)} | x_{-j}^{(i)})} \right\} \quad (19.47)$$

Sampling determines whether to accept the candidate value  $x_j'^{(i)}$ . If  $x_j'^{(i)}$  is accepted, let  $x_j^{(i)} = x_j'^{(i)}$ , otherwise let  $x_j^{(i)} = x_j'^{m(i-1)}$ . The other components do not change in step  $j$ . The transition probability of Markov chain is

$$p(x_j^{(i-1)}, x_j'^{(i)} | x_{-j}^{(i)}) = \alpha(x_j^{(i-1)}, x_j'^{(i)} | x_{-j}^{(i)}) q(x_j^{(i-1)}, x_j^{(i)} | x_{-j}^{(i)}) \quad (19.48)$$

Figure 19.10 shows the iterative process of the single component metropolis Hastings algorithm. The goal is to sample a random variable  $x$  with two variables. If the variable  $x_1$  or  $x_2$  is updated, a movement is generated in the horizontal or vertical direction, and a new sample point is generated by continuous horizontal and vertical movement. Note that since the proposed distribution may not be accepted, Metropolis–Hastings algorithm may not move at some adjacent times.

## 19.5 Gibbs Sampling

This section describes Gibbs sampling, a common algorithm of Markov Chain Monte Carlo method. It can be considered as a special case of Metropolis–Hastings algorithm, but it is easier to implement, so it is widely used.

### 19.5.1 Basic Principles

Gibbs sampling is used to sample and estimate the joint distribution of multivariate variables.<sup>4</sup> The basic method is to define the full conditional probability distribution from the joint probability distribution, and sample the full conditional probability distribution in turn to get the sample sequence. It can be proved that such a sampling process is a random walk on a Markov chain, each sample corresponds to the state of the Markov chain, and the stationary distribution is the joint distribution of the target. As a whole, it becomes a MCMC method, and the samples after the combustion period are random samples of joint distribution.

Suppose the joint probability distribution of multivariate variables is  $p(x) = p(x_1, x_2, \dots, x_k)$ . Gibbs sampling from a beginning  $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_k^{(0)})^T$ . Starting from the initial sample  $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_k^{(0)})^T$ , the joint distribution is obtained by iteration. A sample of  $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)})^T$ . Finally, the sample sequence  $\{x^{(0)}, x^{(1)}, \dots, x^{(n)}\}$  is obtained.

In each iteration, one of the  $k$  random variables is randomly sampled. If the  $j$ -th variable is randomly sampled in the  $i$ -th iteration, then the distribution of sampling is the full condition probability distribution  $p(x_j|x_{-j}^{(i)})$ , where  $x_{-j}^{(i)}$  represents the variables other than  $j$  in the  $i$ -th iteration.

In step  $(i - 1)$ , the sample  $(x_1^{(i-1)}, x_2^{(i-1)}, \dots, x_k^{(i-1)})^T$  is obtained. In step  $i$ , the first variable is randomly sampled according to the following full conditional probability distribution

$$p(x_1|x_2^{(i-1)}, \dots, x_k^{(i-1)})^T$$

Then the  $j$ -th variable is randomly sampled according to the following full condition probability distribution

$$p(x_j|x_1^{(i)}, \dots, x_{j-1}^{(i)}, x_{j+1}^{(i-1)}, \dots, x_k^{(i-1)}), \quad j = 2, \dots, k - 1$$

Finally, the  $k$ -th variable is randomly sampled according to the following full condition probability distribution

$$p(x_k|x_1^{(i)}, \dots, x_{k-1}^{(i)})$$

We get  $x_k^{(i)}$ , then we get the whole sample  $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)})^T$ .

Gibbs sampling is a special case of one component metropolis Hastings algorithm. The proposed distribution is defined as the full conditional probability distribution of the current variable  $x_j$ ,  $j = 1, 2, \dots, k$

$$q(x, x') = p(x'_j|x_{-j}) \tag{19.49}$$

---

<sup>4</sup> Gibbs sampling is named after the founder of statistical mechanics (Josiah Willard Gibbs), and the algorithm is analogous to statistical mechanics.

In this case, the acceptance probability  $\alpha = 1$

$$\begin{aligned}\alpha(x, x') &= \min\left\{1, \frac{p(x')q(x', x)}{p(x)q(x, x')}\right\} \\ &= \min\left\{1, \frac{p(x'_{-j})p(x'_j|x'_{-j})p(x_j|x'_{-j})}{p(x_{-j})p(x_j|x_{-j})p(x'_j|x_{-j})}\right\} = 1\end{aligned}\quad (19.50)$$

We use  $p(x_{-j}) = p(x'_{-j})$  and  $p(\bullet|x_{-j}) = p(\bullet|x'_{-j})$  here.

The transition kernel is the full conditional probability distribution

$$p(x, x') = p(x'_j|x_{-j}) \quad (19.51)$$

In other words, random sampling according to the full conditional probability distribution  $p(x'_j|x_{-j})$  of a single variable can realize the single component Metropolis–Hastings algorithm. Gibbs sampling accepts the result of each sampling, but does not reject it, which is different from the general metropolis Hastings algorithm.

Here, we assume that  $p(x'_j|x_{-j})$  is not zero, that is, Markov chain is irreducible.

## 19.5.2 Gibbs Sampling Algorithm

### Algorithm 19.3 (Gibbs sampling)

Input: density function  $p(x)$  of target probability distribution, function  $f(x)$ ;

Output: random sample  $x_{m+1}, x_{m+2}, \dots, x_n$  of  $p(x)$ , function sample mean  $f_{mn}$ ;

Parameters: convergence steps  $m$ , iteration steps  $n$ .

(1) Initialization. Give the initial sample  $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_k^{(0)})^T$ .

(2) Execute in a loop on  $i$ .

Let the sample at the end of the  $(i - 1)$  iteration be  $x^{(i-1)} = (x_1^{(i-1)}, x_2^{(i-1)}, \dots, x_k^{(i-1)})^T$ , then the  $i$ -th iteration is performed as follows:

$$\left\{ \begin{array}{l} (1) \text{ Extracting } x_1^{(i)} \text{ from full conditional distribution} \\ \quad p(x_1|x_2^{(i-1)}, \dots, x_k^{(i-1)}) \\ \vdots \\ (j) \text{ Extracting } x_j^{(i)} \text{ from full conditional distribution} \\ \quad p(x_j|x_1^{(i)}, \dots, x_{j-1}^{(i)}, x_{j+1}^{(i-1)}, \dots, x_k^{(i-1)}) \\ \vdots \\ (k) \text{ Extracting } x_k^{(i)} \text{ from full conditional distribution } p(x_k|x_1^{(i)}, \dots, x_{k-1}^{(i)}) \end{array} \right.$$

The  $i$ -th iteration value is obtained  $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)})^T$ .

(3) Get the sample set

$$\{x^{(m+1)}, x^{(m+2)}, \dots, x^{(n)}\}$$

(4) Calculate

$$f_{mn} = \frac{1}{n-m} \sum_{i=m+1}^n f(x^{(i)})$$

**Example 19.10** Use Gibbs sampling to take random samples from the following bivariate normal distribution.

$$x = (x_1, x_2)^T \sim p(x_1, x_2)$$

$$p(x_1, x_2) = N(0, \Sigma), \quad \Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

**Solution** The conditional probability distribution is a single normal distribution

$$\begin{aligned} p(x_1|x_2) &= N(\rho x_2, (1-\rho^2)) \\ p(x_2|x_1) &= N(\rho x_1, (1-\rho^2)) \end{aligned}$$

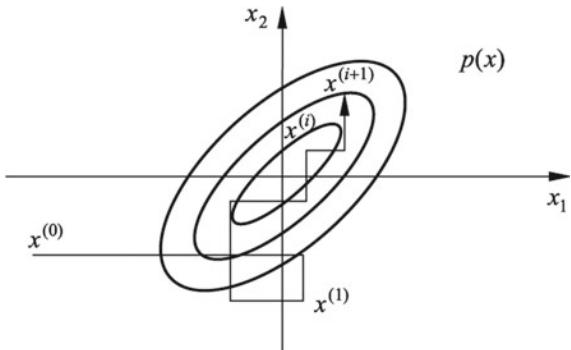
Assuming that the initial sample is  $x^{(0)} = (x_1^{(0)}, x_2^{(0)})$ , through Gibbs sampling, the following sample sequence can be obtained:

Iteration times	Sample $x_1$	Sample $x_2$	Generate samples
1	$x_1 \sim N(\rho x_2^{(0)}, (1-\rho^2))$ , obtain $x_1^{(1)}$	$x_2 \sim N(\rho x_1^{(1)}, (1-\rho^2))$ , obtain $x_2^{(1)}$	$x^{(1)} = (x_1^{(1)}, x_2^{(1)})^T$
	$\vdots$	$\vdots$	$\vdots$
$i$	$x_1 \sim N(\rho x_2^{(i-1)}, (1-\rho^2))$ , obtain $x_1^{(i)}$	$x_2 \sim N(\rho x_1^{(i)}, (1-\rho^2))$ , obtain $x_2^{(i)}$	$x^{(i)} = (x_1^{(i)}, x_2^{(i)})^T$
	$\vdots$	$\vdots$	$\vdots$

The sample set  $\{x^{(m+1)}, x^{(m+2)}, \dots, x^{(n)}\}$ ,  $m < n$  obtained is a random sampling of the bivariate normal distribution. Figure 19.11 shows the process of Gibbs sampling.

The difference between the single-component Metropolis–Hastings algorithm and Gibbs sampling is that in the former algorithm, the sample moves between sample

**Fig. 19.11** Gibbs sampling example



points, but it may stay at some sample points (because the sample is rejected); In the latter algorithm, the sample will continue to move between sample points.

Gibbs sampling is suitable for situations where the full conditional probability distribution is easy to sample, while the single-component Metropolis–Hastings algorithm is suitable for situations where the full conditional probability distribution is not easy to sample. In this case, the easy-to-sample conditional distribution is used as the recommended distribution.

### 19.5.3 Sampling Computation

In Gibbs sampling, the full conditional probability distribution needs to be sampled multiple times. The nature of probability distribution can be used to improve sampling efficiency. Let's take Bayesian learning as an example to introduce this technique.

Let  $y$  denote observed data,  $\alpha, \theta, z$  denote hyperparameters, model parameters, and unobserved data, respectively, and  $x = (\alpha, \theta, z)$ , as shown in Fig. 19.12. The purpose of Bayesian learning is to estimate the posterior probability distribution  $p(x|y)$  and find the model with the largest posterior probability.

$$p(x|y) = p(\alpha, \theta, z|y) \propto p(z, y|\theta) p(\theta|\alpha) p(\alpha) \quad (19.52)$$

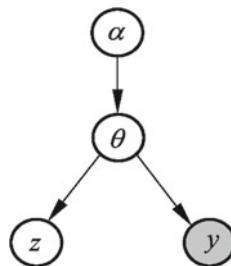
where  $p(\alpha)$  is the hyperparameter distribution,  $p(\theta|\alpha)$  is the prior distribution, and  $p(z, y|\theta)$  is the distribution of complete data.

Now use Gibbs sampling to estimate  $p(x|y)$ , where  $y$  is known and  $x = (\alpha, \theta, z)$  is unknown. The fully conditional distribution of each variable  $\alpha, \theta, z$  in Gibbs sampling has the following relationship:

$$p(\alpha_i | \alpha_{-i}, \theta, z, y) \propto p(\theta|\alpha) p(\alpha) \quad (19.53)$$

$$p(\theta_j | \theta_{-j}, \alpha, z, y) \propto p(z, y|\theta) p(\theta|\alpha) \quad (19.54)$$

**Fig. 19.12** Graphical model representation of Bayesian learning



$$p(z_k|z_{-k}, \alpha, \theta, y) \propto p(z, y|\theta) \quad (19.55)$$

where  $\alpha_{-i}$  represents all variables except variable  $\alpha_i$ ,  $\theta_{-j}$  and  $z_{-k}$  are similar. The full conditional probability distribution is proportional to the product of several conditional probability distributions, and each conditional probability distribution consists of only a small number of related variables (variables represented by adjacent nodes in the graph model). Therefore, sampling based on the full conditional probability distribution can be performed by sampling based on the product of these conditional probability distributions. This can greatly reduce the computational complexity of sampling, because the computation only involves some variables.

## Summary

1. Monte Carlo method is a method of numerical approximate computation through sampling based on probabilistic model. Monte Carlo method can be used for sampling of probability distribution, estimation of mathematical expectation of probability distribution, and approximate computation of definite integral.

Random sampling is an application of Monte Carlo method, including direct sampling method and accept-reject sampling method. The basic idea of the accept-reject method is to find a recommended distribution that is easy to sample, and whose density function is several times greater than or equal to the density function of the probability distribution we want to sample. The sample is obtained by random sampling according to the recommended distribution, and then it is randomly accepted or rejected according to the ratio of the probability distribution to be sampled to the multiple of the recommended distribution, and the above process is repeated.

Mathematical expectation estimation is another application of Monte Carlo method. According to probability distribution  $p(x)$ ,  $n$  independent samples of random variable  $x$  are drawn. According to the law of large numbers, it can be known that when the sample size increases, the sample mean of the function converges to the mathematical expectation of the function with probability 1.

$$\hat{f}_n \rightarrow E_{p(x)}[f(x)], n \rightarrow \infty$$

Calculate the sample mean  $\hat{f}_n$  as an estimate of the mathematical expectation  $E_{p(x)}[f(x)]$ .

2. A Markov chain is a random process with Markov properties

$$P(X_t | X_0 X_1 | \dots X_{t-1}) = P(X_t | X_{t-1}), t = 1, 2, \dots$$

Usually the time homogeneous Markov chain is considered. There are discrete-time Markov chains and continuous-time Markov chains, respectively defined by the probability transition matrix  $P$  and the probability transition kernel  $p(x, y)$ .

The state distribution that satisfies  $\pi = P\pi$  or  $\pi(y) = \int p(x, y)\pi(x)dx$  is called the stationary distribution of the Markov chain.

Markov chains have the properties of irreducibility, aperiodicity, and positive recurrence. If a Markov chain is irreducible, aperiodic, and positive recurrence, then the Markov chain satisfies the ergodic theorem. When the time goes to infinity, the state distribution of the Markov chain tends to a stationary distribution, and the sample of the function converges to the mathematical expectation of the function with probability on average.

$$\lim_{t \rightarrow \infty} P(X_t = i | X_0 = j) = \pi_i, \quad i = 1, 2, \dots; \quad j = 1, 2, \dots$$

$$\hat{f}_t \rightarrow E_\pi[f(X)], \quad t \rightarrow \infty$$

The reversible Markov chain is a sufficient condition to satisfy the ergodic theorem.

3. Markov Chain Monte Carlo method is a Monte Carlo integration method with Markov chain as the probabilistic model. The basic idea is as follows:

- (1) Construct a Markov chain that satisfies the conditions of the ergodic theorem on the state space  $\mathcal{X}$  of the random variable  $x$ , and its stationary distribution is the target distribution  $p(x)$ ;
- (2) Starting from a certain point  $X_0$  in the state space, use the constructed Markov chain to perform a random walk to generate the sample sequence  $X_1, X_2, \dots, X_t, \dots$ ;
- (3) Apply the Markov chain ergodic theorem to determine the positive integers  $m$  and  $n(m < n)$ , and get the sample set  $\{x_{m+1}, x_{m+2}, \dots, x_n\}$ ; perform the mean value of the function  $f(x)$  (traverse mean) estimate:

$$\hat{E} f = \frac{1}{n - m} \sum_{i=m+1}^n f(x_i)$$

4. Metropolis–Hastings algorithm is the most basic Markov Chain Monte Carlo method. Suppose the goal is to sample the probability distribution  $p(x)$ , construct the suggested distribution  $q(x, x')$ , and define the acceptance distribution  $\alpha(x, x')$ . Perform a random walk, assume that the current is in state  $x$ ,

randomly sampled according to the proposed distribution  $q(x, x')$ , accepted for sampling according to the probability  $\alpha(x, x')$  move to state  $x'$ , reject sampling according to probability  $1 - \alpha(x, x')$ , stay in state  $x'$ , and continue the above operation to obtain a series of samples. Such a random wander is based on the transfer kernel as  $p(x, x') = q(x, x')\alpha(x, x')$  of a reversible Markov chain (satisfying the ergodic determinant). Markov chain (satisfying the conditions of the ergodic theorem) is performed, and its stationary distribution is the target distribution  $p(x)$  to be sampled.

5. Gibbs sampling is used for sampling and estimation of multivariate joint distributions. Gibbs sampling is a special case of the single-component Metropolis–Hastings algorithm. This is when the proposed distribution is a full conditional probability distribution

$$q(x, x') = p(x_j' | x_{-j})$$

The basic approach of Gibbs sampling is to define the full conditional probability distribution from the joint distribution and sample from the full conditional probability distribution in turn to obtain a random sample of the joint distribution. Assuming that the multivariate joint probability distribution is  $p(x) = p(x_1, x_2, \dots, x_k)$ , Gibbs sampling starts from an initial sample  $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_k^{(0)})^T$  and iterates continuously, obtaining one sample  $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)})^T$  of the joint distribution at each iteration. In the  $i$ -th iteration, the  $j$ -th variable is sampled randomly according to the full conditional probability distribution  $p(x_j | x_1^{(i)}, \dots, x_{j-1}^{(i)}, x_{j+1}^{(i-1)}, \dots, x_k^{(i-1)})$ ,  $j = 1, 2, \dots, k$  in turn, and  $x_j^{(i)}$  is obtained. The final sample sequence  $\{x^{(0)}, x^{(1)}, \dots, x^{(n)}\}$  is obtained.

## Further Reading

An introduction to Markov chains can be found in the literature [1]. The original papers on the Metropolis–Hastings algorithm and Gibbs sampling are [2, 3], respectively. An introduction to random sampling can be found in the literature [4]. The introduction of Markov Chain Monte Carlo method can be refer to the literature [4–8]. Also watch the video on YouTube: Mathematicalmonk, Markov Chain Monte Carlo (MCMC) Introduction.

## Exercises

### 19.1 Using the Monte Carlo integration method to find

$$\int_{-\infty}^{\infty} x^2 \exp\left(-\frac{x^2}{2}\right) dx$$

- 19.2 Prove that if a Markov chain is irreducible and has one state that is aperiodic, then all other states are also aperiodic, i.e., this Markov chain is aperiodic.
- 19.3 Verify that Markov chains with the following transfer probability matrix are approximate but aperiodic.

$$\begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1 \end{bmatrix}$$

- 19.4 Verify that Markov chains with the following transfer probability matrix are reducible, but aperiodic.

$$\begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1 \\ 0 & 0 & 1/2 & 0 \end{bmatrix}$$

- 19.5 Prove that reversible Markov chains must be irreducible.
- 19.6 The single-component Metropolis–Hastings algorithm is derived from the general Metropolis–Hastings algorithm.
- 19.7 Suppose a Bernoulli experiment is conducted with posterior probability  $P(\theta|y)$ , where the variable  $y \in \{0, 1\}$  denotes the probability of the experimental the possible outcomes of the experiment, and the variable  $\theta$  denotes the probability that the outcome is 1. Suppose again that the prior probability  $P(\theta)$  follows the Beta distribution  $B(\alpha, \beta)$ , where  $\alpha = 1, \beta = 1$ ; and the likelihood function  $P(y|\theta)$  follows the binomial distribution  $Bin(n, k, \theta)$ , where  $n = 10, k = 4$ , i.e., the number of times the experiment is performed 10 times in which the result is 1 is 4. Try Metropolis–Hastings algorithm to find the mean and variance of posterior probability distribution  $P(\theta|y) \propto P(\theta)P(y|\theta)$ . (Hint: Metropolis can be used choice, i.e., the recommended distribution is assumed to be symmetric.)
- 19.8 Let there be five possible outcomes of a certain experiment, the probabilities of their occurrence are

$$\frac{\theta}{4} + \frac{1}{8}, \quad \frac{\theta}{4}, \quad \frac{\eta}{4}, \quad \frac{\eta}{4} + \frac{3}{8}, \quad \frac{1}{2}(1 - \theta - \eta)$$

The model contains two parameters  $\mu$  and  $\sigma$ , both between 0 and 1. The observed values of the results of the existing 22 trials are

$$y = (y_1, y_2, y_3, y_4, y_5) = (14, 1, 1, 1, 5)$$

where  $y_i$  denotes the number of occurrences of the  $i$ -th outcome in 22 trials and  $i = 1, 2, \dots, 5$ . Gibbs sampling was tried to estimate the mean and variance of the parameters  $\theta$  and  $\eta$ .

## References

1. Serfozo R. Basics of applied stochastic processes. Springer; 2009.
2. Metropolis N, Rosenbluth AW, Rosenbluth MN, et al. Equation of state calculations by fast computing machines. *J Chem Phys*. 1953;21(6):1087–92.
3. Geman S, Geman D. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Trans Pattern Anal Mach Intell*. 1984;6:721–41.
4. Bishop CM. Pattern recognition and machine learning. Springer; 2006.
5. Gilks WR, Richardson S, Spiegelhalter DJ. Introducing Markov chain Monte Carlo. *Markov Chain Monte Carlo in Practice*; 1996.
6. Andrieu C, De Freitas N, Doucet A, et al. An introduction to MCMC for machine learning. *Mach Learn*. 2003;50(1):5–43.
7. Hoff PD. A first course in Bayesian statistical methods. Springer; 2009.
8. Mao SS, Wang JL, Pu XL. Advanced mathematical statistics. Beijing: Higher Education Press; 1998.

# Chapter 20

## Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA), a topic model based on Bayesian learning, is an extension of latent semantic analysis, probabilistic latent semantic analysis. It was proposed by Blei et al. in 2002 and has been widely used in text data mining, image processing, bioinformatics, and other fields.

The LDA model is a generative probabilistic model for a collection of texts. It is assumed that each text is represented by a multinomial distribution of topics, and each topic is represented by a multinomial distribution of words. In particular, it is assumed that the prior distribution of the topic distribution of the text and the prior distribution of the word distribution of the topic are both Dirichlet distributions. The import of prior distributions enables LDA to better cope with the overfitting phenomenon in topic model learning.

The process of generating a text collection for LDA is as follows: firstly, a topic distribution is randomly generated for a text, and then a topic is randomly generated at each position of the text, then a word is randomly generated at that position based on the word distribution of the topic, until the last position of the text, the whole text is generated. Repeat the above process to generate all text.

The LDA model is a probabilistic graphical model with hidden variables. In the model, the word distribution of each topic, the topic distribution of each text, and the topic at each position of the text are hidden variables; The words in each position of the text are observable variables. The learning and deduction of the LDA model cannot be solved directly, so Gibbs sampling and variational EM algorithm are usually used. The former is the Monte Carlo method, and the latter is approximate algorithm.

This chapter introduces the Dirichlet distribution in Sect. 20.1, and describes the latent Dirichlet distribution model in Sect. 20.2. Sections 20.3 and 20.4 describes the algorithms of the model, including Gibbs sampling and the variational EM algorithm.

## 20.1 Dirichlet Distribution

### 20.1.1 Definition of Distribution

First, we introduce the multinomial distribution and the Dirichlet distribution, which are the basis of the LDA model.

#### 20.1.1.1 Multinomial Distribution

The multinomial distribution is a probabilistic distribution of the multivariate discrete random variable and is an extension of the binomial distribution.

Suppose that  $n$  independent randomized trials are repeated, and each trial has  $k$  possible outcomes, the probability of the  $i$ -th outcome is  $p_i$ , and the number of times that the  $i$ -th outcome occurs is  $n_i$ . If the random variable  $X = (X_1, X_2, \dots, X_k)$  is used to denote the number of times that all possible outcomes occur, where  $X_i$  denotes the number of times the  $i$ -th outcome occurs, then the random variable  $X$  obeys a multinomial distribution.

**Definition 20.1 (Multinomial Distribution)** If the probability mass function of a multivariate discrete random variable  $X = (X_1, X_2, \dots, X_k)$  is

$$\begin{aligned} P(X_1 = n_1, X_2 = n_2, \dots, X_k = n_k) &= \frac{n!}{n_1! n_2! \dots n_k!} p_1^{n_1} p_2^{n_2} \dots p_k^{n_k} \\ &= \frac{n!}{\prod_{i=1}^k n_i!} \prod_{i=1}^k p_i^{n_i} \end{aligned} \quad (20.1)$$

where  $p = (p_1, p_2, \dots, p_k)$ ,  $p_i \geq 0$ ,  $i = 1, 2, \dots, k$ ,  $\sum_{i=1}^k p_i = 1$ ,  $\sum_{i=1}^k n_i = n$ , then the random variable  $X$  is said to obey a multinomial distribution with parameters  $(n, p)$ , denoted as  $X \sim \text{Mult}(n, p)$

When the number of trials  $n$  is 1, the multinomial distribution becomes a categorical distribution. The categorical distribution represents the probability of the  $k$  possible outcomes of a trial. Clearly the multinomial distribution contains the categorical distribution.

#### 20.1.1.2 Dirichlet Distribution

The Dirichlet distribution is a probabilistic distribution of a multivariate continuous random variable and is an extension of the beta distribution. In Bayesian learning, the Dirichlet distribution is often used as the prior distribution of a multinomial distribution.

**Definition 20.2** (*Dirichlet distribution*) If the probability density function of a multivariate continuous random variable  $\theta = (\theta_1, \theta_2, \dots, \theta_k)$  is

$$p(\theta|\alpha) = \frac{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}{\prod_{i=1}^k \Gamma(\alpha_i)^k} \prod_{i=1}^k \theta_i^{\alpha_i-1} \quad (20.2)$$

where  $\sum_{i=1}^k \theta_i = 1$ ,  $\theta_i \geq 0$ ,  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ ,  $\alpha_i > 0$ ,  $i = 1, 2, \dots, k$ , then the random variable  $\theta$  is said to obey the Dirichlet distribution with parameter  $\alpha$ , denoted as  $\theta \sim Dir(\alpha)$ .

Where  $\Gamma(s)$  is the gamma function, defined as

$$\Gamma(s) = \int_0^\infty x^{s-1} e^{-x} dx, \quad s > 0$$

with the property

$$\Gamma(s+1) = s\Gamma(s)$$

When  $s$  is a natural number, we have

$$\Gamma(s+1) = s!$$

As the following conditions are met

$$\theta_i \geq 0, \sum_{i=1}^k \theta_i = 1$$

the Dirichlet distribution  $\theta$  exists on the  $(k-1)$  dimensional simplex. Figure 20.1 shows the Dirichlet distribution on a two-dimensional simplex (see the pre-colored figure in Appendix).  $\theta_1 + \theta_2 + \theta_3 = 1$ ,  $\theta_1, \theta_2, \theta_3 \geq 0$ . The parameters of the Dirichlet distribution in the figure are  $\alpha = (3, 3, 3)$ ,  $\alpha = (7, 7, 7)$ ,  $\alpha = (20, 20, 20)$ ,  $\alpha = (2, 6, 11)$ ,  $\alpha = (14, 9, 5)$ ,  $\alpha = (6, 2, 6)$ .

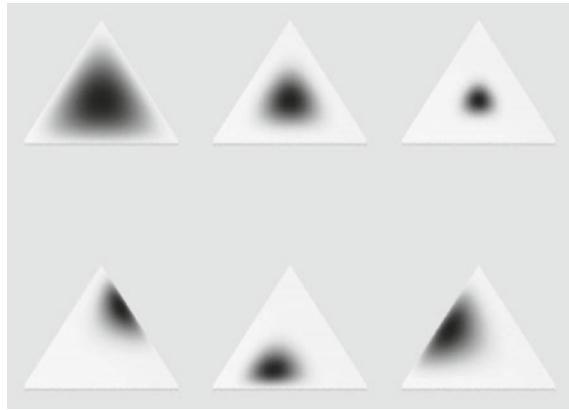
Let

$$B(\alpha) = \frac{\prod_{i=1}^k \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^k \alpha_i\right)} \quad (20.3)$$

Then the density function of the Dirichlet distribution can be written as

$$p(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^k \theta_i^{\alpha_i-1} \quad (20.4)$$

**Fig. 20.1** Examples of Dirichlet distribution



$B(\alpha)$  is the normalization factor and is called the multivariate beta function (or extended beta function). According to the properties of the density function

$$\int \frac{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k \theta_i^{\alpha_i-1} d\theta = \frac{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}{\prod_{i=1}^k \Gamma(\alpha_i)} \int \prod_{i=1}^k \theta_i^{\alpha_i-1} d\theta = 1$$

We have

$$B(\alpha) = \int \prod_{i=1}^k \theta_i^{\alpha_i-1} d\theta \quad (20.5)$$

Thus, Eq. (20.5) is the integral representation of the multivariate beta function.

### 20.1.1.3 Binomial Distribution and Beta Distribution

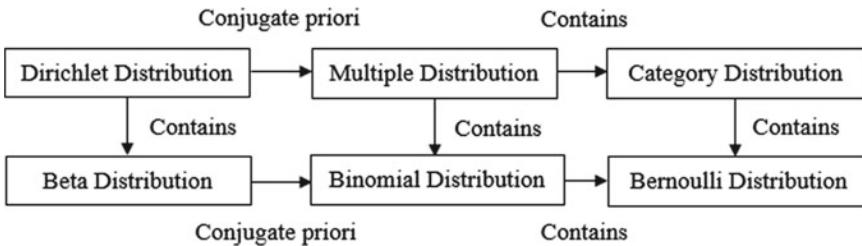
The binomial distribution is a special case of the multinomial distribution, and the beta distribution is a special case of the Dirichlet distribution.

The binomial distribution refers to the following probabilistic distribution.  $X$  is a discrete random variable that takes the value  $m$  and its probability mass function is

$$P(X = m) = \binom{n}{m} p^m (1-p)^{n-m}, \quad m = 0, 1, 2, \dots, n \quad (20.6)$$

where  $n$  and  $p(0 \leq p \leq 1)$  are parameters.

The beta distribution refers to the following probability distribution,  $X$  is a continuous random variable taking values in the range  $[0, 1]$ , and its probability density function is



**Fig. 20.2** The relationship between probabilistic distribution

$$p(x) = \begin{cases} \frac{1}{B(s,t)} x^{s-1} (1-x)^{t-1}, & 0 \leq x \leq 1 \\ 0, & \text{else} \end{cases} \quad (20.7)$$

where  $s > 0$  and  $t > 0$  are parameters,  $B(s, t) = \frac{\Gamma(s)\Gamma(t)}{\Gamma(s+t)}$  is the beta function, defined as

$$B(s, t) = \int_0^1 x^{s-1} (1-x)^{t-1} dx \quad (20.8)$$

When  $s, t$  are natural numbers,

$$B(s, t) = \frac{(s-1)!(t-1)!}{(s+t-1)!} \quad (20.9)$$

When  $n$  is 1, the binomial distribution becomes the Bernoulli distribution or the 0–1 distribution. The Bernoulli distribution represents the probability of 2 possible outcomes of the test. Clearly the binomial distribution contains the Bernoulli distribution. Figure 20.2 gives the relationship between several probability distributions.

### 20.1.2 Conjugate Prior

The Dirichlet distribution has some important properties: (1) the Dirichlet distribution belongs to the family of exponential distributions; (2) the Dirichlet distribution is a conjugate prior of a multinomial distribution.

Conjugate distributions are commonly used in Bayesian learning. If the posterior distribution and the prior distribution are of the same kind, then the two distributions are called conjugate distributions and the prior distribution is called conjugate prior. If the prior distribution of a multinomial distribution is a Dirichlet distribution, then its posterior distribution is also a Dirichlet distribution. If the prior distribution of a multinomial distribution is a Dirichlet distribution, its posterior distribution is also a

Dirichlet distribution, and the two constitute a conjugate distribution. The parameters of the Dirichlet distribution as the prior distribution are also called hyperparameters. The parameters of the Dirichlet distribution as the prior distribution are also called hyperparameters. The advantage of using the conjugate distribution is that it is easy to calculate the posterior distribution from the prior distribution.

Let  $\mathcal{W} \ominus \{w_1, w_2, \dots, w_k\}$  be a set consisting of  $k$  elements. The random variable  $\mathcal{X}$  obeys a multinomial distribution over  $\mathcal{W}$ ,  $\mathcal{X} \sim \text{Mult}(n, \theta)$ , where  $n = (n_1, n_2, \dots, n_k)$  and  $\theta = (\theta_1, \theta_2, \dots, \theta_k)$  are parameters. The parameter  $n$  is the number of times the sample is repeatedly and independently drawn from  $\mathcal{W}$ .  $n_i$  is the number of occurrences of  $w_i$  in the sample ( $i = 1, 2, \dots, k$ ); the parameter  $\theta_i$  is the probability of occurrence of  $w_i$  ( $i = 1, 2, \dots, k$ ).

Denote the sample data as  $D$ , the objective is to calculate the posterior probability  $p(\theta|D)$  of the parameter  $\theta$  given the sample data  $D$ . For a given sample data  $D$ , the likelihood function is

$$p(D|\theta) = \theta_1^{n_1} \theta_2^{n_2} \dots \theta_k^{n_k} = \prod_{i=1}^k \theta_i^{n_i} \quad (20.10)$$

Suppose that the random variable  $\theta$  obeys the Dirichlet distribution  $p(\theta|\alpha)$ , where  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$  is a parameter. Then the prior distribution of  $\theta$  is

$$p(\theta|\alpha) = \frac{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k \theta_i^{\alpha_i-1} = \frac{1}{B(\alpha)} \prod_{i=1}^k \theta_i^{\alpha_i-1} = \text{Dir}(\theta|\alpha), \alpha_i > 0 \quad (20.11)$$

According to the Bayesian rule, given the sample data  $D$  and parameters  $\alpha$ , the posterior probability distribution of  $\theta$  is

$$\begin{aligned} p(\theta|D, \alpha) &= \frac{p(D|\theta)p(\theta|\alpha)}{p(D|\alpha)} \\ &= \frac{\prod_{i=1}^k \theta_i^{n_i} \frac{1}{B(\alpha)} \theta_i^{\alpha_i-1}}{\int \prod_{i=1}^k \theta_i^{n_i} \frac{1}{B(\alpha)} \theta_i^{\alpha_i-1} d\theta} \\ &= \frac{1}{B(\alpha + n)} \prod_{i=1}^k \theta_i^{\alpha_i+n_i-1} \\ &= \text{Dir}(\theta|\alpha + n) \end{aligned} \quad (20.12)$$

It can be seen that the prior distribution (20.11) and the posterior distribution (20.12) are both Dirichlet distribution with different parameters, so Dirichlet distribution is the conjugate prior of multinomial distribution. The parameter of Dirichlet posterior distribution is equal to the parameter of Dirichlet prior distribution  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$  plus the observation count  $n = (n_1, n_2, \dots, n_k)$  of

multinomial distribution. It seems that the count  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$  has been observed before the experiment. Therefore,  $\alpha$  is also called prior pseudo-counts.

## 20.2 Latent Dirichlet Allocation Model

### 20.2.1 Basic Ideas

Latent Dirichlet allocation (LDA) is a generative probabilistic model for text collections. The model assumes that the topic is represented by the multinomial distribution of words, and the text is represented by the multinomial distribution of topics. The difference in text content is due to the different topic distributions (Strictly speaking, the multinomial distributions here are all categorical distributions. In machine learning and natural language processing, sometimes they are not strictly distinguished.)

LDA model represents the automatic generation process of text collections: firstly, multiple word distributions are generated by the prior distribution of word distribution based on word distributions (Dirichlet distribution), i.e., multiple topic contents are determined; then, based on the prior distribution of topic distribution (Dirichlet distribution), multiple topic distributions are generated, i.e., multiple text contents are determined; then, a topic sequence is generated based on each topic distribution. For each topic, words are generated based on the word distribution of the topic to form a whole word sequence, i.e., to generate text. Repeat this process to generate all the text. The word sequence of the text is the observable variable, and the topic sequence of the text is the hidden variable, the topic distribution of the text and the word distribution of the topic are also the hidden variables. Figure 20.3 shows the text generation process of LDA (see the color diagram in Appendix before the article for details).

LDA model is a probabilistic graphical model, which takes Dirichlet distribution as the prior distribution of multinomial distribution. The learning is to infer all model parameters by estimating the posterior probability distribution of a given text collection. Using LDA for topic analysis is to learn the topic distribution of each text and the word distribution of each topic for a given text collection.

It can be considered that LDA is an extension of PLSA (probabilistic latent semantic analysis), the similarity between them is they both assume that topic is the multinomial distribution of words and text is the multinomial distribution of topic. The difference is that LDA uses the Dirichlet distribution as the prior distribution, while PLSA does not use the prior distribution (or suppose the prior distribution is uniform); LDA is based on Bayesian learning, while PLSA is based on maximum likelihood estimation. The advantage of LDA is that it can prevent over-fitting in the learning process by using prior probability distribution.

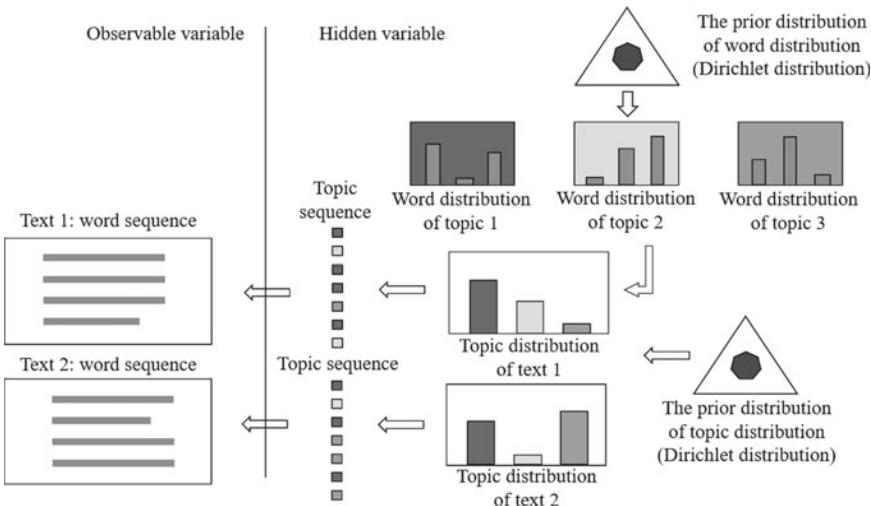


Fig. 20.3 Text generation process of LDA (see color map in Appendix)

### 20.2.2 Model Definition

This book adopts the definition of commonly used LDA model, which is slightly different from the model proposed in the original literature.

#### 20.2.2.1 Model Elements

Latent Dirichlet allocation uses three collections: One is the word collection  $W = \{w_1, \dots, w_v, \dots, w_V\}$ , where  $w_v$  is the  $v$ -th word  $v = 1, 2, \dots, V$ , and  $V$  is the number of words. The second is the text collection  $D = \{w_1, \dots, w_m, \dots, w_M\}$ , where  $w_m$  is the  $m$ -th text,  $m = 1, 2, \dots, M$ , and  $M$  is the number of texts. Text  $w_m$  is a word sequence  $w_m = (w_{m1}, \dots, w_{mn}, \dots, w_{mN_m})$ , where  $w_{mn}$  is the  $n$  word of text  $w_m$ ,  $n = 1, 2, \dots, N_m$ , and  $N_m$  is the number of words in text  $w_m$ . The third is the topic collection  $Z = \{z_1, \dots, z_k, \dots, z_K\}$ , where  $z_k$  is the  $k$ -th topic,  $k = 1, 2, \dots, K$ , and  $K$  is the number of topics.

Each topic  $z_k$  is determined by the conditional probability distribution  $p(w|z_k)$  of a word,  $w \in W$ . The distribution  $p(w|z_k)$  obeys multinomial distribution (Strictly speaking, categorical distribution) and its parameter is  $\varphi_k$ . The parameter  $\varphi_k$  obeys Dirichlet distribution (prior distribution), and its super parameter is  $\beta$ . Parameter  $\varphi_k$  is an  $V$ -dimensional vector  $\varphi_k = (\varphi_{k1}, \varphi_{k2}, \dots, \varphi_{kv})$ , where  $\varphi_{kv}$  represents the probability that topic  $z_k$  generates word  $w_v$ . The parameter vectors of all topics form a  $K \times V$  matrix  $\varphi = \{\varphi_k\}_{k=1}^K$ . The hyperparameter  $\beta$  is also a  $V$ -dimensional vector  $\beta = (\beta_1, \beta_2, \dots, \beta_V)$ .

Each text  $w_m$  is determined by the conditional probability distribution  $p(z|w_m)$  of a topic,  $z \in Z$ . The distribution  $p(z|w_m)$  obeys multinomial distribution (Strictly speaking, categorical distribution) and its parameter is  $\theta_m$ . The parameter  $\theta_m$  obeys Dirichlet distribution (prior distribution), and its super parameter is  $\alpha$ . Parameter  $\theta_m$  is an  $K$ -dimensional vector  $\theta_m = (\theta_{m1}, \theta_{m2}, \dots, \theta_{mK})$ , where  $\theta_{mk}$  represents the probability that text  $w_m$  generates topic  $z_k$ . The parameter vectors of all topics form a  $M \times K$  matrix  $\theta = \{\theta_m\}_{m=1}^M$ . The hyperparameter  $\alpha$  is also a  $K$ -dimensional vector  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_K)$ .

Every word  $w_{mn}$  in every text  $w_m$  is determined by the topic distribution of the text  $p(z|w_m)$  and the word distribution  $p(w|z_k)$  of all the topic.

### 20.2.2.2 Generative Process

The generative process of the LDA text collection is as follows:

Given the word collection  $W$ , text collection  $D$ , topic collection  $Z$ , the hyperparameter  $\alpha$  and  $\beta$  of Dirichlet distribution.

#### (1) Word Distribution of Generative topic

Randomly generate the word distribution of  $K$  topics. The specific process is as follows. According to the Dirichlet distribution  $Dir(\beta)$ , a parameter vector  $\varphi_k$ ,  $\varphi_k \sim Dir(\beta)$  is randomly generated as the word distribution  $p(w|z_k)$  of topic  $z_k$ ,  $w \in W$ ,  $k = 1, 2, \dots, K$ .

#### (2) Topic Distribution of Generative Text

Randomly generate the topic distribution of  $M$  topics. The specific process is as follows. According to the Dirichlet distribution  $Dir(\alpha)$ , a parameter vector  $\theta_m$ ,  $\theta_m \sim Dir(\alpha)$  is randomly generated as the word distribution  $p(z|w_m)$  of text  $w_m$ ,  $m = 1, 2, \dots, M$ .

#### (3) Word Sequence of Generative Text

Randomly generate  $N_m$  words of  $M$  texts. The generative process of word  $w_{mn}$  ( $n = 1, 2, \dots, N_m$ ) of text  $w_m$  ( $m = 1, 2, \dots, M$ ), is as follows:

- (3.1) Firstly, randomly generate a topic  $z_{mn}$ ,  $z_{mn} \sim Mult(\theta_m)$  according to the multinomial distribution  $Mult(\theta_m)$ .
- (3.2) Then randomly generate a word  $w_{mn}$ ,  $w_{mn} \sim Mult(\varphi_{z_{mn}})$  according to the multinomial distribution  $Mult(\varphi_{z_{mn}})$ . Text  $w_m$  is actually the word sequence  $w_m = (w_{m1}, w_{m2}, \dots, w_{mN_m})$ , corresponding to the implicit topic sequence  $z_m = (z_{m1}, z_{m2}, \dots, z_{mN_m})$ .

The algorithm of LDA text generation is summarized as follows.

#### **Algorithm 20.1** (*Text Generation Algorithm for LDA*)

- (1) For topic  $z_k$  ( $k = 1, 2, \dots, K$ ):

Generate the multinomial distribution parameter  $\varphi_k \sim \text{Dir}(\beta)$  as the word distribution  $p(w|z_k)$  of the topic;

- (2) For text  $\mathbf{w}_m (m = 1, 2, \dots, M)$ :

Generate the multinomial distribution parameter  $\theta_m \sim \text{Dir}(\alpha)$  as the topic distribution  $p(z|\mathbf{w}_m)$  of the text;

- (3) For words  $w_{mn}$  of text  $\mathbf{w}_m (m = 1, 2, \dots, M, n = 1, 2, \dots, N_m)$ :

- (a) Generate topic  $z_{mn} \sim \text{Mult}(\theta_m)$  as the corresponding topic of the word;
- (b) Generate the word  $w_{mn} \sim \text{Mult}(\varphi_{z_{mn}})$ .

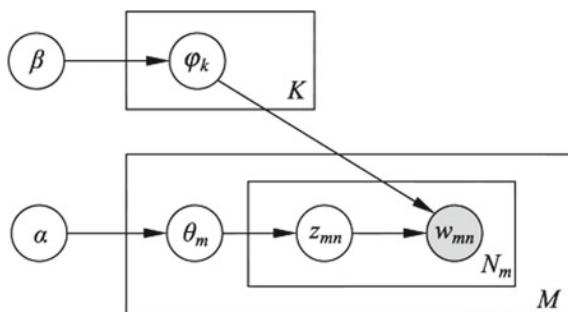
In the text generation process of LDA, it is assumed that the number of topics  $K$  is given, and the actual selection is usually made through experiments. The hyperparameters  $\alpha$  and  $\beta$  of Dirichlet distribution are also usually given in advance. With no other prior knowledge, we can assume that all the components of the vectors  $\alpha$  and  $\beta$  are 1, and the topic distribution  $\theta_m$  of the text is symmetric, as is the word distribution of the topic  $\varphi_k$ .

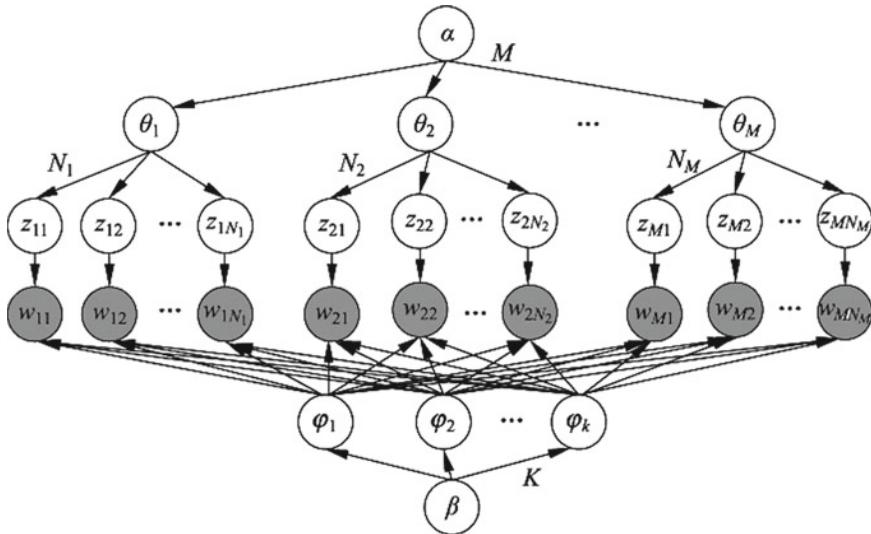
### 20.2.3 Probability Graphical Model

LDA model is essentially a probabilistic graphical model. Figure 20.4 shows the plate notation of LDA as a probabilistic graphical model. Nodes in the figure represent random variables, solid nodes are observed variables, and hollow nodes are hidden variables. Directed edges represent probabilistic dependence. Rectangles (plates) indicate repetitions, and numbers within plates indicate the number of repetitions.

The LDA block in Fig. 20.4 indicates that nodes  $\alpha$  and  $\beta$  are the hyperparameters of the model, node  $\varphi_k$  represents the parameter of the word distribution of the topic, node  $\theta_m$  represents the parameter of the topic distribution of the text, node  $z_{mn}$  represents the topic, and node  $w_{mn}$  represents the word. Node  $\beta$  points to  $\varphi_k$ , and repeat  $K$  times, representing the parameter  $\varphi_k$  that generates the word distribution of  $K$  topics according to the hyperparameter  $\beta$ ; Node  $\alpha$  points to node  $\theta_m$  and repeat  $M$  times, representing parameter  $\theta_m$  of topic distribution of  $m$  texts generated according

**Fig. 20.4** LDA plate notation





**Fig. 20.5** Expansion map model representation of the LDA

to hyperparameter  $\alpha$ ; Node  $\theta_m$  points to node  $z_{mn}$  and repeats  $N_m$  times, indicating that  $N_m$  topics  $z_{mn}$  are generated according to topic distribution  $\theta_m$  of the text; Node  $z_{mn}$  points to the node  $w_{mn}$ , and  $K$  nodes  $\varphi_k$  also points to the node  $w_{mn}$ , indicating that  $\varphi_k$  generates the word  $w_{mn}$  according to the topic  $z_{mn}$  and the distribution of the words of  $K$  topics.

The advantage of the plate notation is its simplicity. After the plate notation is expanded, it becomes a common directed graph representation (Fig. 20.5). Nodes in a directed graph represent random variables and directed edges represent probabilistic dependencies. It can be seen that LDA is a probability graph model in which the same random variable is repeatedly used.

#### 20.2.4 The Changeability of Random Variable Sequences

A finite sequence of random variables is exchangeable, which means that the joint probability distribution of the random variables is unchanged to the arrangement of the random variables.

$$P(x_1, x_2, \dots, x_N) = P(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(N)}) \quad (20.13)$$

where  $\pi(1), \pi(2), \dots, \pi(N)$  represents the natural number  $1, 2, \dots, N$  any arrangement. An infinite sequence of random variables is infinitely exchangeable, which means that any of its finite sub-sequences are exchangeable.

If a sequence of random variables  $X_1, X_2, \dots, X_N, \dots$  is independently and identically distributed, then they are infinitely commutative. The inverse is not true.

The assumption that sequences of random variables are exchangeable is commonly used in Bayesian learning. According to De Finetti theorem, any infinitely exchangeable sequence of random variables is conditionally independent and identically distributed for a random parameter. That is, the conditional probability of any infinite exchangeable random variable sequence  $X_1, X_2, \dots, X_i, \dots$ , based on a random parameter  $Y$ , is equal to the product of the conditional probabilities of all random variables  $X_1, X_2, \dots, X_i, \dots$ , based on this random parameter  $Y$ .

$$P(X_1, X_2, \dots, X_i, \dots | Y) = P(X_1|Y)P(X_2|Y) \dots P(X_i|Y) \dots \quad (20.14)$$

LDA assumes that the text consists of an infinitely exchangeable sequence of topics. According to De Finetti theorem, it is actually assumed that the topic in the text is conditionally independent and identically distributed to a random parameter. So given the parameters, the order of topics in the text can be ignored. By contrast, the probabilistic latent semantic model assumes that the topics in the text are independently and identically distributed, and the order of the topics in the text can be ignored.

### 20.2.5 Probability Formula

The LDA model as a whole is a joint probability distribution composed of observable variables and hidden variables, which can be shown as

$$p(\mathbf{w}, \mathbf{z}, \theta, \varphi | \alpha, \beta) = \prod_{k=1}^K p(\varphi_k | \beta) \prod_{m=1}^M p(\theta_m | \alpha) \prod_{n=1}^{N_m} p(z_{mn} | \theta_m) p(w_{mn} | z_{mn}, \varphi) \quad (20.15)$$

where the observable variable  $\mathbf{w}$  means the word sequence in all texts, the hidden variable  $\mathbf{z}$  means the topic sequence in all texts, the hidden variable  $\theta$  means the topic distribution parameter of all texts, the hidden variable  $\varphi$  means the word distribution parameter of all topics,  $\alpha$  and  $\beta$  are the hyperparameters  $p(\varphi_k | \beta)$

represents the generative probability of the word distribution parameter  $\varphi_k$  of the  $k$ -th topic under the condition that the hyperparameter  $\beta$  is given,  $p(\theta_m | \alpha)$  represents the generative probability of the topic distribution parameter  $\theta_m$  of the  $m$ -th text under the condition that the hyperparameter  $\alpha$  is given,  $p(z_{mn} | \theta_m)$  represents the generative probability of the topic  $z_{mn}$  at the  $n$ -th position of the text under the condition that the topic distribution  $\theta_m$  of the  $m$ -th text is given,  $p(w_{mn} | z_{mn}, \varphi)$  represents the generative probability of the topic  $w_{mn}$  at the  $n$ -th position of the text under the condition that the topic  $z_{mn}$  at the  $n$ -th position of the  $m$ -th text and the word distribution parameter  $\varphi$  of all topics are given. Details are shown in Fig. 20.5.

The joint probability distribution of the  $m$ -th text can be expressed as

$$p(\mathbf{w}_m, \mathbf{z}_m, \theta_m, \varphi | \alpha, \beta) = \prod_{k=1}^K p(\varphi_k | \beta) p(\theta_m | \alpha) \prod_{m=1}^{N_m} p(z_{mn} | \theta_m) p(w_{mn} | z_{mn}, \varphi) \quad (20.16)$$

where  $\mathbf{w}_m$  represents the sequence of words in the text,  $\mathbf{z}_m$  represents the sequence of topics in the text, and  $\theta_m$  represents the topic distribution parameter of the text.

The joint distribution of LDA model contains hidden variables, and edge distribution is obtained by integrating the hidden variables.

The probability of generating  $m$  text under the parameters  $\theta_m$  and  $\varphi$  given is

$$p(\mathbf{w}_m | \theta_m, \varphi) = \prod_{n=1}^{N_m} \left[ \sum_{k=1}^K p(z_{mn} = k | \theta_m) p(w_{mn} | \varphi_k) \right] \quad (20.17)$$

The generative probability of the  $m$ -th text given the hyperparameters  $\alpha$  and  $\beta$  is

$$p(\mathbf{w}_m | \theta_m, \varphi) = \prod_{n=1}^K \int p(\varphi_k | \beta) \left[ \int p(\theta_m | \alpha) \prod_{n=1}^{N_m} \left[ \sum_{l=1}^K p(z_{mn} = l | \theta_m) p(w_{mn} | \varphi_l) \right] d\theta_m \right] d\varphi_k \quad (20.18)$$

The generative probability of all texts under the condition that hyperparameters  $\alpha$  and  $\beta$  are given is

$$p(\mathbf{w} | \alpha, \beta) = \prod_{k=1}^K \int p(\varphi_k | \beta) \left[ \prod_{m=1}^M \int p(\theta_m | \alpha) \prod_{n=1}^{N_m} \left[ \sum_{l=1}^K p(z_{mn} = l | \theta_m) p(w_{mn} | \varphi_l) \right] d\theta_m \right] d\varphi_k \quad (20.19)$$

## 20.3 Gibbs Sampling Algorithm for LDA

The learning (parameter estimation) of Latent Dirichlet Allocation (LDA) is a complex optimization problem, which is difficult to solve accurately and can only be solved approximately. Commonly-used approximate solutions are Gibbs sampling and variational inference. This section describes Gibbs sampling, and the next section

describes the variational inference algorithm. The advantage of Gibbs sampling is that it is simple to implement, but the disadvantage is that the number of iterations may be more.

### 20.3.1 Basic Ideas

The learning of LDA model, given the set of text (word sequence)  $D = \{w_1, \dots, w_m, \dots, w_M\}$ , where  $w_m$  is the  $m$ -th text (word sequence),  $w_m = (w_{m1}, \dots, w_{mm}, \dots, w_{mN_m})$ , the word sequence of the text collection is represented by  $w$ , that is,  $w = (w_{11}, w_{12}, \dots, w_{1N_1}, w_{21}, w_{22}, \dots, w_{2N_2}, \dots, w_{M1}, w_{M2}, \dots, w_{MN_M})$  (refer to Fig. 20.5); the hyperparameters  $\alpha$  and  $\beta$  are known. The goal is to infer: (1) The set of topic sequences  $\mathbf{z} = \{\mathbf{z}_1, \dots, \mathbf{z}_m, \dots, \mathbf{z}_M\}$ ; the posterior probability distribution of  $\mathbf{z}_m$ , where  $\mathbf{z}_m$  is the topic sequence of the  $m$ -th text,  $\mathbf{z}_m = (z_{m1}, \dots, z_{mm}, \dots, z_{mN_m})$ ; (2) parameter  $\theta = \{\theta_1, \dots, \theta_m, \dots, \theta_M\}$ , where  $\theta_m$  is the parameter of the topic distribution of the text  $w_m$ ; (3) parameter  $\varphi = \{\varphi_1, \dots, \varphi_k, \dots, \varphi_M\}$ , where  $\varphi_k$  is the parameter of the word distribution of topic  $z_k$ . That is, to estimate the joint probability distribution  $p(\mathbf{w}, \mathbf{z}, \theta, \varphi | \alpha, \beta)$ , where  $\mathbf{w}$  is the observed variable, and  $\mathbf{z}, \theta, \varphi$  are hidden variables.

Chapter 19 describes Gibbs sampling, which is a commonly used Markov chain Monte Carlo method. In order to estimate the joint distribution  $p(x)$  of the multivariate random variable  $x$ , the Gibbs sampling method selects one component of  $x$ , fixes the other components, and randomly samples according to its conditional probability distribution. This operation is performed on each component in turn to obtain the joint A random sample of the distribution  $p(x)$ , repeat this process, after the combustion period, get a sample set of the joint probability distribution  $p(x)$ .

The learning of the LDA model usually uses the collapsed Gibbs sampling method.<sup>1</sup> The basic idea is to obtain the marginal probability distribution  $p(\mathbf{w}, \mathbf{z} | \alpha, \beta)$  (also a joint distribution) by integrating the hidden variables  $\theta$  and  $\varphi$ , Where the variable  $\mathbf{w}$  is observable and the variable  $\mathbf{z}$  is unobservable; Gibbs sampling is performed on the posterior probability distribution  $p(\mathbf{z}, \mathbf{w} | \alpha, \beta)$  to obtain the sample set of the distribution  $p(\mathbf{z}, \mathbf{w} | \alpha, \beta)$ ; Use this sample set to estimate the parameters  $\theta$  and  $\varphi$ , and finally get all parameter estimates of the LDA model  $p(\mathbf{w}, \mathbf{z}, \theta, \varphi | \alpha, \beta)$ .

---

<sup>1</sup> In principle, full Gibbs sampling can also be considered, but the algorithm is more complicated.

### 20.3.2 Major Parts of Algorithm

According to the above analysis, the problem is transformed to the Gibbs sampling of the posterior probability distribution  $p(z|w, \alpha, \beta)$ , which represents the conditional probabilities of all possible topic sequences under the condition that the word sequences of all texts are given. In this paper, we first give the expression of the distribution, and then give the expression of the full condition distribution of the distribution.

#### 20.3.2.1 The Expression of Sampling Distribution

First of all, it matters

$$p(z|w, \alpha, \beta) = \frac{p(w, z|\alpha, \beta)}{p(w|\alpha, \beta)} \propto p(w, z|\alpha, \beta) \quad (20.20)$$

Here, the variable  $W$ ,  $\alpha$  and  $\beta$ , if it is known that the denominator is the same, it may not be considered. Joint distribution  $p(W, Z|\alpha, \beta)$  can be further decomposed into

$$p(w, z|\alpha, \beta) = p(w|z, \alpha, \beta)p(z|\alpha, \beta) = p(w|z, \beta)p(z|\alpha) \quad (20.21)$$

The two factors can be treated separately.

The expression of the first factor  $p(w|z, \beta)$  is derived. First,

$$p(w|z, \varphi) = \prod_{k=1}^K \prod_{v=1}^V \varphi_{kv}^{n_{kv}} \quad (20.22)$$

Among them  $\varphi_{kv}$  is the probability that the  $k$ -th topic generates the  $v$ -th word in the word set, and  $n_{kv}$  is the number of times that the  $k$ -th topic generates the  $v$ -th word in the data. Therefore

$$\begin{aligned} p(w|z, \beta) &= \int p(w|z, \beta)p(\varphi|\beta) d\varphi \\ &= \int \prod_{k=1}^K \frac{1}{B(\beta)} \prod_{v=1}^V \varphi_{kv}^{n_{kv} + \beta_v - 1} d\varphi \\ &= \prod_{k=1}^K \frac{1}{B(\beta)} \int \prod_{v=1}^V \varphi_{kv}^{n_{kv} + \beta_v - 1} d\varphi \\ &= \prod_{k=1}^K \frac{B(n_k + \beta)}{B(\beta)} \end{aligned} \quad (20.23)$$

where  $n_k = \{n_{k1}, n_{k2}, \dots, n_{kV}\}$ .

The expression of the second factor is  $p(z|\alpha)$  can be derived similarly. First,

$$p(z|\theta) = \prod_{m=1}^M \prod_{k=1}^K \theta_{mk}^{n_{mk}} \quad (20.24)$$

where  $\theta_{mk}$  is the probability that the  $m$ -th text generates the  $k$ -th topic, and  $n_{mk}$  is the number of times the  $m$ -th text generates the  $k$ -th topic in the data. Then

$$\begin{aligned} p(z|\alpha) &= \int p(z|\theta)p(\theta|\alpha)d\theta \\ &= \int \prod_{m=1}^M \frac{1}{B(\alpha)} \prod_{k=1}^K \theta_{mk}^{n_{mk}+\alpha_k-1} d\theta \\ &= \prod_{m=1}^M \frac{1}{B(\alpha)} \int \prod_{k=1}^K \theta_{mk}^{n_{mk}+\alpha_k-1} d\theta \\ &= \prod_{m=1}^M \frac{B(n_m + \alpha)}{B(\alpha)} \end{aligned} \quad (20.25)$$

where  $n_m = \{n_{m1}, n_{m2}, \dots, n_{mK}\}$ . From Eqs. (20.23) and (20.25), we get

$$p(z, w|\alpha, \beta) = \prod_{k=1}^K \frac{B(n_k + \beta)}{B(\beta)} \cdot \prod_{m=1}^M \frac{B(n_m + \alpha)}{B(\alpha)} \quad (20.26)$$

Therefore, from Eqs. (20.20) and (20.26), the formula for the contracted Gibbs sampling distribution

$$p(z_i|z_{-i}, w, \alpha, \beta) \propto \frac{n_{kv} + \beta_v}{\sum_{v=1}^V (n_{kv} + \beta_v)} \cdot \frac{n_{mk} + \alpha_k}{\sum_{k=1}^K (n_{mk} + \alpha_k)} \quad (20.27)$$

### 20.3.2.2 Expression of Fully Conditional Distribution

The fully conditional distribution of the distribution  $p(z|w, \alpha, \beta)$  can be written as

$$p(z_i|z_{-i}, w, \alpha, \beta) = \frac{1}{Z_{z_i}} p(z|w, \alpha, \beta) \quad (20.28)$$

Here  $w_i$  represents the word at the  $i$ -th position in the word sequence of all texts, and  $z_i$  represents the topic corresponding to the word  $w_i$ ,  $i = (m, n)$ ,  $i = 1, 2, \dots, I$ ,  $z_{-i} = \{z_j : j \neq i\}$ ,  $Z_{z_i}$  represents the marginalization factor of

the distribution  $p(z|w, \alpha, \beta)$  on the variable  $z_i$  formula. Equation (20.28) is the conditional probability distribution of the topic at the  $i$ -th position under the given conditions of all text word sequences and topic sequences in other positions. From Eqs. (20.27) and (20.28), we can deduce

$$p(z_i | z_{-i}, w, \alpha, \beta) \propto \frac{n_{kv} + \beta_v}{\sum_{v=1}^V (n_{kv} + \beta_v)} \cdot \frac{n_{mk} + \alpha_k}{\sum_{k=1}^K (n_{mk} + \alpha_k)} \quad (20.29)$$

Among them, the word  $w_i$  at the  $n$ -th position of the  $m$ -th text is the  $v$ -th word in the word set, and its topic  $z_i$  is the  $k$ -th topic in the topic set, and  $n_{kv}$  represents the count of the  $v$ -th word in the  $k$ -th topic, but minus Go to the count of the current word,  $n_{mk}$  represents the count of the  $k$ -th topic in the  $m$ -th text, but subtracts the count of the topic of the current word.

Where the word  $w_i$  at the  $n$ -th position of the  $m$ -th text is the  $v$ -th word in the word set, and its topic  $z_i$  is the  $k$ -th topic in the topic set,  $n_{kv}$  is the  $v$ -th word in the  $k$ -th topic count, but minus the count of the current word, and  $n_{mk}$  represents the count of the  $k$ -th topic in the  $m$ -th text, but subtracts the count of the topic of the current word.

### 20.3.3 Algorithm Post-processing

The sample of the distribution  $p(z|w, \alpha, \beta)$  obtained by Gibbs sampling can obtain the distribution value of the variable  $z$ , and also estimate the variables  $\theta$  and  $\varphi$ .

#### 20.3.3.1 Estimate of the Parameter $\theta = \{\theta_m\}$

According to the definition of the LDA model, the posterior probability satisfies

$$p(\theta_m | z_m, \alpha) = \frac{1}{Z_{\theta_m}} \prod_{n=1}^{N_m} p(z_{mn} | \theta_m) p(\theta_m | \alpha) = \text{Dir}(\theta_m | n_m + \alpha) \quad (20.30)$$

where  $n_m = \{n_{m1}, n_{m2}, \dots, n_{mK}\}$  is the count of the  $m$ -th text topic, and  $Z_{\theta_m}$  represents the marginalization factor of the distribution  $p(\theta_m, z_m | \alpha)$  to the variable  $\theta_m$ . Then we get the estimated formula of parameter  $\theta = \{\theta_m\}$ :

$$\theta_{mk} = \frac{n_{mk} + \alpha_k}{\sum_{k=1}^K (n_{mk} + \alpha_k)}, \quad m = 1, 2, \dots, M; \quad k = 1, 2, \dots, K \quad (20.31)$$

### 20.3.3.2 Estimate of the Parameter $\varphi = \{\varphi_k\}$

The posterior probability satisfies

$$p(\varphi_k | w, z, \beta) = \frac{1}{Z_{\varphi_k}} \prod_{n=1}^I p(w_i | \varphi_k) p(\varphi_k | \beta) = \text{Dir}(\varphi_k | n_k + \beta) \quad (20.32)$$

where  $n_k = \{n_{k1}, n_{k2}, \dots, n_{kV}\}$  is the count of words in the  $k$ -th topic,  $Z_{\varphi_k}$  represents the marginalization factor of the distribution  $p(\varphi_k, w|z, \beta)$  to the variable  $\varphi_k$ , and  $I$  is the total number of words in the word sequence  $w$  in the text collection. Then we get the estimated formula of parameter:

$$\varphi_{kv} = \frac{n_{kv} + \beta_v}{\sum_{v=1}^V n_{kv} + \beta_v}, \quad k = 1, 2, \dots, K; \quad v = 1, 2, \dots, V \quad (20.33)$$

## 20.3.4 Algorithm

Summarize the specific algorithm of Gibbs sampling of LDA.

For a given word sequence  $w$  of all texts, a topic is randomly assigned at each position, which constitutes the topic sequence  $z$  of all texts as a whole. Then cycle through the following operations.

At each position, the full conditional probability distribution of the topic at that position is calculated, and then random sampling is performed to obtain a new topic at that position and assign it to this position.

$$p(z_i | z_{-i}, w, \alpha, \beta) \propto \frac{n_{kv} + \beta_v}{\sum_{v=1}^V (n_{kv} + \beta_v)} \cdot \frac{n_{mb} + \alpha_k}{\sum_{k=1}^K (n_{mb} + \alpha_k)}$$

This conditional probability distribution is composed of two factors, the first factor represents the probability that the topic generates the word at that location, and the second factor represents the probability that the text at that location generates the topic.

Prepare two count matrices as a whole: topic-word matrix  $N_{K \times V} = [n_{kv}]$  and text-topic matrix  $N_{M \times K} = [n_{mk}]$ . At each position, subtract 1 from the existing topic count at that position in the two matrices, calculate full conditional probability distribution, and then sample to get the new topic at this position. Then add 1 to the count of the new topic at that position in the two matrices and move to the next position.

The topic sequence of all texts obtained after the burning period is a sample of the conditional probability distribution  $p(z|w, \alpha, \beta)$ .

**Algorithm 20.2 (LDA Gibbs sampling Algorithm)**

Input: The word sequence of the text  $\mathbf{w} = \{\mathbf{w}_1, \dots, \mathbf{w}_m, \dots, \mathbf{w}_M\}$ ,  $\mathbf{w}_m = \{w_{m1}, \dots, w_{mm}, \dots, w_{mN_m}\}$ ;

Output: The topic sequence of the text  $\mathbf{z} = \{z_1, \dots, z_m, \dots, z_M\}$ , the posterior probability distribution  $p(\mathbf{z}|\mathbf{w}, \alpha, \beta)$  sample counts of  $Z_m = \{z_{m1}, \dots, z_{mm}, \dots, z_{mN_m}\}$ , the estimated values of the model parameters  $\varphi$  and  $\theta$ ;

Parameters: hyperparameters  $\alpha$  and  $\beta$ , the number of topics  $K$ .

- (1) Set the initial value of all the elements of the counting matrix  $n_{mk}$ ,  $n_{kv}$ , and the elements of the counting vector  $n_m$ ,  $n_k$  to 0;

- (2) For all text  $\mathbf{w}_m$ ,  $m = 1, 2, \dots, M$

For all words  $w_{mn}$  in the m-th text,  $n = 1, 2, \dots, N_m$ .

- (a) Sampling topic  $z_{mn} = z_k \sim \text{Mult}(\frac{1}{K})$ ;

Increase text-topic count  $n_{mk} = n_{mk} + 1$ ,

Add text-topic and count  $n_m = n_m + 1$ ,

Increase topic-word count  $n_{kv} = n_{kv} + 1$ ,

Increase topic-words and count  $n_k = n_k + 1$ ;

- (3) Execute the following operations cyclically until it enters the combustion period

For all text  $\mathbf{w}_m$ ,  $m = 1, 2, \dots, M$

For all words in the m-th text  $w_{mn}$ ,  $n = 1, 2, \dots, N_m$ .

- (a) The current word  $w_{mn}$  is the v-th word, and the topic assignment  $z_{mn}$  is the k-th topic; Decrease count  $n_{mk} = n_{mk} - 1$ ,  $n_m = n_m - 1$ ,  $n_{kv} = n_{kv} - 1$ ,  $n_k = n_k - 1$ ;

- (b) Sampling was carried out according to the full condition distribution

$$p(z_i | \mathbf{Z}_{-i}, \mathbf{w}, \alpha, \beta) \propto \frac{n_{kv} + \beta_v}{\sum_{v=1}^V (n_{kv} + \beta_v)} \cdot \frac{n_{mk} + \alpha_k}{\sum_{k=1}^K (n_{mk} + \alpha_k)}$$

Get the new  $k'$  topic and assign it to  $z_{mn}$ .

- (c) Increase count  $n_{mk'} = n_{mk'} + 1$ ,  $n_m = n_m + 1$ ,  $n_{k'v} = n_{k'v} + 1$ ,  $n_{k'} = n_{k'} + 1$ .

- (d) Two updated counting matrices are obtained  $N_{K \times V} = [n_{kv}]$  and  $N_{M \times K} = [n_{mk}]$ , denotes the sample count of posterior probability distribution  $p(\mathbf{z}|\mathbf{w}, \alpha, \beta)$ ;

- (4) The model parameters are calculated by using the sample count

$$\theta_{mk} = \frac{n_{mk} + \alpha_k}{\sum_{k=1}^K (n_{mk} + \alpha_k)}$$

$$\varphi_{kv} = \frac{n_{kv} + \beta_v}{\sum_{v=1}^V (n_{kv} + \beta_v)}$$

## 20.4 Variational EM Algorithm for LDA

This section first introduces the variational reasoning, then introduces the variational EM algorithm, and finally introduces the specific algorithm of applying the variational EM algorithm to LDA model learning. The variational EM algorithm of LDA has the advantages of high efficiency of reasoning and learning.

### 20.4.1 Variational Reasoning

Variational inference is a common learning and reasoning method with hidden variable model in Bayesian learning. Variational reasoning and Markov chain Monte Carlo (MCMC) belong to different techniques. MCMC approximately calculates the posterior probability of the model by random sampling, while variational reasoning approximately calculates the posterior probability of the model by analytic method.

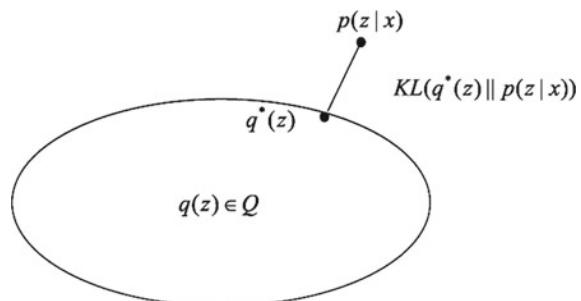
The basic idea of variational reasoning is as follows. Suppose the model is a joint probability distribution  $p(x, z)$ , where  $x$  is the observed variable (data) and  $z$  is the hidden variable, including parameters. The goal is to learn the posterior probability distribution  $p(z|x)$  of the model and use the model for probability reasoning. But it is a complex distribution, so it is difficult to estimate the parameters of the distribution directly. Therefore, consider using the probability distribution  $q(z)$  to approximate the conditional probability distribution  $p(z|x)$ , and using the KL divergence  $D(q(z)\|p(z|x))$  to calculate the similarity between the above two distribution,  $q(z)$  is called the variational distribution. If you can find the nearest distribution  $q^*(z)$  to  $p(z|x)$  in the sense of KL divergence, you can use this distribution to approximate  $p(z|x)$ .

$$p(z|x) \approx q^*(z) \quad (20.34)$$

Figure 20.6 shows the relationship between  $q^*(z)$  and  $p(z|x)$ . See Appendix E for the definition of KL divergence.

KL divergence can be written in the following form

**Fig. 20.6** Principle of variational inference



$$\begin{aligned}
D(q(z)\|p(z|x)) &= E_q[\log q(z)] - E_q[\log p(z|x)] \\
&= E_q[\log q(z)] - E_q[\log p(x, z)] + \log p(x) \\
&= \log p(x) - \{E_q[\log p(x, z)] - E_q[\log q(z)]\} \quad (20.35)
\end{aligned}$$

Note that the KL divergence is greater than or equal to zero, and is zero if and only if the two distributions are consistent, it can be seen that the first term and the second term at the right end of the formula (20.35) satisfy the relationship

$$\log p(x) \geq E_q[\log p(x, z)] - E_q[\log q(z)] \quad (20.36)$$

The right end of the inequality is the lower bound of the left end, the left end is called the evidence, the right end is called the evidence lower bound (ELBO), and the lower bound of evidence is denoted as

$$L(q) = E_q[\log p(x, z)] - E_q[\log q(z)] \quad (20.37)$$

The minimization of KL divergence (20.35) can be achieved by maximizing the lower bound of evidence (20.37), because the goal is to find  $q(z)$  to minimize KL divergence, at this time  $\log p(x)$  is constant. Therefore, variational inference becomes a problem of maximizing the evidence lower bound.

Variational inference can be understood from another perspective. The goal is to estimate the joint probability distribution  $p(x, z)$  by maximizing the evidence  $\log p(x)$ . Because there is a hidden variable  $z$ , it is difficult to maximize the evidence directly, instead, we maximize the lower bound of evidence according to Eq. (20.36).

It is easy to deal with the requirement of variational distribution  $q(z)$ . It is usually assumed that  $q(z)$  is independent of all components of  $z$  (in fact, the condition is independent of parameters), i.e.,  $q(z)$  satisfies the requirement

$$q(z) = q(z_1)q(z_2)\dots q(z_n) \quad (20.38)$$

The variational distribution at this time is called mean field.<sup>2</sup> The minimization of KL divergence or the maximization of the evidence lower bound is actually carried out in the set of mean fields, i.e., the distribution set  $Q = \{q(z)|q(z) = \prod_{i=1}^n q(z_i)\}$  satisfying the independent hypothesis.

To sum up, variational inference has the following steps: define the variational distribution  $q(z)$ ; derive the expression of its evidence lower bound. Using the optimization method to optimize the evidence lower bound, such as coordinate rise, the optimal distribution  $q^*(z)$  is obtained and as an approximation of the posterior distribution  $p(z|x)$ .

---

<sup>2</sup> The concept of mean field originated from physics.

### 20.4.2 Variational EM Algorithm

In variational inference, the evidence lower bound can be maximized by iterative method. At this time, the algorithm is the extension of EM algorithm, which is called variational EM algorithm.

Suppose the model is a joint probability distribution  $p(x, z|\theta)$ , where  $x$  is the observed variable and  $z$  is the hidden variable,  $\theta$  is the parameter. The goal is to estimate the parameter  $\theta$  of the model by maximizing the probability (evidence)  $\log p(x|\theta)$  of the observed data. Using variational inference, the mean field  $q(z) = \prod_{i=1}^n q(z_i)$  is introduced to define the evidence lower bound

$$L(q, \theta) = E_q[\log p(x, z|\theta)] - E_q[\log q(z)] \quad (20.39)$$

Through iteration, the evidence lower bound is maximized with  $q$  and  $\theta$  as variables, and the variational EM algorithm is obtained.

**Algorithm 20.3** (*Variational EM Algorithm*)

Loop through the following step E and M until convergence.

1. Step E: fix  $\theta$ , find the maximization of  $L(q, \theta)$  to  $q$ .
2. Step M: fix  $q$ , find the maximum of  $L(q, \theta)$  to  $\theta$ .

Give an estimate value of the model parameter  $\theta$ .

According to the principle of variational inference, the probability and evidence lower bound of observation data are satisfied

$$\log p(x|\theta) - L(q, \theta) = D(q(z)\|p(z|x, \theta)) \geq 0 \quad (20.40)$$

The following relations hold during the iterations of the variational EM algorithm:

$$\log p(x|\theta^{(t-1)}) = L(q^{(t)}, \theta^{(t-1)}) \leq L(q^{(t)}, \theta^{(t)}) \leq \log p(x|\theta^{(t)}) \quad (20.41)$$

where the upper corners  $t - 1$  and  $t$  denote the number of iterations, the equation on the left is based on the E-step calculation and the variational inference principle, the inequality in the middle is based on the M-step calculation, and the inequality on the right is based on the variational inference principle. It means that each iteration is guaranteed to have a non-decreasing probability of the observed data. Therefore, the variational EM algorithm must converge, but may converge to a local optimum.

The EM algorithm maximizes the evidence lower bound as well. It may be useful to compare the extension of the EM algorithm in Sect. 9.4, the extension of the EM algorithm is to find the maximum-maximum algorithm of the  $F$  function, where the  $F$  function is the evidence lower bound. The EM algorithm assumes that  $q(z) = p(z|x)$  and  $p(z|x)$  is easy to compute, while the variational EM algorithm considers the general case of using the easily computable mean field  $q(z) = \prod_{i=1}^n q(z_i)$ . When the model is complex, the EM algorithm may not be available, but the variational EM algorithm is still can be used.

### 20.4.3 Algorithm Derivation

The application of the variational EM algorithm to the learning of the LDA model in Fig. 20.7 is a simplification of the LDA model in Fig. 20.4. First, the specific variational distribution is defined, an expression for the evidence lower bound is derived, then derive the estimation formula of the parameters of the variational distribution and the parameters of the LDA model, and finally the variational EM algorithm of the LDA model is given.

#### 20.4.3.1 Definition of the Evidence Lower Bound

For simplicity, one text at a time is considered, denoted as  $\mathbf{w}$ . The word sequence  $\mathbf{w} = (w_1, \dots, w_n, \dots, w_N)$  of the text, the corresponding topic sequence  $\mathbf{z} = (z_1, \dots, z_n, \dots, z_N)$ , and the topic distribution  $\theta$ , the joint distribution of the random variables  $\mathbf{w}$ ,  $\mathbf{z}$  and  $\theta$  is

$$p(\theta, \mathbf{z}, \mathbf{w} | \alpha, \varphi) = p(\theta | \alpha) \prod_{n=1}^N p(z_n | \theta) p(w_n | z_n, \varphi) \quad (20.42)$$

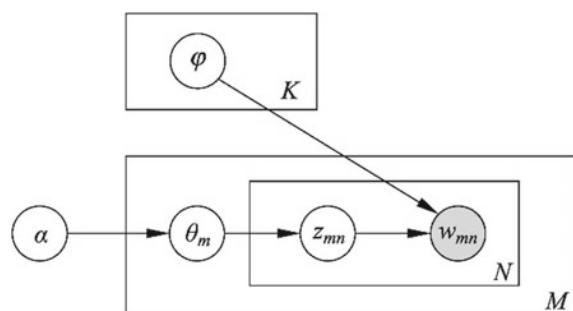
where  $\mathbf{w}$  is the observable variable,  $\theta$  and  $\mathbf{z}$  are the hidden variables,  $\alpha$  and  $\varphi$  are the parameters.

Define the variational distribution based on the mean field

$$q(\theta, \mathbf{z} | \gamma, \eta) = q(\theta | \gamma) \prod_{n=1}^N q(z_n | \eta_n) \quad (20.43)$$

where  $\gamma$  is a Dirichlet distribution parameter,  $\eta = (\eta_1, \eta_2, \dots, \eta_n)$  is a multinomial distribution parameter, and the components of variable  $\theta$  and  $\mathbf{z}$  are conditionally independent. The goal is to find the most similar variational distribution  $q(\theta, \mathbf{z} | \gamma, \eta)$  in the sense of KL divergence in order to approximate the posterior distribution  $p(\theta, \mathbf{z} | \mathbf{w}, \alpha, \varphi)$  of the LDA model.

**Fig. 20.7** LDA model



**Fig. 20.8** The variational distribution based on the mean field

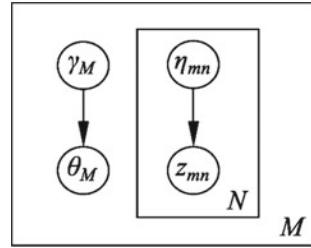


Figure 20.8 is the plate notation of the variational distribution. There are dependencies between hidden variables  $\theta$  and  $\mathbf{z}$  in the LDA model, and these dependencies are removed from the variational distribution, so the variables  $\theta$  and  $\mathbf{z}$  are conditionally independent.

Then the evidence lower bound for the text is obtained

$$L(\gamma, \eta, \alpha, \varphi) = E_q[\log p(\theta, \mathbf{z}, \mathbf{w} | \alpha, \varphi)] - E_q[\log q(\theta, \mathbf{z} | \gamma, \eta)] \quad (20.44)$$

where the mathematical expectation is defined to the distribution  $q(\theta, \mathbf{z} | \gamma, \eta)$ , written as  $E_q[\cdot]$  for convenience.  $\gamma$  and  $\eta$  are parameters of the variational distribution,  $\alpha$  and  $\varphi$  are parameters of the LDA model.

The evidence lower bound for all texts is

$$L_w(\gamma, \eta, \alpha, \varphi) = \sum_{m=1}^M \{E_{qm}[\log p(\theta_m, z_m, w_m | \alpha, \varphi)] - E_{qm}[\log q(\theta_m, z_m | \gamma_m, \eta_m)]\} \quad (20.45)$$

To solve the maximization of the evidence lower bound  $L(\gamma, \eta, \alpha, \varphi)$ , firstly, the expression of the ELBO is given. Therefore, the ELBO Eq. (20.44) is expanded

$$\begin{aligned} L(\gamma, \eta, \alpha, \varphi) &= E_q[\log p(\theta | \alpha)] + E_q[\log p(\mathbf{z} | \theta)] + E_q[\log p(\mathbf{w} | \mathbf{z}, \varphi)] \\ &\quad - E_q[\log q(\theta | \gamma)] - E_q[\log q(\mathbf{z} | \eta)] \end{aligned} \quad (20.46)$$

The model parameters  $\alpha$  and  $\varphi$  continue to expand based on the variational parameters  $\gamma$  and  $\eta$ , and each term of the expansion is written as a row

$$\begin{aligned} L(\gamma, \eta, \alpha, \varphi) &= \log \Gamma \left( \sum_{l=1}^K \alpha_l \right) - \sum_{k=1}^K \log \Gamma(\alpha_k) \\ &\quad + \sum_{k=1}^K (\alpha_k - 1) \left[ \Psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) \right] \\ &\quad + \sum_{n=1}^N \sum_{k=1}^K \eta_{nk} \left[ \Psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) \right] \end{aligned}$$

$$\begin{aligned}
& + \sum_{n=1}^N \sum_{k=1}^K \sum_{v=1}^V \eta_{nk} w_n^v \log \varphi_{kv} \\
& - \log \Gamma \left( \sum_{l=1}^K \gamma_l \right) + \sum_{k=1}^K \log \Gamma(\gamma_k) \\
& - \sum_{k=1}^K (\gamma_k - 1) \left[ \Psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) \right] \\
& - \sum_{n=1}^N \sum_{k=1}^K \eta_{nk} \log \eta_{nk}
\end{aligned} \tag{20.47}$$

where  $\Psi(\alpha_k)$  is the derivative of the logarithmic gamma function, i.e.

$$\Psi(\alpha_k) = \frac{d}{d\alpha_k} \log \Gamma(\alpha_k) \tag{20.48}$$

The first derivation is to find  $E_q[\log p(\theta|\alpha)]$ , which is about the distribution  $q[\theta, \mathbf{z}|\gamma, \eta]$  of mathematical expectation.

$$E_q[\log p(\theta|\alpha)] = \sum_{k=1}^K (\alpha_k - 1) E_q[\log \theta_k] + \log \Gamma \left( \sum_{l=1}^K \alpha_l \right) - \sum_{k=1}^K \log \Gamma(\alpha_k) \tag{20.49}$$

where  $\theta \sim \text{Dir}(\theta|\gamma)$ , so the use of Appendix E equation (E.7)

$$E_{q(\theta|\gamma)}[\log \theta_k] = \Psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) \tag{20.50}$$

therefore,

$$\begin{aligned}
E_q[\log p(\theta|\alpha)] &= \log \Gamma \left( \sum_{l=1}^K \alpha_l \right) - \sum_{k=1}^K \log \Gamma(\alpha_k) \\
&+ \sum_{k=1}^K (\alpha_k - 1) \left[ \Psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) \right]
\end{aligned} \tag{20.51}$$

where  $\alpha_k$  and  $\gamma_k$  represent the Dirichlet distribution parameters of the k-th topic.

The second derivation is to find  $E_q[\log p(\mathbf{z}|\theta)]$ , which is about the distribution  $q(\theta, \mathbf{z}|\gamma, \eta)$  of mathematical expectation.

$$\begin{aligned}
E_q(\log p(\mathbf{z}|\theta)) &= \sum_{n=1}^N E_q[\log p(z_n|\theta)] \\
&= \sum_{n=1}^N E_{q(\theta, z_n|\gamma, \eta)}[\log p(z_n|\theta)] \\
&= \sum_{n=1}^N \sum_{k=1}^K q(z_{nk}|\eta) E_{q(\theta|\gamma)}[\log \theta_k] \\
&= \sum_{n=1}^N \sum_{k=1}^K \eta_{nk} \left[ \Psi(\gamma_k) - \Psi\left(\sum_{l=1}^K \gamma_l\right) \right]
\end{aligned} \tag{20.52}$$

where  $\eta_{nk}$  represents the probability that the word at the  $n$ -th position of the document is generated by the  $k$ -th topic, and  $k$  represents the Dirichlet distribution parameter of the  $k$ -th topic. The last step uses the formula of Appendix E (E.4).

The third derivation is to find  $E_q[\log p(\mathbf{w}|\mathbf{z}, \varphi)]$ , which is the mathematical expectation of the distribution  $q(\theta, \mathbf{z}|\gamma, \eta)$ .

$$\begin{aligned}
E_q(\log p(\mathbf{w}|\mathbf{z}, \varphi)) &= \sum_{n=1}^N E_q[\log p(w_n|z_n, \varphi)] \\
&= \sum_{n=1}^N E_{q(z_n|\eta)}[\log p(w_n|z_n, \varphi)] \\
&= \sum_{n=1}^N \sum_{k=1}^K q(z_{nk}|\eta) \log p(w_n|z_n, \varphi) \\
&= \sum_{n=1}^N \sum_{k=1}^K \sum_{v=1}^V \eta_{nk} w_n^v \log \varphi_{kv}
\end{aligned} \tag{20.53}$$

where  $\eta_{nk}$  represents the probability that the word at the  $n$ -th position of the document is generated by the  $k$ -th topic,  $w_n^v$  when the word at the  $n$ -th position is the  $v$ -th word of the word set, the value is 1; otherwise, the value is 0.  $\varphi_{kv}$  represents the probability of generating the  $v$ -th word in the word set for the  $k$ -th topic.

The fourth derivation solves for  $E_q(\log q(\theta|\gamma))$ , which is the mathematical expectation of the distribution  $q(\theta, \mathbf{z}|\gamma, \eta)$ . Since  $\theta \sim Dir(\gamma)$ , a similar Formula (20.50) can be obtained

$$\begin{aligned}
E_q[\log q(\theta|\gamma)] &= \log \Gamma\left(\sum_{l=1}^K \gamma_l\right) - \sum_{k=1}^K \log \Gamma(\gamma_k) \\
&\quad + \sum_{k=1}^K (\gamma_k - 1) \left[ \psi(\gamma_k) - \psi\left(\sum_{l=1}^K \gamma_l\right) \right]
\end{aligned} \tag{20.54}$$

where  $\gamma_k$  represents the Dirichlet distribution parameter of the  $k$ -th topic.

The fifth formula derivation obtains  $E_q[\log q(z|\eta)]$ , which is about the mathematical expectation of distribution  $q(\theta, z|\gamma, \eta)$ .

$$\begin{aligned}
E_q[\log q(z|\eta)] &= \sum_{n=1}^N E_q[\log q(z_n|\eta)] \\
&= \sum_{n=1}^N E_{q(z_n|\eta)}[\log q(z_n|\eta)] \\
&= \sum_{n=1}^N \sum_{k=1}^K q(z_{nk}|\eta) \log q(z_{nk}|\eta) \\
&= \sum_{n=1}^N \sum_{k=1}^K \eta_{nk} \log \eta_{nk}
\end{aligned} \tag{20.55}$$

where  $\eta_{nk}$  is the probability that the word in the  $n$ -th position of the document is generated by the  $k$ -th topic,  $\gamma_k$  is the Dirichlet distribution parameter of the  $k$ -th topic.

#### 20.4.3.2 Estimation of Variational Parameters $\gamma$ and $\eta$

Firstly, the parameter  $\eta$  is optimally estimated by the lower bound of evidence.  $\eta_{nk}$  is the probability that the word in the  $n$ -th position is generated by the  $k$ -th topic. Consider the maximization of  $\eta_{nk}$  in Eq. (20.47),  $\eta_{nk}$  satisfies the constraint  $\sum_{l=1}^K \eta_{nl} = 1$ . The Lagrange function of constrained optimization problem with  $\eta_{nk}$  is

$$L_{[\eta_{nk}]} = \eta_{nk} \left[ \psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) \right] + \eta_{nk} \log \varphi_{kv} - \eta_{nk} \log \eta_{nk} + \lambda_n \left( \sum_{l=1}^K n_{nl} - 1 \right) \tag{20.56}$$

Here  $\varphi_{kv}$  is the probability (at the  $n$ -th position) that the  $k$ -th topic generates the  $v$ -th word.

To get the partial derivative of  $\eta_{nk}$

$$\frac{\partial L}{\partial \eta_{nk}} = \psi(\gamma_k) - \Psi \left( \sum_{l=1}^K \gamma_l \right) + \log \varphi_{kv} - \log \eta_{nk} - 1 + \lambda_n \tag{20.57}$$

Let the partial derivative be zero, and the estimated value of the parameter  $\eta_{nk}$  is obtained

$$\eta_{nk} \propto \varphi_{kv} \exp\left(\Psi(\gamma_k) - \Psi\left(\sum_{l=1}^K \gamma_l\right)\right) \quad (20.58)$$

Then the parameter  $\gamma$  is optimized by the lower bound of evidence.  $\gamma_k$  is the Dirichlet distribution parameter of the  $k$ -th topic. Consider the maximization of Eq. (20.47) on  $\gamma_k$ .

$$\begin{aligned} L_{[\gamma_k]} = & \sum_{k=1}^K (\alpha_k - 1) \left[ \Psi(\gamma_k) - \Psi\left(\sum_{l=1}^K \gamma_l\right) \right] + \sum_{n=1}^N \sum_{k=1}^K \eta_{nk} \left[ \Psi(\gamma_k) - \Psi\left(\sum_{l=1}^K \gamma_l\right) \right] \\ & - \log \Gamma\left(\sum_{l=1}^K \gamma_l\right) + \log \Gamma(\gamma_k) - \sum_{k=1}^K (\gamma_k - 1) \left[ \Psi(\gamma_k) - \Psi\left(\sum_{l=1}^K \gamma_l\right) \right] \end{aligned} \quad (20.59)$$

Simplified to

$$\begin{aligned} L_{[\gamma_k]} = & \sum_{k=1}^K \left[ \Psi(\gamma_k) - \Psi\left(\sum_{l=1}^K \gamma_l\right) \right] \left( \alpha_k + \sum_{n=1}^N \eta_{nk} - \gamma_k \right) \\ & - \log \Gamma\left(\sum_{l=1}^K \gamma_l\right) + \log \Gamma(\gamma_k) \end{aligned} \quad (20.60)$$

Take the partial derivative of  $\gamma_k$  to get.

$$\frac{\partial L}{\partial \gamma_k} = \left[ \Psi'(\gamma_k) - \Psi'\left(\sum_{l=1}^K \gamma_l\right) \right] \left( \alpha_k + \sum_{n=1}^N \eta_{nk} - \gamma_k \right) \quad (20.61)$$

Let the partial derivative be zero, and solve for the estimated value of the parameter  $\gamma_k$

$$\gamma_k = \alpha_k + \sum_{n=1}^N \eta_{nk} \quad (20.62)$$

Based on this, a method for estimating the variational parameters by the coordinate ascent algorithm is obtained. The specific algorithm is as follows.

#### **Algorithm 20.4 (LDA's variational parameter estimation algorithm)<sup>3/4</sup>**

1. Initialization: for all  $k$  and  $n$ ,  $\eta_{nk}^{(0)} = 1/K$
2. Initialization: for all  $k$ ,  $\gamma_k = \alpha_k + N/K$
3. Repeat
4.     For  $n = 1$  to  $N$
5.         For  $k = 1$  to  $K$

6.  $\eta_{nk}^{(t+1)} = \varphi_{kv} \exp \left[ \Psi(\gamma_k^{(t)}) - \Psi \left( \sum_{l=1}^K \gamma_l^{(t)} \right) \right]$
7. Normalize  $\eta_{nk}^{(t+1)}$  so that the sum is 1
8.  $\gamma^{(t+1)} = \alpha + \sum_{n=1}^N \eta_n^{(t+1)}$
9. Until convergence

#### 20.4.3.3 Estimation of Model Parameters $\alpha$ and $\varphi$

Given a text set  $D = \{w_1, \dots, w_m, \dots, w_M\}$  the model parameter estimation is performed on all texts at the same time.

First, estimate  $\varphi$  by the maximum estimation of the lower bound of the evidence.  $\varphi_{kv}$  denotes the probability that the  $k$ -th topic generates the  $v$ -th word in the word set. Extend formula (20.47) to all texts, and consider the maximization of  $\varphi$ . Satisfy  $K$  constraints

$$\sum_{v=1}^V \varphi_{kv} = 1, \quad k = 1, 2, \dots, K$$

The Lagrangian function of the constrained optimization problem is

$$L_{[\beta]} = \sum_{m=1}^M \sum_{n=1}^{N_m} \sum_{k=1}^K \sum_{v=1}^V \eta_{mnk} w_{mn}^v \log \varphi_{kv} + \sum_{k=1}^K \lambda_k \left( \sum_{v=1}^V \varphi_{kv} - 1 \right) \quad (20.63)$$

Calculate the partial derivative of  $\varphi_{kv}$  and set it to zero, normalize the solution to obtain the estimated value of the parameter  $\varphi_{kv}$ .

$$\varphi_{kv} = \sum_{m=1}^M \sum_{n=1}^{N_m} \eta_{mnk} w_{mn}^v \quad (20.64)$$

where  $\eta_{mnk}$  is the probability that the  $n$ th word of the  $m$ -th text belongs to the  $k$ -th topic, and  $w_{mn}^v$  takes the value 1 when the  $n$ th word of the  $m$ -th text is the  $v$ -th word of the word set, otherwise it is 0.

Then estimate the parameter  $\alpha$  by maximizing the lower bound of evidence.  $\alpha_k$  represents the Dirichlet distribution parameter of the  $k$ th topic. Extend formula (20.47) to all texts and consider the maximization of  $\alpha$ .

$$\begin{aligned} L_{[\alpha]} = & \sum_{m=1}^M \left\{ \log \Gamma \left( \sum_{l=1}^K \alpha_l \right) - \sum_{k=1}^K \log \Gamma(\alpha_k) \right. \\ & \left. + \sum_{k=1}^K (\alpha_k - 1) \left[ \psi(\gamma_{mk}) - \psi \left( \sum_{l=1}^K \gamma_{ml} \right) \right] \right\} \end{aligned} \quad (20.65)$$

Take the partial derivative of  $\alpha_k$  to get

$$\frac{\partial L}{\partial \alpha_k} = M \left[ \psi \left( \sum_{l=1}^K \alpha_l \right) - \psi(\alpha_k) \right] + \sum_{m=1}^M \left[ \psi(\gamma_{mk}) - \psi \left( \sum_{l=1}^K \gamma_{ml} \right) \right] \quad (20.66)$$

Then take the partial derivative of  $\alpha_l$  to get

$$\frac{\partial^2 L}{\partial \alpha_k \partial \alpha_l} = M \left[ \psi \left( \sum_{l=1}^K \alpha_l \right) - \delta(k, l) \psi(\alpha_k) \right] \quad (20.67)$$

where  $\delta(k, l)$  is the delta function.

Equations (20.65) and (20.66) are the gradient  $g(\alpha)$  and Hessian matrix  $H(\alpha)$  of the function (20.64) to the variable  $\alpha$ , respectively. Apply Newton's method (also called Newton–Raphson method) to maximize the function.<sup>3</sup> Iterate with the following formula to get the estimated value of parameter  $\alpha$ .

$$\alpha_{\text{new}} = \alpha_{\text{old}} - H(\alpha_{\text{old}})^{-1} g(\alpha_{\text{old}}) \quad (20.68)$$

Based on this, an algorithm for estimating the parameter  $\alpha$  is obtained.

#### 20.4.4 Algorithm Summary

According to the above derivation, the variational EM algorithm of LDA is given.

**Algorithm 20.5** (*the Variational EM Algorithm of LDA*)

Input: Given text set  $D = \{\mathbf{w}_1, \dots, \mathbf{w}_m, \dots, \mathbf{w}_M\}$ ;

Output: Variational parameters  $\gamma, \eta$ , model parameters  $\alpha, \varphi$ .

Iterate E steps and M steps alternately until convergence.

(1) E-step

Fix the model parameters  $\alpha, \varphi$ , and estimate the variational parameters  $\gamma, \eta$  by maximizing the lower bound of the evidence about the variational parameters  $\gamma, \eta$ , see Algorithm 20.4 for details.

(2) M-step

Fixed variational parameters  $\gamma, \eta$ , by maximizing the lower bound of the evidence about the model parameters  $\alpha, \varphi$ , estimate the model parameters  $\alpha, \varphi$ . The specific algorithm is shown in Eqs. (20.63) and (20.67).

According to the variational parameters  $(\gamma, \eta)$ , the model parameters  $\theta = (\theta_1, \dots, \theta_m, \dots, \theta_M)$ ,  $\mathbf{z} = (z_1, \dots, z_m, \dots, z_M)$  can be estimated.

---

<sup>3</sup> Refer to Appendix B for the introduction of Newton's method.

The above is the variational EM algorithm of the simplified LDA model in Fig. 20.7, and the variational EM algorithm of the complete LDA model in Fig. 20.4 can be similarly derived as a generalization.

## Summary

1. The probability density function of the Dirichlet distribution is

$$p(\theta|\alpha) = \frac{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k \theta_i^{\alpha_i-1}$$

where  $\sum_{i=1}^k \theta_i = 1$ ,  $\theta_i \geq 0$ ,  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ ,  $\alpha_k > 0$ ,  $i = 1, 2, \dots, k$ . The Dirichlet distribution is a conjugate prior of multinomial distribution.

2. Latent Dirichlet assignment (LDA) is a probabilistic model for generating text sets. The model assumes that the topic is represented by the multinomial distribution of words, and the text is represented by the multinomial distribution of topics. The LDA model belongs to the probability graph model, which can be represented by plate representation. In the LDA model, the word distribution of each topic, the topic distribution of each text and the topic in each position of the text are hidden variables, and the words in each position of the text are observed variables.
3. The generation process of LDA generated text set is as follows:
  - (1) Word distribution of topic: the word distribution of all topics is generated randomly. The word distribution of topic is multinomial distribution, and its prior distribution is Dirichlet distribution.
  - (2) Topic distribution of text: the topic distribution of all texts is generated randomly. The topic distribution of text is multinomial distribution, and its prior distribution is Dirichlet distribution.
  - (3) Content of text: randomly generate the content of all text. In each position of each text, a topic is randomly generated according to the topic distribution of the text, and then a word is randomly generated according to the word distribution of the topic.
4. The learning and reasoning of LDA model cannot be solved directly. The commonly used methods are Gibbs sampling algorithm and variational EM algorithm, the former is Monte Carlo method, the latter is approximate algorithm.
5. The basic idea of LDA shrinkage Gibbs sampling algorithm is as follows. The goal is to estimate the joint probability distribution  $p(w, z, \theta, \varphi|\alpha, \beta)$ . The hidden variables  $\theta$  and  $\mu$  are eliminated by integral summation, and the marginal probability distribution  $p(w, z|\alpha, \beta)$  is obtained; the random samples of distribution  $p(w, z|\alpha, \beta)$  are obtained by Gibbs sampling of probability distribution  $p(w, z|\alpha, \beta)$ ; then we use the sample to analyze the variable  $Z$ ,  $\theta$  and  $\mu$ . Finally, the parameter estimation of  $p(w, z, \theta, \varphi|\alpha, \beta)$  of LDA model is obtained. The

specific algorithm is as follows. For a given text word sequence, each position is randomly assigned a topic to form a whole topic series. Then loop through the following. The whole text sequence is scanned, and the full conditional probability distribution of the topic in each position is calculated, and then the new topic in this position is obtained by random sampling, and assigned to this position.

6. The basic idea of variational reasoning is as follows. Suppose the model is a joint probability distribution  $p(x, z)$ , where  $x$  is the observed variable (data) and  $z$  is the hidden variable. The goal is to learn the posterior probability distribution  $p(z|x)$  of the model. In this paper, we use the variational distribution  $q(z)$  to approximate the conditional probability distribution  $p(z|x)$ . Using KL divergence to calculate the similarity between them, we find the nearest  $q^*(z)$  to  $p(z|x)$  in the sense of KL divergence, and use this distribution to approximate  $p(z|x)$ . Suppose that all components of  $z$  in  $q(z)$  are independent of each other. Using Jensen inequality, it is found that the minimization of KL divergence can be achieved by maximizing the lower bound of evidence. Therefore, variational reasoning becomes to solve the problem of maximizing the lower bound of evidence:

$$L(q, \theta) = E_q[\log p(x, z|\theta) - E_q[\log q(z)]]$$

7. The variational EM algorithm of LDA is as follows. For LDA model, the variational distribution is defined and the variational EM algorithm is applied. The goal is to maximize the lower bound of evidence  $L(\gamma, \eta, \alpha, \varphi)$ , where  $\alpha$  and  $\varphi$  are model parameters,  $\gamma$  and  $\eta$  are variational parameters. Iterating step  $E$  and step  $M$  alternately until convergence.
  - (1) E-step: Fix the model parameters  $\alpha$  and  $\varphi$ , and estimate the variational parameters  $\gamma$  and  $\eta$  by maximizing the lower bound of the evidence about the variational parameters  $\gamma$  and  $\eta$ .
  - (2) E-step: Fix the variational parameters  $\gamma$  and  $\eta$ , and estimate the model parameters  $\alpha$  and  $\varphi$  by maximizing the lower bound of the evidence about the model parameters  $\alpha$  and  $\varphi$ .

## Further Readings

The original paper on LDA is in the literature [1, 2], the Gibbs sampling algorithm of LDA is in the literature [3–5], and the variational EM algorithm is in the literature [2]. The introduction of variational reasoning can refer to the literature [6]. The distributed learning algorithm of LDA is available in the literature [7], the fast learning algorithm is available in the literature [8], and the online learning algorithm is available in the literature [9].

## Exercise

- 20.1 Derive the mathematical expectation formula of Dirichlet distribution.
- 20.2 For the text example of 17.2.2, use the LDA model for topic analysis.

- 20.3 Find out the part of the Dirichlet distribution used in the Gibbs sampling algorithm and the variational EM algorithm of LDA, and consider the importance of using the Dirichlet distribution in LDA.
- 20.4 Give the algorithm complexity of the Gibbs sampling algorithm and the variational EM algorithm of LDA.
- 20.5 Prove that the variational EM algorithm converges.

## References

1. Blei DM, Ng AY, Jordan MI. Latent Dirichlet allocation. In: Advances in neural information processing systems, vol. 14. The MIT Press; 2002.
2. Blei DM, Ng AY, Jordan MI. Latent Dirichlet allocation. J Mach Learn Res. 2003;3:933–1022.
3. Griffiths TL, Steyvers M. Finding scientific topics. Proceedings of the National Academy of Science. 2004;101:5228–35.
4. Steyvers M, Griffiths T. Probabilistic topic models. In: Landauer T, McNamara D, Dennis S et al., editors. Handbook of latent semantic analysis. Psychology Press; 2014.
5. Heinrich G. Parameter estimation for text analysis. Technical Note; 2004.
6. Blei DM, Kucukelbir A, McAuliffe JD. Variational inference: a review for statisticians. J Am Stat Assoc. 2017; 112(518).
7. Newman D, Smyth P, Welling M, Asuncion AU. Distributed inference for latent Dirichlet allocation. In: Advances in neural information processing systems; 2008. p. 1081–88.
8. Porteous I, Newman D, Ihler A, et al. Fast collapsed Gibbs sampling for latent Dirichlet allocation. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining; 2008. p. 569–77.
9. Hoffman M, Bach FR, Blei DM. Online learning for latent Dirichlet allocation. In: Advances in neural information processing systems; 2010. p. 856–64.

# Chapter 21

## The PageRank Algorithm

In practical applications, a lot of data exist in the form of a graph. For example, the Internet and social network can be regarded as a graph. Machine learning on graph data is therefore of great significance in theory and application. The PageRank algorithm is a representative algorithm for graph link analysis, it belongs to the unsupervised learning method on graph data.

The PageRank algorithm was originally proposed by Page and Brin in 1996 as a method for computing the importance of web pages on the Internet and was used in the web page ranking of the Google search engine. PageRank can be defined on any directed graph, and it has been applied to a variety of problems such as social influence analysis and text summarization.

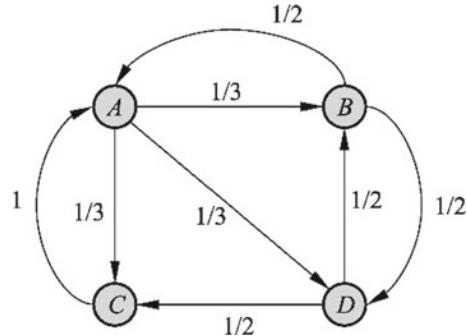
The basic idea of the PageRank algorithm is to define a random walk model on the directed graph, i.e., a first-order Markov chain, which describes the behavior of random walkers randomly accessing each node along with the directed graph. Under certain conditions, the probability of access to each node in the limit case converges to the stationary distribution. Then the stationary probability value of each node is its PageRank value, which indicates the significance of the node. PageRank is defined recursively, and the computation of PageRank can be carried out by an iterative algorithm.

Section 21.1 of this chapter defines PageRank, and Sect. 21.2 describes the computation method of PageRank, including the commonly used power method.

### 21.1 The Definition of PageRank

#### 21.1.1 Basic Ideas

The PageRank algorithm was proposed to compute the importance of Internet pages. PageRank is a function defined on a set of web pages. It gives a positive real number for each web page, indicating the importance of the web page, and then constitutes

**Fig. 21.1** Directed graph

a vector as a whole. The higher the PageRank value is, the more important the web page is, and it is more likely to be ranked higher in the ranking of Internet searches.<sup>1</sup>

Assuming that the Internet is a directed graph (or digraph), based on which a random walk model, i.e., a first-order Markov chain, is defined to represent the process of random browsing web pages on the Internet. Assume that the user jumps to the next web page with an equal probability according to the hyperlinks in each web page and makes such random jumps continuously on the Internet, which forms a first-order Markov chain. PageRank represents the stationary distribution of the Markov chain. The PageRank value of each web page is the stationary probability.

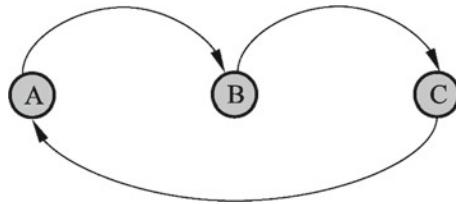
Figure 21.1 shows a directed graph. Suppose it is a simplified Internet, where nodes  $A$ ,  $B$ ,  $C$ , and  $D$  represent web pages. The directed edges between nodes represent hyperlinks between web pages, and the weights on the edges represent the probability of random hops between web pages. Suppose there is a visitor who walks randomly on the Internet. If the visitor is on page  $A$ , he will move to pages  $B$ ,  $C$ , and  $D$  at the next step with a probability of  $1/2$ . If the visitor is on Web page  $B$ , the probability of jumping to web pages  $A$  and  $D$  at the next step is  $1/2$ . If the visitor is browsing page  $C$ , the probability of moving to page  $A$  is 1. If the visitor is on page  $D$ , the next step will shift to pages  $B$  and  $C$  with a probability of  $1/2$ .

Intuitively, the more hyperlinks a web page has, the higher the probability of randomly jumping to that page, the higher the PageRank value of that page, and the more important that page is. The higher the PageRank value of a web page, the higher the probability of randomly jumping to that web page, the higher the PageRank value of that web page, and the more important that web page will be. The PageRank value depends on the topology of the network, and once the network's topology (connectivity) is determined, the PageRank value is determined.

The PageRank computation, usually an iterative process, can be performed on a directed graph of the Internet. An initial distribution is first assumed, and the PageRank values of all pages are computed iteratively until they converge.

<sup>1</sup> Besides the importance of web pages, the ranking of web pages in search engines is also determined by how well the page matches the query. In the Internet search, the PageRank of a web page is independent of the query and can be computed offline in advance and added to the web page index.

**Fig. 21.2** Periodic directed graph



In the following, we first define a directed graph and random walk model on a directed graph and then give the basic and general definitions of PageRank. The basic definition corresponds to the ideal case, and the general one corresponds to the practical case.

## 21.1.2 The Directed Graph and Random Walk Model

### 21.1.2.1 Directed Graph

**Definition 21.1** (*Directed graph*) A directed graph is denoted as  $G = (V, E)$ , where  $V$  and  $E$  denote the set of nodes and directed edges.

For example, the Internet can be regarded as a directed graph, each web page is a node of the directed graph, and each hyperlink between web pages is an edge of the directed graph.

From one node to another, a sequence of edges is called a path, and the number of edges on the path is called the length of the path. If a directed graph starts from any one of its nodes and reaches any other node, it is called a strongly connected graph. The directed graph in Fig. 21.1 is a strongly connected graph.

Suppose  $k$  is a natural number greater than 1. If the length of the path from a node of a digraph to this node is a multiple of  $k$ , then this node is called a periodic node. If a directed graph does not contain periodic nodes, it is called an aperiodic graph, otherwise, it is periodic.

Figure 21.2 is an example of a periodic digraph. Starting from node  $A$  and returning to  $A$ , we must go through the path  $A-B-C-A$ . the length of all possible paths is a multiple of 3, so node  $A$  is a periodic node. This directed graph is a periodic graph.

### 21.1.2.2 Random Walk Model

**Definition 21.2** (*Random walk model*) Given a directed graph with  $n$  nodes, a random walk model, i.e. first-order Markov chain,<sup>2</sup> is defined on the directed graph, where nodes represent states and directed edges represent transitions between states. It

---

<sup>2</sup> The introduction of Markov chain can refer to Chap. 19.

is assumed that the transition probabilities from a node to all nodes connected by directed edges are equal. Specifically, the transition matrix is an  $n$ -order matrix  $M$ .

$$M = [m_{ij}]_{n \times n} \quad (21.1)$$

The value rule of element  $m_{ij}$  in the  $i$ -th row and  $j$ -th column is as follows: if node  $j$  has  $k$  directed edges connected, and node  $i$  is a connected node, then  $m_{ij} = \frac{1}{k}$ ; Otherwise,  $m_{ij} = 0, i, j = 1, 2, \dots, n$ .

Note that the transfer matrix has some properties:

$$m_{ij} \geq 0 \quad (21.2)$$

$$\sum_{i=1}^n m_{ij} = 1 \quad (21.3)$$

each element is non-negative and the sum of the elements in each column is 1. In other words, the matrix  $M$  is a stochastic matrix.

The random walk on a directed graph forms a Markov chain, where the state shift of the random walker occurs every unit of time. If at the current moment, it is at the  $j$ -th node (state), then the probability that it shifts to the  $i$ -th node (state) next moment is  $m_{ij}$ , which only depends on the current state. It is independent of the past and has the Markov property.

The random walk model can be defined on the directed graph in Fig. 21.1. There are directed edges from node  $A$  to nodes  $B$ ,  $C$  and  $D$ , which enables the shift from  $A$  to  $B$ ,  $C$ , and  $D$  with a probability of  $1/3$ . However, the possibility of the transition to  $A$  is 0. Thus, the first column of the transition matrix can be written. There are directed edges from node  $B$  to nodes  $A$  and  $D$ . The probabilities that the random walker shifts from  $B$  to  $A$  and  $D$  are  $1/2$ , while the probability of shifting to  $B$  and  $C$  is 0. Then column 2 of the matrix can be written, and so on. The transfer matrix is thus obtained.

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

The probability distribution of the random walker visiting each node at a certain time  $t$  is the state distribution of the Markov chain at time  $t$ , which can be represented by an  $n$ -dimensional column vector  $R_t$ . Then the probability distribution of the random walk visiting each node at time  $t + 1$  meets the requirements of  $R_{t+1}$

$$R_{t+1} = M R_t \quad (21.4)$$

### 21.1.3 The Basic Definition of PageRank

Given a strongly connected and aperiodic directed graph with  $n$  nodes, a random walk model is defined based on it. Assume that the transition matrix is  $M$ , and the probability distribution of accessing each node at time  $0, 1, 2, \dots, t, \dots$  is

$$R_0, MR_0, M^2R_0, \dots, M^tR_0, \dots$$

The limit

$$\lim_{t \rightarrow \infty} M^t R_0 = R \quad (21.5)$$

exists, the limit vector  $R$  represents the stationary distribution of the Markov chain, which satisfies

$$MR = R$$

**Definition 21.3** (*The basic definition of PageRank*) Given a strongly connected and aperiodic directed graph containing  $n$  nodes  $v_1, v_2, \dots, v_n$ , a random walk model is defined on the directed graph, namely a first-order Markov chain. The characteristic of random walk is that the transition probability from one node to all nodes connected with a directed edge is equal, and the transition matrix is  $M$ .

The characteristic of random walk is that the transition probability from a node to all nodes connected by a directed edge is equal, and the transition matrix is  $M$ . This Markov chain has a stationary distribution  $R$

$$MR = R \quad (21.6)$$

The stationary distribution  $R$  is called the PageRank of this directed graph. Each component of  $R$  is called the PageRank value of each node.

$$R = \begin{bmatrix} PR(v_1) \\ PR(v_2) \\ \vdots \\ PR(v_n) \end{bmatrix}$$

where  $PR(v_i), i = 1, 2, \dots, n$ , represents the PageRank value of node  $v_i$ .

Apparently

$$PR(v_i) \geq 0, \quad i = 1, 2, \dots, n \quad (21.7)$$

$$\sum_{i=1}^n PR(v_i) = 1 \quad (21.8)$$

$$PR(v_i) = \sum_{v_j \in M(v_i)} \frac{PR(v_j)}{L(v_j)}, \quad i = 1, 2, \dots, n \quad (21.9)$$

Here  $M(v_i)$  represents the set of nodes pointing to node  $v_i$ , and  $L(v_j)$  represents the number of directed edges connected to node  $v_j$ .

The basic definition of PageRank is an ideal situation. In this case, PageRank exists, and the PageRank value can be obtained through continuous iteration.

**Theorem 21.1** *An irreducible and aperiodic finite-state Markov chain has a unique stationary distribution, and the state distribution converges to the unique stationary distribution when time goes to infinity.*

According to the Markov chain stationary distribution theorem, the random walk model (Markov chain) is a strongly connected and aperiodic directed graph, when the time tends to infinity, the state distribution converges to a unique smooth distribution.

**Example 21.1** Given the directed graph in Fig. 21.1, find the PageRank of the graph.<sup>3</sup>

Solution transition matrix

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

Take the initial distribution vector  $R_0$  as

$$R_0 = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

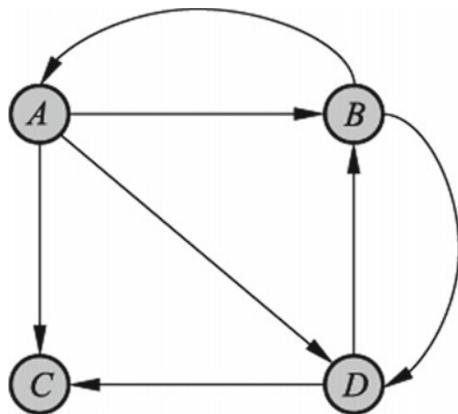
The initial vector  $R_0$  is multiplied by the transition matrix  $M$  to get the vector sequence

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}, \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix}, \dots, \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

Finally, the limit vector is obtained

---

<sup>3</sup> Examples 21.1 and 21.2 are from the literature [2].

**Fig. 21.3** Directed graph

$$R = \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

i.e., the PageRank value of the directed graph.

A general digraph may not satisfy the strongly connected and aperiodic conditions. For example, on the Internet, most web pages have no hyperlinks, that is to say, they can't jump to other web pages. So the basic definition of PageRank doesn't apply.

**Example 21.2** The edge from  $C$  to  $A$  is removed from the directed graph of Fig. 21.1, and the directed graph of Fig. 21.3 is obtained. In the directed graph of Fig. 21.3, node  $C$  is not connected with the edge.

The transition matrix  $M$  of the directed graph in Fig. 21.3 is

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

At this time,  $M$  is not a stochastic matrix, because the stochastic matrix requires the sum of the elements in each column to be 1, where the sum of the third column is 0, not 1.

If we still calculate the probability distribution of each node at each time, we will get the following result

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It can be seen that as time goes by, the probability of visiting each node becomes 0.

### 21.1.4 General Definition of PageRank

The general definition of PageRank is to introduce smoothing items based on the basic definition.

Given an arbitrary directed graph with  $n$  nodes  $v_i$ ,  $i = 1, 2, \dots, n$ , suppose we consider a random walk model on the graph, namely a first-order Markov chain, and its transition matrix is  $M$ , and the transition probability from a node to all the connected nodes is equal. This Markov chain may not have a stationary distribution. Suppose we consider another completely random walk model, the elements of the transition matrix are all  $1/n$ , that is to say, the transition probability from any node to any node is  $1/n$ . The linear combination of the two transition matrices constitutes a new transition matrix on which a new Markov chain can be defined. It is easy to prove that this Markov chain must have a stationary distribution, and the stationary distribution satisfies

$$R = dMR + \frac{1-d}{n}\mathbf{1} \quad (21.10)$$

In the formula,  $d(0 \leq d \leq 1)$  is the coefficient, called the damping factor,  $R$  is an  $n$ -dimensional vector and  $\mathbf{1}$  is an  $n$ -dimensional vector with all components being 1.  $R$  represents the general PageRank of the directed graph.

$$R = \begin{bmatrix} PR(v_1) \\ PR(v_2) \\ \vdots \\ PR(v_n) \end{bmatrix}$$

$PR(v_i)$ ,  $i = 1, 2, \dots, n$  represents the PageRank value of node  $v_i$ .

The first term in Eq. (21.10) represents the probability of visiting each node according to the transition matrix  $M$  (when the state distribution is a stationary distribution), and the second term represents the probability of completely random visiting each node. The value of the damping factor  $d$  is determined by experience, for example,  $d = 0.85$ . When  $d$  is close to 1, the random walk is mainly performed

according to the transition matrix  $M$ ; when  $d$  is close to 0, the random walk mainly randomly visits each node with equal probability.

The PageRank of each node can be written by Eq. (21.10), which is the general definition of PageRank.

$$PR(v_i) = d \left( \sum_{v_j \in M(v_i)} \frac{PR(v_j)}{L(v_j)} \right) + \frac{1-d}{n}, \quad i = 1, 2, \dots, n \quad (21.11)$$

Here  $M(v_i)$  is the set of nodes pointing to node  $v_i$ , and  $L(v_j)$  is the number of edges connected to node  $v_j$ .

The second term is called the smoothing term. Due to the smoothing term, the PageRank value of all nodes will not be 0 and has the following properties:

$$PR(v_i) > 0, \quad i = 1, 2, \dots, n \quad (21.12)$$

$$\sum_{i=1}^n PR(v_i) = 1 \quad (21.13)$$

The general definition of PageRank is given below.

**Definition 21.4** (*General Definition of PageRank*) Given an arbitrary directed graph with  $n$  nodes, define a general random walk model on the directed graph, that is, a first-order Markov chain. The transition matrix of the general random walk model consists of a linear combination of two parts. One part is the basic transition matrix  $M$  of the directed graph, which means that the transition probability from one node to all nodes connected to it is equal, and the other part is complete. The random transition matrix means that the transition probability from any node to any node is  $1/n$ , and the linear combination coefficient is the damping factor  $d(0 \leq d \leq 1)$ . This general random walk Markov chain has a stationary distribution, denoted as  $R$ . Define the stationary distribution vector  $R$  as the general PageRank of this directed graph.  $R$  is determined by the formula

$$R = dMR + \frac{1-d}{n}\mathbf{1} \quad (21.14)$$

Here the  $\mathbf{1}$  is an  $n$ -dimensional vector with all 1 component.

The general definition of PageRank means that Internet viewers randomly walk on the Internet according to the following method: On any web page, the viewer may decide to jump randomly according to the hyperlink with probability  $d$ , and then with equal probability from the connected hyperlink jump to the next webpage; or decide to jump completely randomly with probability  $(1 - d)$ , then jump to any webpage with equal probability  $1/n$ . The second mechanism ensures that web pages that are not connected to hyperlinks can also be jumped out. This can ensure a stationary

distribution, i.e., the existence of a general PageRank, so the general PageRank is suitable for any structure of the network.

## 21.2 Computation of PageRank

The definition of PageRank is constructive, i.e., the definition itself gives the algorithm. This section lists the computation methods of PageRank, including iterative algorithm, power method, and algebraic algorithm. The commonly used method is the power method.

### 21.2.1 Iterative Algorithm

Given a directed graph with  $n$  nodes, the transition matrix is  $M$ , the general PageRank of the directed graph is given by the limit vector  $R$  of the iterative formula

$$R_{t+1} = dMR_t + \frac{1-d}{n}\mathbf{1} \quad (21.15)$$

The iterative algorithm of PageRank is to iterate according to this general definition until it converges.

#### Algorithm 21.1 (Iterative Algorithm of PageRank)

Input: a directed graph with  $n$  nodes, transition matrix  $M$ , damping factor  $d$ , initial vector  $R_0$ ;

Output: PageRank vector  $R$  of the directed graph.

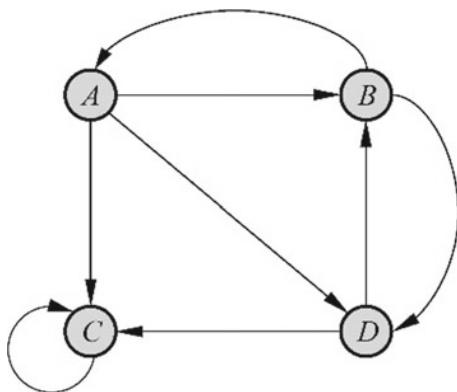
- (1) Let  $t = 0$
- (2) Compute

$$R_{t+1} = dMR_t + \frac{1-d}{n}\mathbf{1}$$

- (3) If  $R_{t+1}$  is sufficiently close to  $R_t$ , set  $R = R_{t+1}$  and stop the iteration.
- (4) Otherwise, let  $t = t + 1$ , and execute step (2).

**Example 21.3** Given the directed graph shown in Fig. 21.4, take  $d = 0.8$  to find the PageRank of the graph.

**Solution** From Fig. 21.4, the transition matrix is

**Fig. 21.4** Directed graph

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

Compute according to Formula (21.15)

$$dM = \frac{4}{5} \times \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix}$$

$$\frac{1-d}{n} \mathbf{1} = \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

The iterative formula is

$$R_{t+1} = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} R_t + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

Let the initial vector

$$R_0 = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

Iterate

$$R_1 = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix} = \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix} = \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix}$$

etc. Finally get

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix}, \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix}, \begin{bmatrix} 543/4500 \\ 707/4500 \\ 2543/4500 \\ 707/4500 \end{bmatrix}, \dots, \begin{bmatrix} 15/148 \\ 19/148 \\ 95/148 \\ 19/148 \end{bmatrix}$$

The computation results show that the PageRank value of node  $C$  is more than half, and other nodes also have corresponding PageRank values.

### 21.2.2 Power Method

The power method is a commonly used PageRank computation method. The general PageRank of a directed graph is obtained by approximately computing the main eigenvalues and main eigenvectors of the matrix.

First, introduce the power method. The power method is mainly used to approximate the dominant eigenvalue and dominant eigenvector of the matrix. The dominant eigenvalue refers to the eigenvalue with the largest absolute value, and the dominant eigenvector is its corresponding eigenvector. Note that the eigenvector is not unique, but its direction is determined, multiplied by any coefficient or feature vector.

Assuming that the dominant eigenvalues and dominant eigenvectors of the  $n - order$  matrix  $A$  are required, the following steps are used.

First, take any initial  $n$ -dimensional vector  $x_0$  and construct an  $n$ -dimensional vector sequence as follows

$$x_0, x_1 = Ax_0, x_2 = Ax_1, \dots, x_k = Ax_{k-1}$$

Then, suppose that the matrix  $A$  has  $n$  eigenvalues, arranged according to the absolute value

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$$

The corresponding  $n$  linearly independent eigenvectors are

$$u_1, u_2, \dots, u_n$$

These  $n$  eigenvectors constitute a set of bases of  $n$ -dimensional space. Thus, the initial vector  $x_0$  can be expressed linear combination as  $u_1, u_2, \dots, u_n$ .

$$x_0 = a_1 u_1 + a_2 u_2 + \cdots + a_n u_n$$

we can obtain

$$\begin{aligned} x_1 &= Ax_0 = a_1 A u_1 + a_2 A u_2 + \cdots + a_n A u_n \\ &\vdots \\ x_k &= A^k x_0 = a_1 A^k u_1 + a_2 A^k u_2 + \cdots + a_n A^k u_n \\ &= a_1 \lambda_1^k u_1 + a_2 \lambda_2^k u_2 + \cdots + a_n \lambda_n^k u_n \end{aligned}$$

Next, the dominant eigenvalues of matrix  $A$  are assumed,  $\lambda_1$  is the single root of the characteristic equation

$$x_k = a_1 \lambda_1^k \left[ u_1 + \frac{a_2}{a_1} \left( \frac{\lambda_2}{\lambda_1} \right)^k u_2 + \cdots + \frac{a_n}{a_1} \left( \frac{\lambda_n}{\lambda_1} \right)^k u_n \right] \quad (21.16)$$

Because of  $|\lambda_1| > |\lambda_j|$ ,  $j = 2, \dots, n$ , when  $k$  is sufficiently large, there is

$$x_k = a_1 \lambda_1^k [u_1 + \varepsilon_k] \quad (21.17)$$

Here  $\varepsilon_k$  is the infinitesimal when  $k \rightarrow \infty$ ,  $\varepsilon_k \rightarrow 0$  ( $k \rightarrow \infty$ ). That is

$$x_k \rightarrow a_1 \lambda_1^k u_1 (k \rightarrow \infty) \quad (21.18)$$

When  $k$  is large enough, the difference between vector  $x_k$  and eigenvector  $u_1$  is only one coefficient. It can be seen from Formula (21.18),

$$\begin{aligned} x_k &\approx a_1 \lambda_1^k u_1 \\ x_{k+1} &\approx a_1 \lambda_1^{k+1} u_1 \end{aligned}$$

So the dominant eigenvalue  $\lambda_1$  can be expressed as

$$\lambda_1 \approx \frac{x_{k+1,j}}{x_{k,j}} \quad (21.19)$$

where  $x_{k,j}$  and  $x_{k+1,j}$  is the  $j$ -th component of  $x_k$  and  $x_{k+1}$  respectively.

In practical computations, in order to avoid situations where the absolute values are too large or too small, the normalization is usually performed after each iteration step by dividing the vector by its parametric number, i.e.

$$y_{t+1} = Ax_t \quad (21.20)$$

$$x_{t+1} = \frac{y_{t+1}}{\|y_{t+1}\|} \quad (21.21)$$

Here the norm is the infinite norm of the vector, i.e. the maximum of the absolute values of the components of the vector

$$\|x\|_\infty = \max\{|x_1|, |x_2|, \dots, |x_n|\}$$

Now back to computing the general PageRank.

The transfer matrix can be written

$$R = \left( dM + \frac{1-d}{n} \mathbf{E} \right) R = AR \quad (21.22)$$

where  $d$  is the damping factor and  $\mathbf{E}$  is a square matrix of order  $n$  with all elements of 1. According to the Perron-Frobenius theorem,<sup>4</sup> the vector  $R$  of the general PageRank is the dominant eigenvector of the matrix  $A$  with dominant eigenvalue 1. So the general PageRank can be approximated using the power method.

### **Algorithm 21.2 (Power method for computing general PageRank)**

Input: directed graph with  $n$  nodes, transfer matrix  $M$  of the directed graph, coefficients  $d$ , initial vector  $x_0$ , computation accuracy  $\varepsilon$ .

Output: PageRank  $R$  of the directed graph.

- (1) Let  $t = 0$ , choose the initial vector  $x_0$ .
- (2) Compute the general transfer matrix  $A$  of the directed graph

$$A = dM + \frac{1-d}{n} \mathbf{E}$$

- (3) Iterate and normalize the result vector

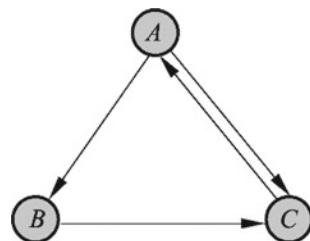
$$y_{t+1} = Ax_t$$

$$x_{t+1} = \frac{y_{t+1}}{\|y_{t+1}\|}$$

- (4) When  $\|x_{t+1} - x_t\| < \varepsilon$ , let  $R = x_t$  and stop the iteration.

---

<sup>4</sup> The form of the Perron-Frobenius theorem is more complicated and will not be described here.

**Fig. 21.5** Directed graph

- (5) Otherwise, let  $t = t + 1$  and execute step (3).
- (6) Normalize  $R$  so that it represents the probability distribution.

**Example 21.4** Given a directed graph shown as Fig. 21.5, take  $d = 0.85$ , find the general PageRank of the directed graph.

**Solution** Using the power method, according to Algorithm 21.2, compute the general PageRank of the directed graph.

The transfer matrix can be seen from Fig. 21.5

$$M = \begin{bmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{bmatrix}$$

- (1) Let  $t = 0$

$$x_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- (2) Compute the general transfer matrix  $A$  of directed graph

$$\begin{aligned} A &= dM + \frac{1-d}{n}E \\ &= 0.85 \times \begin{bmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{bmatrix} + \frac{0.15}{3} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.05 & 0.05 & 0.9 \\ 0.475 & 0.05 & 0.05 \\ 0.475 & 0.9 & 0.05 \end{bmatrix} \end{aligned}$$

- (3) Iteration and normalization

$$y_1 = Ax_0 = \begin{bmatrix} 1 \\ 0.575 \\ 1.425 \end{bmatrix}$$

$$x_1 = \frac{1}{1.425} \begin{bmatrix} 1 \\ 0.575 \\ 1.425 \end{bmatrix} = \begin{bmatrix} 0.7018 \\ 0.4035 \\ 1 \end{bmatrix}$$

$$y_2 = Ax_1 = \begin{bmatrix} 0.05 & 0.05 & 0.9 \\ 0.475 & 0.05 & 0.05 \\ 0.475 & 0.9 & 0.05 \end{bmatrix} \begin{bmatrix} 0.7018 \\ 0.4035 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.9553 \\ 0.4035 \\ 0.7465 \end{bmatrix}$$

$$x_2 = \frac{1}{0.9553} \begin{bmatrix} 0.9553 \\ 0.4035 \\ 0.7465 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.4224 \\ 0.7814 \end{bmatrix}$$

$$y_3 = Ax_2 = \begin{bmatrix} 0.05 & 0.05 & 0.9 \\ 0.475 & 0.05 & 0.05 \\ 0.475 & 0.9 & 0.05 \end{bmatrix} \begin{bmatrix} 1 \\ 0.4224 \\ 0.7814 \end{bmatrix} = \begin{bmatrix} 0.7744 \\ 0.5352 \\ 0.8943 \end{bmatrix}$$

$$x_3 = \frac{1}{0.8943} \begin{bmatrix} 0.7744 \\ 0.5352 \\ 0.8943 \end{bmatrix} = \begin{bmatrix} 0.8659 \\ 0.5985 \\ 1 \end{bmatrix}$$

In this way, the iterative normalization continues, and the vector sequence of  $x_t$ ,  $t = 0, 1, 2, \dots, 21, 22$ , is obtained

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.7018 \\ 0.4035 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.4224 \\ 0.7814 \end{bmatrix}, \begin{bmatrix} 0.8659 \\ 0.5985 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.9732 \\ 0.4912 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.5516 \\ 0.9807 \end{bmatrix}, \\ \begin{bmatrix} 0.9409 \\ 0.5405 \\ 1 \end{bmatrix}, \dots, \begin{bmatrix} 0.9760 \\ 0.5408 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.9755 \\ 0.5404 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.9761 \\ 0.5406 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.9756 \\ 0.5406 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.9758 \\ 0.5404 \\ 1 \end{bmatrix}$$

Assuming that the two vectors obtained later have met the computation accuracy requirements, then take

$$R = \begin{bmatrix} 0.9756 \\ 0.5406 \\ 1 \end{bmatrix}$$

That is, the general PageRank required is obtained. If the general PageRank is taken as a probability distribution and normalized so that the sum of the components is 1, then the corresponding general PageRank can be written as

$$R = \begin{bmatrix} 0.3877 \\ 0.2149 \\ 0.3974 \end{bmatrix}$$

### 21.2.3 Algebraic Algorithms

The algebraic algorithm computes the general PageRank of the directed graph through the inverse matrix computation of the general transition matrix.

According to the general PageRank definition (21.14)

$$R = dMR + \frac{1-d}{n}\mathbf{1}$$

Then,

$$(I - dM)R = \frac{1-d}{n}\mathbf{1} \quad (21.23)$$

$$R = (I - dM)^{-1} \frac{1-d}{n}\mathbf{1} \quad (21.24)$$

Here  $I$  is the identity matrix. When  $0 < d < 1$ , the solution of the linear equation system (21.23) exists and is unique. In this way, the inverse matrix  $(I - dM)^{-1}$  gets the general PageRank of the directed graph.

### Summary

- (1) PageRank is a computation method for the importance of Internet web pages, which can be defined and extended to the computation of the importance of any directed graph node. The basic idea is to define a random walk model on a directed graph, that is, a first-order Markov chain, which describes the behavior of the walker randomly visiting each node along with the directed graph. Under certain conditions, the probability of access to each node in the limit case converges to a stationary distribution. At this time, the probability value of each node is its PageRank value, which indicates the relative importance of the node.
- (2) A random surfer model can be defined on a directed graph, that is, a first-order Markov chain, in which nodes represent states and directed edges represent transitions between states. It is assumed that the transition probabilities from a node to all connected nodes are equal. The transition probability is represented by the transition matrix  $M$

$$M = [m_{ij}]_{n \times n}$$

The element  $m_{ij}$  in row  $i$  and column  $j$  represents the probability of jumping from node  $j$  to node  $i$ .

- (3) When a directed graph with  $n$  nodes is strongly connected and aperiodic, the random walk model defined on the basis of it, that is, the first-order Markov chain has a stationary distribution, and the stationary distribution vector  $R$  is called the PageRank of the directed graph. If matrix  $M$  is a transfer matrix of Markov chain, then vector  $R$  satisfies

$$MR = R$$

Each component of vector  $R$  is called the PageRank value of each node.

$$R = \begin{bmatrix} PR(v_1) \\ PR(v_2) \\ \vdots \\ PR(v_n) \end{bmatrix}$$

where  $PR(v_i)$ ,  $i = 1, 2, \dots, n$ , denotes the PageRank value of node  $v_i$ . This is the basic definition of PageRank.

- (4) The conditions of the basic definition of PageRank are often not satisfied in reality, so we extend it to get the general definition of PageRank. On any directed graph with  $n$  nodes, a random walk model, i.e. first order Markov chain, can be defined. The transition matrix consists of a linear combination of two parts, The transition probability from any node to any node is  $1/n$ . The Markov chain has a stationary distribution, and the stationary distribution vector  $R$  is called the general PageRank of the digraph

$$R = dMR + \frac{1-d}{n} \mathbf{1}$$

where  $d(0 \leq d \leq 1)$  is the damping factor and  $\mathbf{1}$  is the  $n$ -dimensional vector with all components of 1.

- (5) The Computation Methods of PageRank Include Iterative algorithm, Power method and Algebraic algorithm.

The equivalent of PageRank is written by power method as

$$R = \left( dM + \frac{1-d}{n} \mathbf{E} \right) R = AR$$

where  $d$  is the damping factor and  $\mathbf{E}$  is the  $n-th$  order square matrix with all elements of 1.

It can be seen that  $R$  is the dominant eigenvector of the general transition matrix  $A$ , that is, the eigenvector corresponding to the largest eigenvalue. The

power method is a method of computing the dominant eigenvalues and dominant eigenvectors of a matrix.

The steps are: select the initial vector  $x_0$ ; compute the general transition matrix  $A$ ; iterate and normalize the vector until convergence.

$$\begin{aligned}y_{t+1} &= Ax_t \\x_{t+1} &= \frac{y_{t+1}}{\|y_{t+1}\|}\end{aligned}$$

## Further Reading

PageRank's original paper is literature [1], and its detailed introduction can be found in the literature [2, 3]. There are textbooks that introduce the Markov process [4]. The same well-known link analysis algorithm as PageRank and the HITS algorithm [5] can find the hub and authority in the network. PageRank has many extensions and variations. The original PageRank is based on discrete-time Markov chains, and BrowseRank is based on the promotion of continuous-time Markov chains [6], which can better prevent web ranking fraud. Personalized PageRank is a personalized PageRank (Ref. [7]), Topic Sensitive PageRank is a topic-based PageRank (Ref. [8]), and TrustRank is a PageRank to prevent web ranking fraud (Ref. [9]).

## Exercise

- 21.1 Assume that the square matrix  $A$  is a stochastic matrix, i.e., each element is non-negative, and the sum of the elements in each column is 1. Prove that  $A^k$  is still a stochastic matrix, where  $k$  is a natural number.
- 21.2 Please verify that in Example 21.1, iterating with different initial distribution vector  $R_0$  still yields the same limit vector  $R$ , i.e., PageRank.
- 21.3 Prove that the Markov chain in the general definition of PageRank has a stationary distribution, i.e., Eq. (21.11) holds.
- 21.4 Prove that the maximum eigenvalue of the stochastic matrix is 1.

## References

1. Page L, Brin S, Motwani R, et al. The PageRank citation ranking: bringing order to the Web. Stanford University; 1999.
2. Leskovec J, Rajaraman A, Ullman JD. Mining of massive datasets. Cambridge University Press; 2014.
3. Liu B. Web data mining: exploring hyperlinks, contents, and usage data. Springer Science & Business Media; 2007.
4. Serfozo R. Basics of applied stochastic processes. Springer; 2009.
5. Kleinberg JM. Authoritative sources in a hyperlinked environment. J ACM (JACM). 1999;46(5):604–32.
6. Liu Y, Gao B, Liu TY, et al. BrowseRank: letting Web users vote for page importance. In: Proceedings of the 31st SIGIR conference. 2008. p. 451–8.

7. Jeh G, Widom J. Scaling personalized Web search. In: Proceedings of the 12th WWW conference. 2003. p. 271–9.
8. Haveliwala TH. Topic-sensitive PageRank. In: Proceedings of the 11th WWW conference. 2002. p. 517–26.
9. Gyöngyi Z, Garcia-Molina H, Pedersen J. Combating Web spam with TrustRank. In: Proceedings of VLDB conference. 2004. p. 576–87.

# Chapter 22

## A Summary of Unsupervised Learning Methods

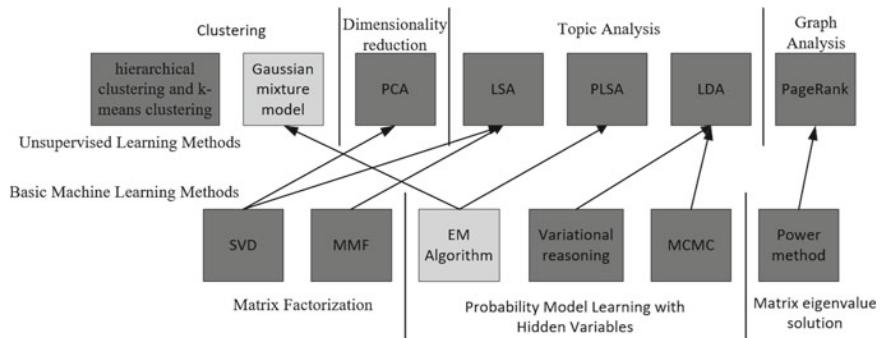
### 22.1 The Relationships and Characteristics of Unsupervised Learning Methods

The second part of this book introduces eight commonly used machine learning methods in detail, namely clustering methods (including hierarchical clustering and k-means clustering), Singular Value Decomposition (SVD), Principal Component Analysis (PCA), Latent Semantic Analysis (LSA), Probabilistic Latent Semantic Analysis (PLSA), Markov Chain Monte Carlo method (MCMC, including Metropolis–Hastings algorithm and Gibbs sampling), Latent Dirichlet Allocation (LDA), and PageRank algorithm. In addition, three other commonly used machine learning methods, namely Nonnegative Matrix Factorization (NMF), variational reasoning, and the power method, are briefly introduced in this part. These methods are usually for unsupervised learning clustering, dimensionality reduction, topic modeling, and graph analytics.

#### 22.1.1 *The Relationships Between Various Methods*

Figure 22.1 summarizes the relationships among some machine learning methods, including those presented in Part 1 and Part 2, shown in dark and light grey respectively. Methods placed in the upper part of the figure are unsupervised learning methods, and the lower part displays basic machine learning methods.

Unsupervised learning is for clustering, dimensionality reduction, topic modeling, and graph analytics. Clustering methods include hierarchical clustering,  $k$ -means clustering, and the Gaussian mixture model; methods for dimensionality reduction include PCA; topic modeling methods include LSA, PLSA, and LDA; methods for graph analysis include PageRank.



**Fig. 22.1** Relationship among machine learning methods

Basic machine learning methods do not involve specific machine learning models. They can not only be for unsupervised learning but also supervised and semi-supervised learning. Basic methods are divided into matrix decomposition, matrix eigenvalue solving, and probability model estimation with hidden variables. The first two are linear algebra problems, and the latter are probability and statistics problems. Methods of matrix decomposition include SVD and NMF. Methods for solving matrix eigenvalues include the power method and those for learning probability models with hidden variables cover the EM algorithm, variational reasoning, and MCMC.

### 22.1.2 *Unsupervised Learning Methods*

Clustering includes hard clustering and soft clustering, with hierarchical clustering and  $k$ -means clustering being the hard clustering methods. The Gaussian mixture model is a soft clustering method. Hierarchical clustering is based on the heuristic algorithm,  $k$ -means clustering is based on the iterative algorithm, and the Gaussian mixture model learning is usually based on an EM algorithm.

There are linear dimensionality reduction and nonlinear dimensionality reduction, and PCA is a linear dimensionality reduction method based on SVD.

Topic modeling has both clustering and dimensionality reduction features, including non-probabilistic and probabilistic models. LSA and NMF are non-probabilistic models, while PLSA and LDA are probabilistic models. PLSA does not assume that the model has a prior distribution, and its learning is based on maximum likelihood estimation. Based on Bayesian learning, LDA supposes that the model has the prior distribution, specifically performing posterior probability estimation. The LSA learning is based on SVD, and NMF can be used directly for topic modeling. PLSA learning is based on the EM algorithm, and LDA learning is based on Gibbs sampling or variational inference.

**Table 22.1** Characteristics of unsupervised learning methods

	Method	Model	Strategy	Algorithm
Clustering	Hierarchical clustering	Clustering tree	Minimum sample distance within the class	Heuristic algorithm
	$k$ -means clustering	$k$ -center clustering	Minimum distance between sample and class center	Iterative algorithms
	Gaussian mixture model	Gaussian mixture model	Maximum likelihood function	EM algorithm
Dimensionality reduction	PCA	Low-dimensional orthogonal space	Maximum variance	SVD
Topic modeling	LSA	Matrix factorization model	Minimum square loss	SVD
	NMF	Matrix factorization model	Minimum square loss	Nonnegative matrix factorization
	PLSA	PLSA model	Maximum likelihood function	EM algorithm
	LDA	LDA model	Posterior probability estimation	Gibbs sampling and variational inference
Graph analysis	PageRank	Markov Chain on a directed graph	Stationary distribution solution	Power method

One problem in graph analysis is link analysis, which is the computation of the nodes' importance. PageRank is a method of link analysis and is usually based on the power method.

Table 22.1 summarizes the models, strategies, and algorithms of unsupervised learning methods.

### 22.1.3 Basic Machine Learning Methods

Matrix decomposition is based on different assumptions: SVD is based on orthogonal assumption, i.e., the left and right matrices are orthogonal, and the middle matrix is nonnegative diagonal; nonnegative matrix factorization is based on nonnegative assumption, that is, the left and right matrices are nonnegative.

There are two approaches to learning probability models with hidden variables: the iterative computation method and the random sampling method. The EM algorithm and variational reasoning (including the variational EM algorithm) are iterative, and

**Table 22.2** Characteristics of learning methods with an hidden variable probability model

Algorithm	Basic principles	Astringency	Convergence rate	Difficulty of implementation	Applicability
EM algorithm	Iterative computation and posterior probability estimation	Convergence to local optimum	Faster	Easy	Simple model
Variational reasoning	Iterative computation and posterior probability estimation	Convergence to local optimum	Slower	More complex	Complex model
Gibbs sampling	Random sampling and posterior probability estimation	Convergence to the global optimum in probability	Slower	Easy	Complex model

Gibbs sampling is a random sampling method. The Variational EM algorithm is a generalization of the EM algorithm.

The power method is commonly used in solving eigenvalues and eigenvectors of matrices.

Table 22.2 summarizes the characteristics of the learning methods of probability models with hidden variables.

## 22.2 The Relationships and Characteristics of Topic Models

This book introduces four topic models, i.e., LSA, NMF, PLSA, and LDA. The former two of them are non-probabilistic models, and the latter two are probabilistic models. The relationship between them is discussed below (see the Refs. [1, 2] for details).

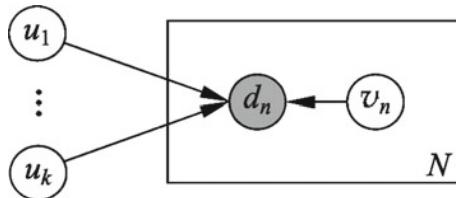
LSA, NMF and PLSA can be viewed in the context of a unified framework for matrix decomposition, in which a constrained matrix decomposition  $D = UV$  is performed by minimizing the generalized Bregman scatter to obtain the three topic models.

$$\min_{U,V} B(D||UV)$$

Here,  $B(D||UV)$  represents the generalized Bregman divergence between  $D$  and  $UV$ , and the value is 0 if and only if they are equal. The generalized Bregman divergence includes square loss, KL divergence, etc. The three topic models have three different specific forms. Table 22.3 shows the loss function and constraint

**Table 22.3** A matrix factorization perspective on topic models

Method	General loss function $B(D  UV)$	Constraints on the matrix $U$	Constraints on matrix $V$
LSA	$\ D - UV\ _F^2$	$U^T U = I$	$V V^T = \Lambda^2$
NMF	$\ D - UV\ _F^2$	$u_{mk} \geq 0$	$v_{kn} \geq 0$
PLSA	$\sum_{mn} d_{mn} \log \frac{d_{mn}}{(UV)_{mn}}$	$U^T 1 = 1$ $u_{mk} \geq 0$	$V^T 1 = 1$ $v_{kn} \geq 0$

**Fig. 22.2** Probabilistic graphical model representation of the topic models LSA and NMF**Table 22.4** Constraints of the topic models LSA and NMF

Method	Constraints on the variable $u_k$	Constraints on the variable $v_n$
LSA	Orthogonal	Orthogonal
NMF	$u_{mk} \geq 0$	$v_{kn} \geq 0$

formulas of the three topic models, in which the matrix  $D$  of PLSA needs to be normalized as  $\sum_{m,n} d_{mn} = 1$ .

Topic models LSA and NMF are non-probabilistic models, but they also have probabilistic model interpretations. LSA, NMF, PLSA, and LDA can be viewed in the unified framework of probabilistic graphical models, in which the text is considered to be generated by probabilistic models, and four different topic models are obtained based on different hypotheses.

The four topic models have different probabilistic graphical model definitions. For LSA and NMF, each text  $d_n$  is generated by Gaussian distribution  $P(d_n|U, v_n) \propto \exp(-\|d_n - Uv_n\|^2)$ , whose parameters are  $U$  and  $v_n$ . There are a total of  $N$  texts, as shown in Fig. 22.2. The two topic models have different constraints. The formulas for the constraints are given in Table 22.4.

## References

1. Singh AP, Gordon GJ. A unified view of matrix factorization models. In: Daelemans W, Goethals B, Morik K, editors. Machine learning and knowledge discovery in databases. ECML PKDD 2008. Lecture notes in computer science, vol. 5212. Berlin: Springer; 2008.

2. Wang Q, Xu J, Li H, et al. Regularized latent semantic indexing: a new approach to large-scale topic modeling. ACM Trans Inf Syst (TOIS). 2013;31(1):5.

# Appendix A

## Gradient Descent

Gradient descent or steepest descent is one of the most commonly used methods for solving unconstrained optimization problems, which has the advantage of simple implementation. The gradient descent method is an iterative algorithm, and each step needs to solve the gradient vector of the objective function.

Let's say that  $f(x)$  is a function of  $\mathbf{R}^n$  with the first continuous partial derivative. The unconstrained optimization problem that requires a solution is

$$\min_{x \in \mathbf{R}^n} f(x) \quad (\text{A.1})$$

$x^*$  represents the minima of the objective function  $f(x)$ .

Gradient descent is an iterative algorithm. Select the appropriate initial value  $x^{(0)}$ , keep iterating, update the value of  $x$ , and minimize the objective function until convergence. Since the negative gradient direction causes the fastest decline of the function value, in each step of the iteration, update the value of  $x$  in the negative gradient direction to achieve the purpose of reducing the function value.

Since  $f(x)$  has a first-order continuous partial derivative, if the value of the  $k$ -th iteration is  $x^{(k)}$ , the first-order Taylor expansion of  $f(x)$  can be carried out near  $x^{(k)}$ :

$$f(x) = f(x^{(k)}) + g_k^T(x - x^{(k)}) \quad (\text{A.2})$$

where  $g_k = g(x^{(k)}) = \nabla f(x^{(k)})$  is the gradient of  $f(x)$  on  $x^{(k)}$ .

Calculate the  $k + 1$  iteration value  $x^{(k+1)}$ :

$$x^{(k+1)} \leftarrow x^{(k)} + \lambda_k p_k \quad (\text{A.3})$$

where  $p_k$  is the search direction, taking the negative gradient direction  $p_k = -\nabla f(x^{(k)})$ ,  $\lambda_k$  is the step size, determined by one-dimensional search, i.e.,  $\lambda_k$  makes

$$f(x^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(x^{(k)} + \lambda p_k) \quad (\text{A.4})$$

The algorithm of gradient descent method is as follows:

**Algorithm A.1** (*Gradient Descent*)

Input: the objective function  $f(x)$ , the gradient function  $g(x) = \nabla f(x)$ , and the calculation accuracy  $\varepsilon$ ;

Output:  $x^*$ , which is the smallest point of  $f(x)$ .

- (1) Take the initial value  $x^{(0)} \in \mathbf{R}^n$  and set  $k = 0$ .
- (2) Calculate  $f(x^{(k)})$ .
- (3) Calculate the gradient  $g(x) = g(x^{(k)})$ . When  $\|g_k\| < \varepsilon$ , the iteration is stopped and  $x^* = x^{(k)}$ . Otherwise, let  $p_k = -g(x^{(k)})$  and solve for  $\lambda_k$ , so that

$$f(x^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(x^{(k)} + \lambda p_k)$$

- (4) Set  $x^{(k+1)} = x^{(k)} + \lambda_k p_k$ , and calculate  $f(x^{(k+1)})$ .  
When  $\|f(x^{(k+1)}) - f(x^{(k)})\| < \varepsilon$  or  $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ , the iteration is stopped and  $x^* = x^{(k+1)}$ .
- (5) Otherwise, set  $k = k + 1$ , turn to (3).

When the objective function is convex, the solution of the gradient descent method is globally optimal. In general, its solution is not guaranteed to be the optimal global solution. Gradient descent does not necessarily converge very quickly.

## Appendix B

# Newton Method and Quasi-Newton Method

Newton method and quasi-Newton method are also commonly used to solve unconstrained optimization problems, which have the advantage of fast convergence. However, Newton method is an iterative algorithm, and each step needs to solve the inverse matrix of Hessian matrix of the objective function, the calculation is more complex. The quasi-Newton method simplifies the calculation process by approximating the Hessian matrix or Hessian matrix's inverse matrix by a positive definite matrix.

### 1. Newton method

Consider an unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \quad (\text{B.1})$$

where  $x^*$  is the minimal value of the objective function.

Assuming that  $f(x)$  has a second-order continuous partial derivative, if the  $k$ -th iteration value is  $x^{(k)}$ , then  $f(x)$  can be carried out a second-order Taylor expansion near  $x^{(k)}$ :

$$f(x) = f(x^{(k)}) + g_k^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T H(x^{(k)})(x - x^{(k)}) \quad (\text{B.2})$$

where  $g_k = g(x^{(k)}) = \nabla f(x^{(k)})$  is the gradient of  $f(x)$  on  $x^{(k)}$ , and  $H(x^{(k)})$  is the Hessian matrix of  $f(x)$

$$H(x) = \left[ \frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{n \times n} \quad (\text{B.3})$$

value at the point  $x^{(k)}$ . The necessary condition for  $f(x)$  to have an extreme value is that the first derivative at the extreme point is 0, that is, the gradient vector is 0. In particular, when  $H(x^{(k)})$  is a positive definite matrix, the extremum of the function  $f(x)$  is a minimal value.

The necessary condition for Newton's method to use the minimal point is

$$\nabla f(x) = 0 \quad (\text{B.4})$$

In each iteration, we start from the point  $x^{(k)}$  and find the minimal point of the objective function, which is the value of the  $k + 1$  iteration value  $x^{(k+1)}$ .

Specifically, assume that  $x^{(k+1)}$  satisfies:

$$\nabla f(x^{(k+1)}) = 0 \quad (\text{B.5})$$

By Eq. (B.2)

$$\nabla f(x) = g_k + H_k(x - x^{(k)}) \quad (\text{B.6})$$

where  $H_k = H(x^{(k)})$ . Thus, Eq. (B.5) becomes

$$g_k + H_k(x^{(k+1)} - x^{(k)}) = 0 \quad (\text{B.7})$$

Therefore,

$$x^{(k+1)} = x^{(k)} - H_k^{-1}g_k \quad (\text{B.8})$$

or,

$$x^{(k+1)} = x^{(k)} + p_k \quad (\text{B.9})$$

where,

$$H_k p_k = -g_k \quad (\text{B.10})$$

The algorithm using Eq. (B.8) as the iterative formula is Newton method.

### **Algorithm B.1** (*Newton Method*)

Input: the objective function  $f(x)$ , the gradient function  $g(x) = \nabla f(x)$ , the Hessian matrix  $H(x)$ , and the calculation accuracy  $\varepsilon$ ;

Output:  $x^*$ , which is the smallest point of  $f(x)$ .

- (1) Take the initial value  $x^{(0)}$  and set  $k = 0$ .
- (2) Calculate  $g_k = g(x^{(k)})$ .
- (3) If  $\|g_k\| < \varepsilon$ , the calculation is stopped and an approximate solution  $x^* = x^{(k)}$  is obtained.
- (4) Calculate  $H_k = H(x^{(k)})$  and calculate  $p_k$ .

$$H_k p_k = -g_k$$

- (5) Set  $x^{(k+1)} = x^{(k)} + p_k$ .  
 (6) Set  $k = k + 1$ , and turn to (2).

Step (4) calculates  $p_k$ ,  $p_k = -H_k^{-1}g_k$ .  $H_k^{-1}$  is required, the calculation is relatively complex, so there are other improved methods.

## 2. The idea of quasi-Newton method

In the iteration of Newton method, the inverse matrix  $H^{-1}$  of Hessian matrix is needed to be calculated. This calculation is relatively complicated, and an  $n$ -order matrix  $G_k = G(x^{(k)})$  is considered to approximate  $H_k^{-1} = H^{-1}(x^{(k)})$ . So, that's the basic idea of the quasi-Newton method.

Let's first look at the conditions that Hessian matrix  $H_k$  satisfies in Newton method iteration. First of all,  $H_k$  satisfies the following relationship. In Eq. (B.6), take  $x = x^{(k+1)}$ , and get

$$g_{k+1} - g_k = H_k(x^{(k+1)} - x^{(k)}) \quad (\text{B.11})$$

Let  $y_k = g_{k+1} - g_k$ ,  $\delta_k = x^{(k+1)} - x^{(k)}$ , then

$$y_k = H_k\delta_k \quad (\text{B.12})$$

or,

$$H_k^{-1}y_k = \delta_k \quad (\text{B.13})$$

Equation (B.12) or Eq. (B.13) are called quasi-Newton conditions.

If  $H_k$  is positive definite ( $H_k^{-1}$  is also positive definite), then the search direction  $p_k$  of Newton's method can be guaranteed to be the descending direction. This is because the search direction is  $p_k = -H_k^{-1}g_k$ , as shown in Eq. (B.8),

$$x = x^{(k)} + \lambda p_k = x^{(k)} - \lambda H_k^{-1}g_k \quad (\text{B.14})$$

Therefore, the Taylor expansion (B.2) of  $f(x)$  at  $x^{(k)}$  can be approximately written as:

$$f(x) = f(x^{(k)}) - \lambda g_k^T H_k^{-1} g_k \quad (\text{B.15})$$

Since  $H_k^{-1}$  is positive definite,  $g_k^T H_k^{-1} g_k > 0$ . When  $\lambda$  is a sufficiently small positive number, there is always  $f(x) < f(x^{(k)})$ , that is,  $p_k$  is in the downward direction.

Quasi-Newton method takes  $G_k$  as an approximation of  $H_k^{-1}$  and requires the matrix  $G_k$  to satisfy the same conditions. First, each iteration of the matrix  $G_k$  is positive definite. Meanwhile,  $G_k$  satisfies the following quasi-Newton conditions:

$$G_{k+1}y_k = \delta_k \quad (\text{B.16})$$

The algorithm that chooses  $G_k$  as the approximation of  $H_k^{-1}$  or  $B_k$  as the approximation of  $H_k$  according to the quasi-Newton condition is called the quasi-Newton method.

According to the quasi-Newton condition, the update matrix  $G_{k+1}$  can be selected in each iteration:

$$G_{k+1} = G_k + \Delta G_k \quad (\text{B.17})$$

There is some flexibility to this choice, so there are a number of concrete ways to implement it. The Broyden quasi-Newton method is described below.

### 3. Davidon-Fletcher-Powell (DFP) algorithm

The DFP algorithm selects  $G_{k+1}$  by assuming that the matrix  $G_{k+1}$  in each iteration is composed of  $G_k$  plus two additional items, i.e.,

$$G_{k+1} = G_k + P_k + Q_k \quad (\text{B.18})$$

where  $P_k$  and  $Q_k$  are undetermined matrices. At this time,

$$G_{k+1}y_k = G_k y_k + P_k y_k + Q_k y_k \quad (\text{B.19})$$

In order to make  $G_{k+1}$  meet the quasi-Newton condition,  $P_k$  and  $Q_k$  can meet:

$$P_k y_k = \delta_k \quad (\text{B.20})$$

$$Q_k y_k = -G_k y_k \quad (\text{B.21})$$

In fact, it's not hard to find such  $P_k$  and  $Q_k$ , for example, take

$$P_k = \frac{\delta_k \delta_k^T}{\delta_k^T y_k} \quad (\text{B.22})$$

$$Q_k = -\frac{G_k y_k y_k^T G_k}{y_k^T G_k y_k} \quad (\text{B.23})$$

In this way, the iterative formula of matrix  $G_{k+1}$  can be obtained:

$$G_{k+1} = G_k + \frac{\delta_k \delta_k^T}{\delta_k^T y_k} - \frac{G_k y_k y_k^T G_k}{y_k^T G_k y_k} \quad (\text{B.24})$$

Which is called the DFP algorithm.

It can be proved that if the initial matrix  $G_0$  is positive definite, then every matrix  $G_k$  in the iteration process is positive definite.

The DFP algorithm is as follows:

**Algorithm B.2 (DFP Method)**

Input: the objective function  $f(x)$ , the gradient function  $g(x) = \nabla f(x)$ , and the calculation accuracy  $\varepsilon$ ;

Output:  $x^*$ , which is the smallest point of  $f(x)$ .

- (1) Take the initial value  $x^{(0)}$ , let  $G_0$  be a positive definite symmetric matrix and set  $k = 0$ .
- (2) Calculate  $g_k = g(x^{(k)})$ . If  $\|g_k\| < \varepsilon$ , the calculation is stopped and an approximate solution  $x^* = x^{(k)}$  is obtained; otherwise, turn to (3).
- (3) Set  $p_k = -G_k g_k$ .
- (4) One dimensional search:  $\lambda_k$  makes

$$f(x^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(x^{(k)} + \lambda p_k)$$

- (5) Set  $x^{(k+1)} = x^{(k)} + \lambda_k p_k$ .
- (6) Calculate  $g_{k+1} = g(x^{(k+1)})$ . If  $\|g_{k+1}\| < \varepsilon$ , the calculation is stopped and an approximate solution  $x^* = x^{(k+1)}$  is obtained; otherwise, calculate  $G_{k+1}$  according to Eq. (B.24).
- (7) Set  $k = k + 1$ , and turn to (3).

#### 4. Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

BFGS algorithm is the most popular quasi-Newton algorithm.

The inverse matrix  $H^{-1}$  of Hessian matrix can be approximated by  $G_k$ , and the inverse matrix  $H$  can be approximated by  $H_k$ .

In this case, the corresponding quasi-Newton condition is

$$B_{k+1} \delta_k = y_k \quad (\text{B.25})$$

Another iterative formula can be obtained in the same way. The first order

$$B_{k+1} = B_k + P_k + Q_k \quad (\text{B.26})$$

$$B_{k+1} \delta_k = B_k \delta_k + P_k \delta_k + Q_k \delta_k \quad (\text{B.27})$$

Consider making  $P_k$  and  $Q_k$  satisfy:

$$P_k \delta_k = y_k \quad (\text{B.28})$$

$$Q_k \delta_k = -B_k \delta_k \quad (\text{B.29})$$

Find out  $P_k$  and  $Q_k$  that are suitable for the conditions, and get the iterative formula of BFGS algorithm matrix  $B_{k+1}$ :

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T \delta_k} - \frac{B_k \delta_k \delta_k^T B_k}{\delta_k^T B_k \delta_k} \quad (\text{B.30})$$

It can be proved that if the initial matrix  $B_0$  is positive definite, then every matrix  $B_k$  in the iteration process is positive definite.

Let's write the BFGS quasi-Newton algorithm.

### Algorithm B.3 (BFGS Method)

Input: the objective function  $f(x)$ , the gradient function  $g(x) = \nabla f(x)$ , and the calculation accuracy  $\varepsilon$ ;

Output:  $x^*$ , which is the smallest point of  $f(x)$ .

- (1) Take the initial value  $x^{(0)}$ , let  $B_0$  be a positive definite symmetric matrix and set  $k = 0$ .
- (2) Calculate  $g_k = g(x^{(k)})$ . If  $\|g_k\| < \varepsilon$ , the calculation is stopped and an approximate solution  $x^* = x^{(k)}$  is obtained; otherwise, turn to (3).
- (3) Calculate  $p_k$  from  $B_k p_k = -g_k$ .
- (4) One dimensional search:  $\lambda_k$  makes

$$f(x^{(k)} + \lambda_k p_k) = \min_{\lambda \geq 0} f(x^{(k)} + \lambda p_k)$$

- (5) Set  $x^{(k+1)} = x^{(k)} + \lambda_k p_k$ .
- (6) Calculate  $g_{k+1} = g(x^{(k+1)})$ . If  $\|g_{k+1}\| < \varepsilon$ , the calculation is stopped and an approximate solution  $x^* = x^{(k+1)}$  is obtained; otherwise, calculate  $B_{k+1}$  according to Eq. (B.30).
- (7) Set  $k = k + 1$ , and turn to (3).

## 5. Broyden's algorithm

We can obtain the iterative formula of BFGS algorithm on  $G_k$  from the iterative equation (B.30) of BFGS algorithm matrix  $B_k$ . In fact, if we write  $G_k = B_k^{-1}$  and  $G_{k+1} = B_{k+1}^{-1}$ , then the Sherman-Morrison equation<sup>1</sup> is applied to Eq. (B.30) twice,

$$G_{k+1} = \left( I - \frac{\delta_k y_k^T}{\delta_k^T y_k} \right) G_k \left( I - \frac{\delta_k y_k^T}{\delta_k^T y_k} \right)^T + \frac{\delta_k \delta_k^T}{\delta_k^T y_k} \quad (\text{B.31})$$

is called the iterative equation of BFGS algorithm on  $G_k$ .

<sup>1</sup> Sherman-Morrison equation: Suppose  $A$  is an invertible matrix of  $n$ -order,  $u$  and  $v$  are  $n$ -dimensional vectors, and  $A + uv^T$  is also an invertible matrix, then

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$$

The  $G_{k+1}$  obtained by the iterative equation (B.23) of the DFP algorithm  $G_k$  is denoting  $G^{\text{DFP}}$ , and the  $G_{k+1}$  obtained by the iterative equation (B.31) of the BFGS algorithm  $G_k$  is denoting  $G^{\text{BFGS}}$ . Both of them satisfy the quasi-Newton condition equation, so they are linear combination

$$G_{k+1} = \alpha G^{\text{DFP}} + (1 - \alpha) G^{\text{BFGS}} \quad (\text{B.32})$$

which also satisfies the quasi-Newton condition, and is positive definite, where  $0 \leq \alpha \leq 1$ . This leads to a class of quasi-Newton methods, known as Broyden's algorithms.

# Appendix C

## Language Duality

In constrained optimization problems, Lagrange duality is often used to transform the original problem into a dual problem, and the solution of the original problem is obtained by solving the dual problem. This method is used in many statistical learning methods, such as the maximum entropy model and support vector machine. The main concepts and results of Lagrangian duality are briefly described here.

### 1. Original problem

Let  $f(x)$ ,  $c_i(x)$ ,  $h_j(x)$  be continuously differentiable functions defined on  $\mathbb{R}^n$ . Consider constrained optimization problems,

$$\min_{x \in \mathbb{R}^n} f(x) \quad (C.1)$$

$$s.t. \quad c_i(x) \leq 0, \quad i = 1, 2, \dots, k \quad (C.2)$$

$$h_j(x) = 0, \quad j = 1, 2, \dots, l \quad (C.3)$$

The constrained optimization problem is called a primitive optimization problem or primitive problem.

Firstly, the generalized Lagrange function is introduced

$$L(x, \alpha, \beta) = f(x) + \sum_{i=1}^k \alpha_i c_i(x) + \sum_{j=1}^l \beta_j h_j(x) \quad (C.4)$$

Here,  $x = (x^{(1)}, x^{(2)}, \dots, x^{(n)})^\top \in \mathbb{R}^n$ ,  $\alpha_i, \beta_j$  is the Lagrange multiplier,  $\alpha_i \geq 0$ . Consider the function of  $x$ :

$$\theta_P(x) = \max_{\alpha, \beta: \alpha_i \geq 0} L(x, \alpha, \beta) \quad (C.5)$$

Here, the subscript  $P$  denotes the original problem.

Assume that  $x$  is given. If  $x$  violates the constraints of the original problem, that is, if there is a  $i$  such that  $c_i(x) > 0$  or a  $j$  such that  $h_j(x) \neq 0$ , then there is

$$\theta_P(x) = \max_{\alpha, \beta: \alpha_i \geq 0} \left[ f(x) + \sum_{i=1}^k a_i c_i(x) + \sum_{j=1}^l \beta_j h_j(x) \right] = +\infty \quad (\text{C.6})$$

Because if a  $i$  makes the constraint  $c_i > 0$ , then  $a_i \rightarrow +\infty$ , if  $j$  makes  $h_j(x) \neq 0$ ,  $\beta_j$  make  $\beta_j h_j \rightarrow +\infty$ , and take the rest  $\alpha_i, \beta_j$  as 0.

On the contrary, if  $x$  satisfies the constraint conditions of Eqs. (C.2) and (C.3), it can be seen from Eqs. (C.5) and (C.4),  $\theta_P(x) = f(x)$ . Therefore,

$$\theta_P(x) = \begin{cases} f(x), & x \text{ satisfies the constraints of the original problem} \\ +\infty, & \text{other} \end{cases} \quad (\text{C.7})$$

So if you think about minimization

$$\min_x \theta_P(x) = \min_x \max_{\alpha, \beta: \alpha_i \geq 0} L(x, \alpha, \beta) \quad (\text{C.8})$$

It is equivalent to the original optimization problem (C.1)–(C.3), that is, they have the same solution. The  $\min_x \max_{\alpha, \beta: \alpha_i \geq 0} L(x, \alpha, \beta)$  problem is called the minimax problem of generalized Lagrangian function. In this way, the original optimization problem is expressed as a minimax problem of generalized Lagrangian function. For convenience, the optimal value of the original problem is defined

$$p^* = \min_x \theta_P(x) \quad (\text{C.9})$$

This is called the value of the original problem.

## 2. Dual problem

Definition

$$\theta_D(\alpha, \beta) = \min_x L(x, \alpha, \beta) \quad (\text{C.10})$$

Consider maximization again  $\theta_D(\alpha, \beta) = \min_x L(x, \alpha, \beta)$ , that is

$$\max_{\alpha, \beta: \alpha_i \geq 0} \theta_D(\alpha, \beta) = \max_{\alpha, \beta: \alpha_i \geq 0} \min_x L(x, \alpha, \beta) \quad (\text{C.11})$$

Questions  $\max_{\alpha, \beta: \alpha_i \geq 0} \min_x L(x, \alpha, \beta)$  Polar functions are called generalized Lagrangian functions.

The minimax problem of generalized Lagrangian function can be expressed as a constrained optimization problem:

$$\max_{\alpha, \beta} \theta_D(\alpha, \beta) = \max_{\alpha, \beta} \min_x L(x, \alpha, \beta) \quad (\text{C.12})$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, 2, \dots, k \quad (\text{C.13})$$

It is called dual problem of primal problem. Define the optimal value of dual problem

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \theta_D(\alpha, \beta) \quad (\text{C.14})$$

It is called the value of the dual problem.

### 3. The relationship between Primal problem and Dual problem

The relationship between the primal problem and the dual problem is discussed below.

**Theorem C.1** *If both primal and dual problems have optimal values, then*

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \min_x L(x, \alpha, \beta) \leq \min_x \max_{\alpha, \beta: \alpha_i \geq 0} L(x, \alpha, \beta) = p^* \quad (\text{C.15})$$

**Proof** It is proved that by Formulas (C.12) and (C.5), for any  $\alpha, \beta$  and  $x$ ,

$$\theta_D(\alpha, \beta) = \min_x L(x, \alpha, \beta) \leq L(x, \alpha, \beta) \leq \max_{\alpha, \beta: \alpha_i \geq 0} L(x, \alpha, \beta) = \theta_P(x) \quad (\text{C.16})$$

i.e.,

$$\theta_D(\alpha, \beta) \leq \theta_P(x) \quad (\text{C.17})$$

Because both the primal problem and the dual problem have the optimal values, so

$$\max_{\alpha, \beta: \alpha_i \geq 0} \theta_D(\alpha, \beta) \leq \min_x \theta_P(x) \quad (\text{C.18})$$

i.e.,

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \min_x L(x, \alpha, \beta) \leq \min_x \max_{\alpha, \beta: \alpha_i \geq 0} L(x, \alpha, \beta) = p^* \quad (\text{C.19})$$

**Corollary C.1** *Let  $x^*$  and  $\alpha^*, \beta^*$  is the feasible solution of the primal problem (C.1)–(C.3) and the dual problem (C.12)–(C.13), and  $d^* = p^*$ , then  $x^*$  and  $\alpha^*, \beta^*$  are the optimal solutions of the primal problem and the dual problem respectively.*

Under some conditions, the optimal values of the primal problem and dual problem are equal,  $d^* = p^*$ . In this case, we can solve the dual problem instead of the original problem. In the following, the important conclusions are described in the form of theorems without proof.

**Theorem C.2** Consider the primal problem (C.1)–(C.3) and the dual problem (C.12)–(C.13). Let  $f(x)$  and  $c_i(x)$  be convex functions and  $h_j(x)$  be affine functions. And suppose that the inequality constraint  $c_i(x)$  is strictly feasible, that is, there exists  $x$  with  $c_i(x) < 0$  for all  $i$ , then there exists  $x^*, \alpha^*, \beta^*$ , so that  $x^*$  is the solution of the original problem,  $\alpha^*, \beta^*$  is the solution of the dual problem, and

$$p^* = d^* = L(x^*, \alpha^*, \beta^*) \quad (\text{C.20})$$

**Theorem C.3** For the primal problem (C.1)–(C.3) and dual problem (C.12)–(C.13), suppose that functions  $f(x)$  and  $c_i(x)$  are convex functions,  $h_j(x)$  are affine functions, and inequality constraint  $c_i(x)$  is strictly feasible, then  $x^*$  and  $\alpha^*, \beta^*$  is the solution of the primal problem and the dual problem respectively, if and only if  $x^*, \alpha^*, \beta^*$  satisfy the following Karush-Kuhn-Tucher (KKT) condition:

$$\nabla_x L(x^*, \alpha^*, \beta^*) = 0 \quad (\text{C.21})$$

$$\alpha_i^* c_i(x^*) = 0, \quad i = 1, 2, \dots, k \quad (\text{C.22})$$

$$c_i(x^*) \leq 0, \quad i = 1, 2, \dots, k \quad (\text{C.23})$$

$$\alpha_i^* \geq 0, \quad i = 1, 2, \dots, k \quad (\text{C.24})$$

$$h_j(x^*) \geq 0, \quad j = 1, 2, \dots, l \quad (\text{C.25})$$

In particular, Eq. (C.22) is called the dual complementary condition of KKT. It can be seen from this condition that if  $\alpha_i^* > 0$ , then  $c_i(x^*) = 0$ .

# Appendix D

## Basic Subspaces of Matrix

This paper briefly introduces the definition and theorem of the basic subspace of the matrix used in this book.

### 1. Subspace of vector space

If  $S$  is a nonempty subset of vector space  $V$ , and  $S$  satisfies the following conditions:

- (1) For any real number  $a$ , if  $x \in S$ , then  $ax \in S$ ;
- (2) If  $x \in S$  and  $y \in S$ , then  $x + y \in S$ ;

Then  $S$  is called the subspace of  $V$ .

Let  $v_1, v_2, \dots, v_n$  be the vector in the vector space  $V$ , then its linear combination

$$a_1v_1 + a_2v_2 + \cdots + a_nv_n$$

forms the subspace of  $V$ , which is called the subspace spanned by  $v_1, v_2, \dots, v_n$ , or span of  $v_1, v_2, \dots, v_n$  is denoted

$$\text{span}\{v_1, v_2, \dots, v_n\}^\circ$$

If  $\text{span}\{v_1, v_2, \dots, v_n\} = V$ , then  $v_1, v_2, \dots, v_n$  span  $V$ .

### 2. Basis and dimension of vector space

The vectors  $v_1, v_2, \dots, v_n$  in vector space  $V$  are called the bases of space  $V$ , if the conditions are satisfied.

- (1)  $v_1, v_2, \dots, v_n$  are linearly independent;
- (2)  $v_1, v_2, \dots, v_n$  span  $V$ .

vice versa. The number of bases of vector space is the dimension of vector space.

### 3. Row space and column space of the matrix

Let  $A$  be one  $m \times n$  matrix. Each row of  $A$  can be regarded as a vector in  $\mathbb{R}^n$ , which is called row vector of  $A$ . Similarly, each column of  $A$  can be regarded as a vector in  $\mathbb{R}^m$ , which is called column vector of  $A$ .

Let  $A$  be one  $m \times n$  matrix, then the subspace of  $\mathbb{R}^n$ , which is spanned by the row vector of  $A$ , is called the row space of  $A$ ; The subspace of  $\mathbb{R}^m$  spanned by the column vector of  $A$  is called the column space of  $A$ .

The dimension of the row space of the matrix  $A$  is equal to that of column space.

The dimension of row space of a matrix (equivalent to the dimension of column space) is called the rank of a matrix.

#### 4. Null space of the Matrix

Let  $A$  be  $m \times n$  a matrix, let  $N(A)$  be the set of all solutions of the system of homogeneous equations  $Ax = 0$ , then  $N(A)$  is a subspace of  $\mathbb{R}^n$ , which is called the null space of  $A$ , i.e.,

$$N(A) = \{x \in \mathbb{R}^n | Ax = 0\} \quad (\text{D.1})$$

The dimension of the zero space of a matrix is called the nullity of a matrix.

**Rank-nullity theorem.** Let  $A$  be one  $m \times n$  matrix, then the sum of the rank of  $A$  and the nullity of  $A$  is  $n$ . In fact, if the rank of  $A$  is  $r$ , then the number of independent variables of  $Ax = 0$  is  $r$  and the number of free variables is  $(n - r)$ . The dimension of  $N(A)$  is equal to the number of free variables. So the theorem holds.

#### 5. Orthogonal complement of a subspace

Let  $X$  and  $Y$  be subspaces of  $\mathbb{R}^n$ . If  $x^T y = 0$  is satisfied for every  $x \in X$  and  $y \in Y$ , then  $X$  and  $Y$  are said to be orthogonal, denoted as  $X \perp Y$ .

Let  $Y$  be the subspace of  $\mathbb{R}^n$ , and the set of vectors in  $\mathbb{R}^n$  orthogonal to each vector in  $Y$  is denoted as  $Y^\perp$ , i.e.,

$$Y^\perp = \{x \in \mathbb{R}^n | x^T y = 0, \forall y \in Y\} \quad (\text{D.2})$$

The set  $Y^\perp$  is called the orthogonal complement of  $Y$ .

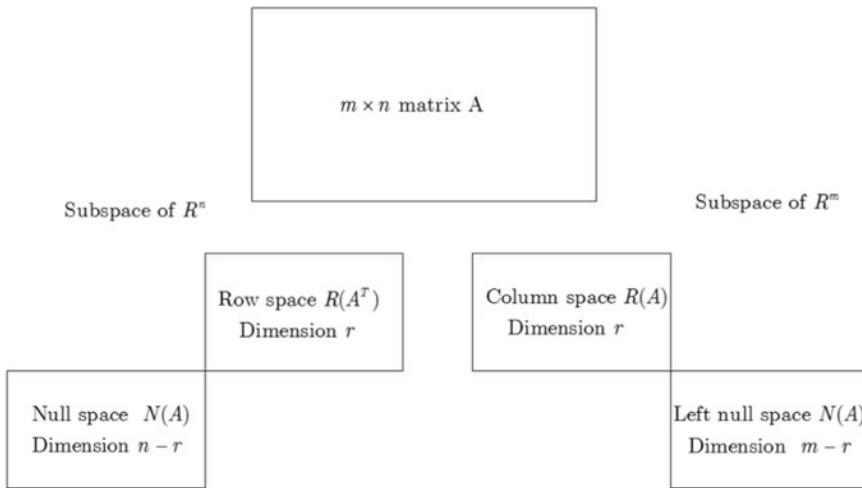
It can be proved that if  $Y$  is a subspace of  $\mathbb{R}^n$ , then  $Y^\perp$  is also a subspace of  $\mathbb{R}^n$ .

#### 6. The basic subspace of matrix

Let  $A$  be one  $m \times n$  matrix,  $A$  can be regarded as a linear transformation from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ . The necessary and sufficient condition for a vector  $z \in \mathbb{R}^m$  in the column space of  $A$  is that there exists  $x \in \mathbb{R}^n$  such that  $z = Ax$ . In this way, the column space of  $A$  and the range of  $A$  are the same. Let a be  $R(A)$ , then

$$\begin{aligned} R(A) &= \{z \in \mathbb{R}^m | \exists x \in \mathbb{R}^n, z = Ax\} \\ &= \text{Column space of } A \end{aligned} \quad (\text{D.3})$$

Similarly, the necessary and sufficient condition for a vector  $y \in \mathbb{R}^n$ ,  $y^T$  in the row space of  $A$  is that there exists  $x \in \mathbb{R}^m$  such that  $y = A^T x$ . In this way, the row space of  $A$  and the range  $R(A^T)$  of  $A^T$  are the same.



**Fig. D.1** The relationship between the basic subspaces of a matrix

$$\begin{aligned} R(A^T) &= \{y \in \mathbf{R}^n \mid \exists x \in \mathbf{R}^m, y = A^T x\} \\ &= \text{Row space of } A \end{aligned} \quad (\text{D.4})$$

Matrix  $A$  has four basic subspaces: column space, row space, null space and transposed null space of  $A$  (left null space). The following theorem holds.

**Theorem D.1** If  $A$  is a  $m \times n$  matrix, then  $N(A) = R(A^T)^\perp$ , and  $N(A^T) = R(A)^\perp$ .

**Proof** It is easy to verify  $R(A^T) \perp N(A)$ . Because  $R(A^T) \perp N(A)$ ,  $N(A) \subset R(A^T)^\perp$  is obtained. On the other hand, if  $x$  is any vector in  $R(A)^\perp$ , then  $x$  and each column vector of  $A^T$  are orthogonal. Therefore,  $Ax = 0$  can be obtained. So  $x$  must be an element of  $N(A)$ , and from this we get

$$N(A) = R(A^T)^\perp \quad (\text{D.5})$$

Similar available

$$N(A^T) = R(A)^\perp \quad (\text{D.6})$$

Figure D.1 shows the relationship between the basic subspaces of the matrix.

# Appendix E

## The Definition of KL Divergence and the Properties of Dirichlet Distribution

### 1. Definition of KL divergence

First, give the definition of KL divergence (Kullback–Leibler divergence). KL divergence is a metric for describing the similarity between two probability distributions  $Q(x)$  and  $P(x)$ , denoted as  $D(Q\|P)$ . For discrete random variables, the KL divergence is defined as

$$D(Q\|P) = \sum_i Q(i) \log \frac{Q(i)}{P(i)} \quad (\text{E.1})$$

For continuous random variables, the KL divergence is defined as

$$D(Q\|P) = \int Q(x) \log \frac{Q(x)}{P(x)} dx \quad (\text{E.2})$$

It is easy to prove that the KL divergence has the property:  $D(Q\|P) \geq 0$ . If and only if  $Q = P$ ,  $D(Q\|P) = 0$ . In fact, using Jensen inequality, we get

$$\begin{aligned} -D(Q\|P) &= \int Q(x) \log \frac{P(x)}{Q(x)} dx \\ &\leq \log \int Q(x) \frac{P(x)}{Q(x)} dx \\ &= \log \int P(x) dx = 0 \end{aligned} \quad (\text{E.3})$$

KL divergence is asymmetric, and does not satisfy the triangular inequality, so it is not a distance metric in the strict sense.

### 2. The properties of Dirichlet distribution

Assuming that the random variable  $\theta$  obeys the Dirichlet distribution  $\theta \sim Dir(\theta|\alpha)$ , using the properties of the exponential distribution family, find the mathematical expectation  $E[\log \theta]$  of the Dirichlet distribution of the function  $E[\log \theta]$ .

The exponential distribution family refers to a collection of probability distributions whose probability distribution density can be written as follows:

$$p(x|\eta) = h(x) \exp\{\eta^T T(x) - A(\eta)\} \quad (\text{E.4})$$

where  $\eta$  is a natural parameter,  $T(x)$  is a sufficient statistic,  $h(x)$  is a potential measure,  $A(\eta)$  is a logarithmic normalization factor, the specific explanation of  $A(\eta)$  is as follows:  $A(\eta) = \log \int h(x) \exp\{\eta^T T(x)\} dx$ .

The exponential distribution family has properties: the derivative of the logarithmic normalization factor  $A(\eta)$  to the natural parameter  $\eta$  is equal to the mathematical expectation of the sufficient statistic  $T(x)$ . In fact,

$$\begin{aligned} \frac{d}{d\eta} A(\eta) &= \frac{d}{d\eta} \log \int h(x) \exp\{\eta^T T(x)\} dx \\ &= \frac{\int T(x) \exp\{\eta^T T(x)\} h(x) dx}{\int h(x) \exp\{\eta^T T(x)\} dx} \\ &= \int T(x) \exp\{\eta^T T(x) - A(\eta)\} h(x) dx \\ &= \int T(x) p(x|\eta) dx \\ &= E[T(X)] \end{aligned} \quad (\text{E.5})$$

The Dirichlet distribution belongs to the exponential distribution family, because its density function can be written as the density function form of the exponential distribution family

$$\begin{aligned} p(\theta|\alpha) &= \frac{\Gamma\left(\sum_{l=1}^K \alpha_l\right)}{\prod_{k=1}^K \Gamma(\alpha_k)} \prod_{k=1}^K \theta_k^{\alpha_k-1} \\ &= \exp\left\{ \left( \sum_{k=1}^K (\alpha_k - 1) \log \theta_k \right) + \log \Gamma\left(\sum_{l=1}^K \alpha_l\right) - \sum_{l=1}^K \log \Gamma(\alpha_k) \right\} \quad (\text{E.6}) \end{aligned}$$

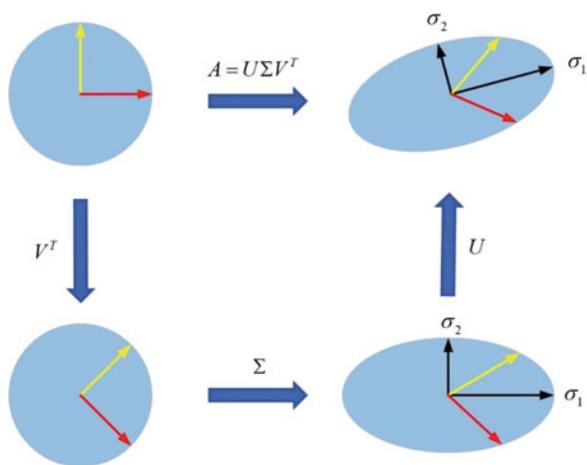
The natural parameter is  $\eta_k = \alpha_k - 1$ . The sufficient statistic is  $T(\theta_k) = \log \theta_k$ , and the logarithmic normalization factor is  $A(\alpha) = \sum_{k=1}^K \log \Gamma(\alpha_k) - \log \Gamma\left(\sum_{l=1}^K \alpha_l\right)$ .

Using the property (E.5), the derivative of the logarithmic normalization factor to the natural parameter is equal to the mathematical expectation of sufficient statistics, and the mathematical expectation  $E_{p(\theta|\alpha)}[\log \theta]$  of the Dirichlet distribution is obtained as follows

$$\begin{aligned}
 E_{p(\theta|\alpha)}[\log \theta_k] &= \frac{d}{d\alpha_k} A(\alpha) = \frac{d}{d\alpha_k} \left[ \sum_{k=1}^K \log \Gamma(\alpha_k) - \log \Gamma\left(\sum_{l=1}^K \alpha_l\right) \right] \\
 &= \Psi(\alpha_k) - \Psi\left(\sum_{l=1}^K \alpha_l\right), \quad k = 1, 2, \dots, K
 \end{aligned} \tag{E.7}$$

where  $\Psi$  is the digamma function, i.e., the first derivative of the log Gamma function.

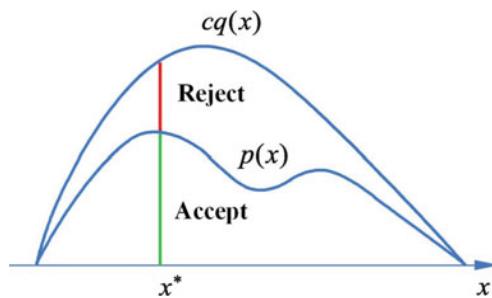
# Color Diagrams



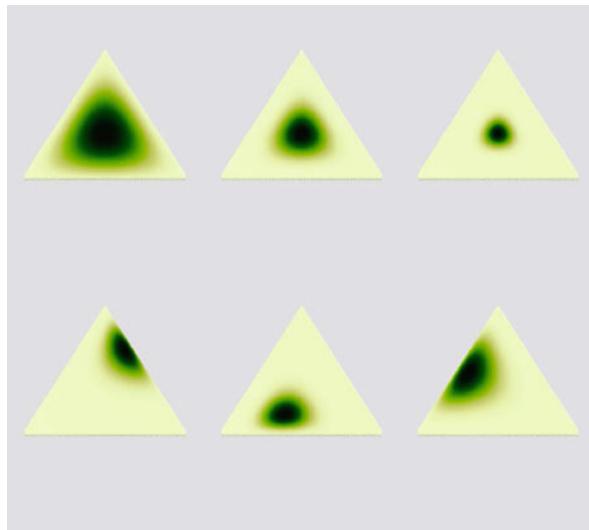
Color diagram 15.1 Geometric interpretation of singular value decomposition

	doc 1	doc 2	doc 3	doc 4
word 1	2	2	4	3
word 2	2	1	5	3
word 3	1	1	2	0
word 4	0	1	2	1

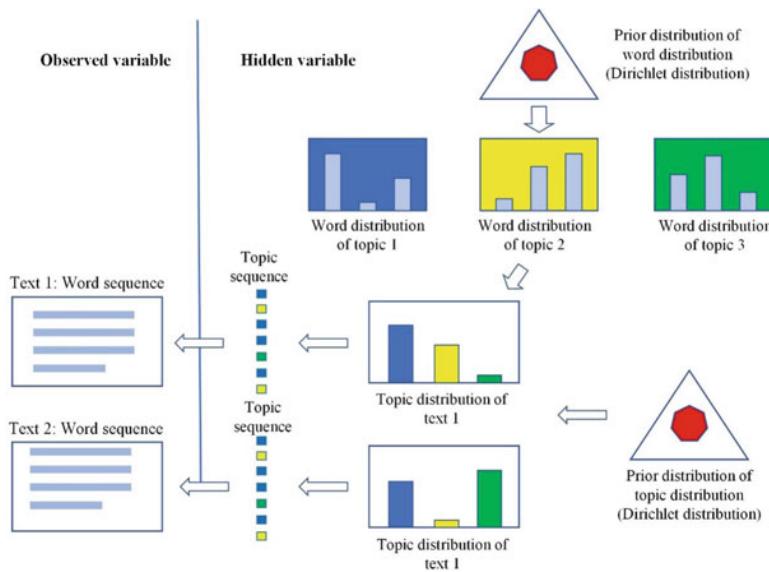
Color diagram 18.1 Intuitive explanation of probabilistic latent semantic analysis



Color diagram 19.1 Accept-reject sampling method



Color diagram 20.1 Dirichlet distribution example



Color diagram 20.3 The text generation process of LDA

# Index

## A

Absolute loss function, 17  
Acceptance distribution, 423–425, 434  
Accept-reject sampling method, 402, 433, 522  
Accuracy, 13, 21, 31, 34, 65, 68, 122, 179, 277, 421, 488, 500, 502, 505, 506  
Action, 9, 10  
Action value function, 10  
Active learning, 4, 10, 11  
AdaBoost algorithm, 179, 180, 182, 183, 185, 187, 189–192, 198, 199  
Additive model, 187–189, 191, 196, 198, 275  
Agglomerative, 293, 300, 301, 308  
Algorithm, 1, 3, 7, 12, 13, 15, 20, 36, 39, 43, 44, 46–52, 55, 56, 60, 61, 63, 65, 67, 70, 72, 77, 80, 81, 85, 87–94, 97–100, 102, 114, 117, 118, 121, 124, 127, 133, 134, 136, 141–145, 148–150, 164–166, 170, 172, 174–176, 179–181, 187–189, 191–194, 196–199, 201–208, 210–214, 216–221, 223, 225–230, 233–235, 238–240, 243, 244, 258, 260, 262, 263, 265–267, 270, 273–276, 278, 280, 283, 285, 286, 289–293, 300–302, 304–306, 308, 309, 311, 327, 337, 352, 355, 356, 360, 362, 363, 365, 372, 377, 380–384, 387, 391, 394, 396–398, 401, 402, 420, 422–432, 434–436, 439, 447, 452, 453, 455–458, 460, 461, 466, 468–470, 473, 482, 486, 487, 489–491, 493–496, 499–502, 504–507

Aperiodic, 415–417, 434, 436, 477–479, 490

Aperiodic graph, 475

Approximation error, 58, 59, 65

## B

Backward, 226, 230–232, 243, 244, 258–260, 263, 270  
Batch learning, 12, 13  
Baum-Welch algorithm, 219, 233, 234, 238, 243  
Bayesian inference, 13  
Bayesian learning, 13, 421, 422, 432, 433, 439, 440, 443, 445, 450, 458, 494  
Bernoulli distribution, 84, 443  
Beta distribution, 436, 440, 442  
Bias, 40, 46, 104  
Binomial distribution, 436, 440, 442, 443  
Binomial logistic regression model, 104  
Boost, 2  
Boosting, 32, 179, 180, 187, 191, 193, 198, 273, 275, 277–280  
Boosting tree, 179, 191–194, 196–198  
Bottom-up, 99, 293, 300, 308  
Broyden-Fletcher-Goldfarb-Shanno, 122, 265, 505–507  
Broyden's algorithm, 506

## C

Categorical distribution, 440, 445–447  
Cell, 56, 79, 94, 287  
Chebyshev distance, 295, 307  
Class, 31, 32, 39–41, 51, 55, 56, 59, 65, 68, 69, 71, 74, 77–79, 85, 88–90, 95, 97, 100, 101, 104, 107, 123, 127–130,

- 134, 135, 152, 153, 176, 179, 180, 183, 186, 187, 192, 199, 273, 274, 279, 282, 283, 285, 286, 291, 293, 294, 297–309, 495, 507
- Classification**, 1, 3, 4, 6, 11–13, 27, 28, 31–34, 36, 39, 55, 56, 59, 64, 65, 67–72, 74, 77–79, 81, 83, 85, 90, 92, 93, 95, 96, 100, 101, 103, 104, 107, 110, 123, 124, 127–130, 132–134, 136, 137, 140–143, 145, 148–150, 152–154, 156, 162, 164, 173–176, 179–187, 190–192, 198, 199, 208, 273–275, 277–279
- Classification and Regression Tree (CART)**, 77, 81, 92, 93, 96–102
- Classifier**, 31, 69, 71, 162, 164, 179–185, 187, 189–192, 198, 199, 275, 278
- Class label**, 56, 67, 71, 128, 129, 144, 273
- Clique**, 250
- Cluster**, 283, 293, 294, 297, 298, 300, 302, 305, 307–309
- Clustering**, 8, 281–283, 286–288, 291, 293, 294, 297, 300–309, 356, 493–495
- Collapsed Gibbs sampling**, 452
- Compact singular value decomposition**, 317, 331
- Complete-data**, 204, 205
- Complete linkage**, 299
- Conditional entropy**, 83–85, 101, 111
- Conditional Random Field (CRF)**, 11, 12, 30, 34, 36, 247, 251–256, 258, 260–262, 264–267, 269, 270, 273, 277–280
- Conjugate distributions**, 443, 444
- Conjugate prior**, 443, 444, 469
- Convex quadratic programming**, 127, 128, 133, 145, 148, 150, 164, 165, 173, 174, 280
- Correlation coefficient**, 296, 297, 300, 307, 346, 347, 350–352, 356, 361
- Cosine**, 296, 300, 307, 367
- Cosine similarity**, 163
- Cost function**, 17, 379
- Covariance matrix**, 295, 298, 299, 307, 337, 340, 342–347, 349, 351, 353–355, 359–363, 425
- Cross validation**, 1, 24–26, 36, 59, 65, 98–100
- D**
- Damping factor**, 480–482, 486, 490
- Data**, 1–14, 20–23, 25, 26, 30–32, 35–37, 45, 47, 49, 50, 60–62, 65, 68, 77, 128, 129, 136, 142, 154, 162, 180, 182, 183, 192–195, 198, 199, 201, 202, 204, 205, 208, 210–220, 223, 234, 235, 238, 241, 243, 244, 260, 262, 281–291, 293–295, 307, 309, 311, 319, 328, 336–339, 352, 354–356, 358, 362–367, 371, 372, 378, 387–391, 394–397, 402, 422, 432, 439, 444, 453, 454, 458, 460, 470, 473
- Davidon-Fletcher-Powell (DFP) algorithm**, 124
- Decision function**, 6, 16, 17, 29–31, 69, 129, 133, 134, 140–143, 145, 147–149, 156, 162, 164, 165, 173–175, 208, 277
- Decision stump**, 191
- Decision tree**, 11, 12, 32, 36, 77–83, 85, 87–102, 191, 192, 199, 273, 274, 277–280
- Decoding**, 225
- Deep learning**, 12
- Detailed balance equation**, 418, 419
- Deterministic model**, 11
- Diameter**, 298, 300, 306, 307
- Dimensionality reduction**, 8, 281, 283, 284, 286–288, 291, 337, 347, 493–495
- Directed edge**, 77, 291, 408, 448, 449, 474–478, 489
- Directed graph**, 284, 289–292, 390, 408, 449, 473–482, 484, 486, 487, 489, 490, 495
- Dirichlet distribution**, 13, 439–448, 461, 463–467, 469, 517, 518
- Discount factor**, 9
- Discriminative approach**, 30, 36
- Discriminative model**, 1, 30, 39, 40, 247, 269, 274–277
- Distance**, 41, 42, 44, 51, 55–58, 60, 64, 65, 129–131, 136, 149, 274, 282, 286, 293–308, 339, 495, 517
- Divergence**, 379–381, 384, 517
- Divisive**, 293, 300, 308
- Dominant eigenvalue**, 484–486, 491
- Dominant eigenvector**, 484, 486, 490, 491
- Dual algorithm**, 137
- Dual problem**, 112–114, 124, 137–139, 141, 142, 145–147, 149, 156, 165, 173–175, 280, 509–512
- Dynamic Bayesian network**, 243
- Dynamic programming**, 92, 163, 239, 243

**E**

- Early stopping, 279  
 Edge, 247–250, 252, 253, 269, 336, 408, 451, 474, 475, 479, 481  
 Empirical conditional entropy, 85, 86  
 Empirical entropy, 85, 86, 90, 91  
 Empirical loss, 18, 41, 150, 278  
 Empirical risk, 18, 19, 24, 25, 27, 42, 59, 150, 278  
 Empirical Risk Minimization (ERM), 18, 19, 23, 27, 37, 65, 192  
 Entropy, 103, 107–110, 124, 215, 270, 379  
 Error rate, 21, 180–185, 190, 198  
 Estimation error, 58, 59, 65  
 Estimation of mathematical expectation, 403, 433  
 Euclidean distance, 56, 57, 65, 286, 294, 295, 300, 301, 306–308  
 Evaluation criterion, 3, 20, 36  
 Evidence, 459, 460, 465–468, 470  
 Evidence Lower Bound (ELBO), 459–462  
 Exchangeable, 449, 450  
 Expectation, 10, 69, 201, 204, 205, 219, 233, 258–260, 394, 395, 401–404, 417–420, 422, 433, 434, 462–465, 518  
 Expectation maximization algorithm, 201  
 Expected loss, 17, 18  
 Exponential loss function, 189, 192, 196, 278–280

**F**

- Factorization, 247, 249–251, 253, 270, 311, 312, 365, 371, 377–379, 382–384, 398, 495, 497  
 Factor loading, 346, 347, 351, 357–359, 361  
 Feature function, 110, 111, 114, 117, 121–123, 253, 254, 259, 260, 262, 265, 268  
 Feature space, 5, 15, 39, 40, 51, 55–57, 60, 62, 65, 79–82, 93, 100, 127–129, 144, 154–156, 163, 168, 175, 274, 278  
 Feature vector, 5, 8, 32, 39, 40, 51, 55, 56, 67, 127, 128, 144, 163, 254, 255, 266, 273, 281, 294, 302, 484  
 Forward, 179, 187–189, 191, 192, 196, 198, 226–229, 231, 232, 243, 244, 258–260, 263, 270, 275  
 Forward-backward algorithm, 258  
 Forward stagewise algorithm, 275

- Frobenius norm, 319, 327, 328, 331–333, 335, 364  
 Full conditional distribution, 425, 426  
 Full singular value decomposition, 317  
 Functional margin, 42, 130, 131

**G**

- Gaussian kernel function, 162  
 Gaussian mixture model, 11, 201, 210, 211, 214, 219, 220, 284, 286, 308, 309, 493–495  
 Generalization ability, 1, 21, 26, 27, 36  
 Generalization error, 26–28  
 Generalization error bound, 27–29  
 Generalized Bregman divergence, 496  
 Generalized Expectation Maximization (GEM), 201, 214, 217–219  
 Generalized Lagrange function, 509  
 Generative approach, 30, 36  
 Generative model, 1, 30, 68, 202, 208, 221, 243, 274, 276–278, 387–395, 397, 398  
 Geometric margin, 129–132, 173  
 Gibbs sampling, 401, 420, 428–432, 435, 437, 439, 451–457, 469, 470, 493–496  
 Gini index, 93, 95–99, 101  
 Global Markov property, 248, 249  
 Gradient boosting, 196, 198  
 Gradient descent, 39, 43, 51, 106, 117, 124, 260, 270, 274, 276, 280, 286, 380, 381, 384, 499, 500  
 Gram matrix, 50, 157, 158, 160–162, 175  
 Graph, 11, 12, 58, 103, 104, 152, 247–252, 269, 270, 284, 285, 289–292, 336, 433, 449, 469, 473, 475, 478, 480, 482, 493, 495  
 Graph analytics, 281, 289, 292, 493

**H**

- Hard clustering, 282, 283, 286, 291, 297, 300, 302, 494  
 Hard margin maximization, 127, 128, 132, 143, 145  
 Hessian matrix, 468, 501–503, 505  
 Hidden Markov Model (HMM), 11, 30, 34, 36, 219, 221–225, 227, 230, 233–235, 238, 239, 243, 244, 251, 258, 273, 276–280  
 Hidden variable, 11, 30, 201, 202, 204, 205, 208, 211, 215, 218, 219, 234, 277, 279, 280, 387, 389, 390, 394, 397,

422, 439, 445, 448, 450–452,  
458–462, 469, 470, 494–496  
Hierarchical clustering, 286, 291, 293,  
300–302, 306, 308, 493–495  
Hilbert space, 127, 128, 154, 157, 160, 161,  
278

Hinge loss function, 127, 150, 152, 278,  
279

Hypothesis space, 3, 6, 8, 16–21, 27–29,  
36, 40, 42

## I

Ill-formed problem, 18

Improved Iterative Scaling (IIS), 103,  
117–119, 121, 124, 260–262, 270,  
274, 276

Incomplete-data, 204, 205, 345, 395

Indicator function, 21, 56, 70

Infinitely exchangeable, 449, 450

Information gain, 82, 83, 85–89, 91, 100,  
101

Information gain ratio, 82, 87, 89–91, 101,  
102

Input space, 5, 6, 8, 14–16, 39, 41, 51, 67,  
127, 128, 153–156, 168, 175, 192,  
277, 283, 284

Instance, 5, 7, 8, 11, 15, 31, 32, 35, 39–41,  
44, 47, 49–52, 55–61, 63–65, 69, 71,  
77–79, 88–90, 94, 100, 104,  
128–132, 135, 136, 142, 143, 149,  
151, 153, 154, 156, 173, 175, 180,  
182, 183, 273, 281–283, 291

Internal node, 77, 78, 89, 93, 98–100

Irreducible, 414, 416–418, 423, 430, 434,  
436, 478

## J

Jensen inequality, 206, 470, 517

## K

Karush-Kuhn-Tucker (KKT) conditions,  
140, 148, 165, 166, 170–172, 175,  
512

*kd-tree*, 60–65

Kernel function, 12, 14, 15, 124, 127, 137,  
153–157, 161, 162, 164, 175, 278

Kernel method, 14, 15, 127, 176, 363

Kernel trick, 127, 153–156, 164, 176, 275

*k*-means clustering, 286, 291, 293,  
302–306, 308, 309, 493–495

*k*-Nearest Neighbor (k-NN), 36, 55, 56, 65,  
273, 277  
Kullback-Leibler divergence (KL  
divergence), 458, 459, 461, 470,  
496, 517

## L

Lagrange duality, 114, 137, 138, 509

Lagrange function, 113, 137, 138, 146, 236,  
342–344, 396, 465

Lagrange multiplier, 137, 165, 236, 237,  
342–344, 396, 509

Language model, 408, 409

Laplace Smoothing, 73

Latent Dirichlet Allocation (LDA), 11, 12,  
286, 288, 292, 439, 440,  
445–452, 455–458, 461, 462, 466,  
468–470, 493–497

Latent Semantic Analysis (LSA), 11, 12,  
286, 288, 292, 311, 365, 369–372,  
374, 375, 379, 383, 384, 387, 393,  
394, 397, 439, 493–497

Latent Semantic Indexing (LSI), 365

Latent variable, 201, 388

Leaf node, 60, 62–64, 77–79, 81, 88–92,  
98, 100, 101, 191, 192, 197

Learning rate, 44

Least squares, 23, 35

$L_p$  distance, 57, 58, 65

Least squares regression tree, 94

Leave-one-out cross validation, 26

Left singular vector, 321–323, 334, 372,  
377

Linear chain, 247, 251–254, 256, 269, 270

Linear chain conditional random field, 252,  
253, 269, 270

Linear classification model, 39, 40, 51, 156

Linear classifier, 40, 127

Linear model, 12, 14, 44, 45, 52, 153, 247,  
260, 269, 274, 276, 278

Linear scan, 60, 64

Linear support vector machine, 48, 127,  
128, 142, 144, 145, 148, 151, 153,  
164, 175, 176

Linkage, 299, 307

Linearly separable data set, 41, 46, 49, 133

Linear support vector machine in linearly  
separable case, 128

Link analysis, 289, 292, 473, 491, 495

Local Markov property, 248, 249

Logarithmic loss function, 17

Logistic distribution, 103, 104, 123

Logistic regression, 11, 12, 30, 32, 36, 103–107, 117, 123, 199, 273, 274, 277–280  
 Log-likelihood loss function, 17, 274–276  
 Log linear model, 117, 253  
 Log odds, 105  
 0-1 Loss function, 13, 17–21, 23, 25, 27, 28, 31, 35, 37, 39, 41–43, 45, 51, 59, 69, 74, 77, 80, 90–93, 98, 99, 102, 152, 187–189, 192, 193, 196–198, 274–276, 278–280, 303, 379–381, 384, 496

**M**

Mahalanobis distance, 295, 300, 307  
 Majority voting rule, 56, 59  
 Manhattan distance, 57, 295, 307  
 Manifold, 283  
 Margin, 127–130, 132–137, 142, 144, 149–152, 170, 171, 173, 174, 176  
 Marginalization, 422, 454–456  
 Markov chain, 221, 223, 243, 290, 401, 406–421, 423, 424, 427–430, 434–436, 473–478, 480, 481, 489–491, 495  
 Markov Chain Monte Carlo (MCMC), 289, 401, 405, 406, 420–423, 425, 428, 429, 434, 435, 452, 458, 493, 494  
 Markov decision process, 9, 10  
 Markov process, 406, 491  
 Markov random field, 12, 247, 249, 251, 269  
 Maximal clique, 250–253, 269  
 Maximization, 69, 70, 114, 116, 117, 127, 128, 132, 147, 201, 204–207, 214, 216–219, 459, 460, 465–467, 510  
 Maximization-maximization algorithm, 214  
 Maximum entropy model, 30, 36, 103, 107, 110–112, 114, 116–118, 122–124, 273, 274, 277–280, 509  
 Maximum likelihood estimation, 14, 18, 19, 36, 37, 70, 72–74, 85, 91, 106, 116–118, 124, 201, 203–205, 208, 218, 220, 225, 233, 247, 251, 260, 270, 274–276, 279, 280, 394, 397, 445, 494  
 Maximum margin method, 133, 173  
 Maximum Posterior Probability Estimation (MAP), 19, 201, 274–276, 279  
 Mean field, 459–462  
 Median, 60, 61  
 Mercer kernel, 162

Minkowski distance, 57, 294, 300, 307  
 Model, 1–3, 5–8, 10–27, 29–37, 39–42, 45, 46, 49–52, 55–59, 65, 67, 68, 74, 77, 79–81, 88, 90–93, 98, 100, 103–111, 114, 117, 118, 121–124, 127, 133, 136, 145, 150, 154, 176, 179, 187–189, 191–194, 197–199, 201–204, 208, 211, 213, 214, 217, 218, 220, 221, 223–226, 229, 232, 235, 237, 238, 240, 243, 247, 249, 251, 253, 254, 260–262, 264, 265, 267, 269, 270, 273–286, 289, 291, 302, 365, 367, 368, 379, 387–394, 397, 398, 408, 410, 422, 432, 433, 436, 439, 440, 445, 446, 448–452, 455, 457, 458, 460–462, 467–470, 473–478, 480, 481, 489, 490, 494–497  
 Model selection, 1, 3, 20–26, 36, 91, 109  
 Monte Carlo integration, 404, 405, 434, 435  
 Monte Carlo method, 401–406, 419, 433, 439, 469  
 Multinomial distribution, 388, 439, 440, 442–448, 461, 469  
 Multi-nominal logistic regression model, 107  
 Mutual information, 85

**N**

Naïve Bayes, 67–71, 74, 273, 274, 277–280  
 Naïve Bayes model, 71, 277  
 Newton method, 106, 117, 122, 124, 260, 264, 265, 270, 274, 276, 280, 501–504, 507  
 Node, 60–65, 77, 78, 239, 240, 247–253, 269, 289, 290, 292, 336, 408, 433, 448, 449, 473–482, 484, 486, 489, 490, 495  
 Non-linear model, 12, 14, 153, 278  
 Non-linear support vector machine, 153, 164, 175  
 Non-negative Matrix Factorization (NMF), 365, 377–379, 384, 493–497  
 Non-parametric model, 12  
 Non-probabilistic model, 153, 277, 494, 496, 497  
 Normalization, 130, 338, 364, 422, 486–488  
 Normalization factor, 114, 123, 181, 191, 250, 253, 254, 256, 257, 442, 518  
 Null space, 315, 322, 324, 326, 514, 515

**O**

- Observable variable, 201, 439, 445, 450, 461  
 Observation sequence, 33, 34, 221–227, 229–231, 233, 234, 241, 243, 251–254, 256, 260, 263, 266–270, 273, 276  
 Occam’s razor, 25  
 Odds, 105  
 Online learning, 12, 13, 470  
 Orthogonal matrices, 311–316, 319, 323–328, 334, 335, 345, 347, 349, 363, 393  
 Outlier, 144  
 Output space, 5, 6, 8, 16, 39, 67, 192, 277, 283, 284  
 Over fitting, 391

**P**

- Pairwise Markov property, 248  
 Parameter space, 16, 397  
 Parametric model, 12  
 Partition, 56, 57, 60–62, 65, 79–82, 88, 90, 93, 94, 100, 303, 304, 306  
 Path, 78, 79, 228, 229, 239–244, 256, 257, 266–268, 270, 475  
 Penalty term, 19, 24  
 Perceptron, 11–13, 30, 36, 39–52, 127, 128, 132, 152, 176, 273, 274, 277, 278, 280  
 Periodic, 415, 417, 418, 475  
 Plate notation, 448, 449, 462  
 Polynomial kernel function, 162  
 Polysemy, 368  
 Positive definite kernel function, 157, 160, 162, 164, 165, 176, 177  
 Positive recurrent, 416–418  
 Potential function, 250, 253  
 Power method, 473, 482, 484, 486, 487, 490, 491, 493–496  
 Precision, 31, 32, 34, 172  
 Primal problem, 137–139, 141, 145, 147–149, 511, 512  
 Prior pseudo-counts, 445  
 Probabilistic context-free grammar, 243  
 Probabilistic graphical model, 11, 12, 248, 270, 284–286, 291, 439, 445, 448, 497  
 Probabilistic Latent Semantic Analysis (PLSA), 11, 12, 286, 288, 292, 387, 388, 390–395, 397, 398, 439, 445, 493–497

**Probabilistic Latent Semantic Indexing (PLSI)**

- (PLSI), 387  
 Probabilistic model, 6, 11–13, 16, 103, 107, 109, 124, 201, 205, 208, 218, 219, 221, 234, 243, 275, 277, 282, 291, 401, 421, 433, 434, 439, 445, 469, 494, 496, 497

**Probabilistic undirected graphical model, 247–251, 269, 270**

- Probability model estimation, 284–286, 288, 291, 494

**Probably Approximately Correct (PAC), 179, 198****Proposal distribution, 402, 423****Pruning, 77, 81, 90–93, 98–102, 274, 280****Q****Q function, 212, 235, 395****Quadratic loss function, 17, 327****Quasi-Newton method, 106, 117, 122, 124, 260, 264, 265, 274, 276, 280, 501, 503, 504, 507****R****Radial basis function, 162****Random sampling, 401–403, 420, 430, 431, 433, 435, 456, 458, 470, 495, 496****Random walk, 290, 401, 420, 421, 423, 425, 429, 434, 473–478, 480, 481, 489, 490****Recall, 31, 32, 34****Rectangular diagonal matrix, 311–313, 315, 323, 334****Reducible, 414, 436****Region, 56, 59–61, 63–65, 192, 197****Regression, 1, 3, 6, 22, 25, 31, 34–36, 55, 64, 77, 92–94, 102, 124, 191–198, 208, 273, 274****Regularization, 1, 19, 24, 25, 36, 151, 175, 278, 279****Regularizer, 19, 24, 25****Reinforcement learning, 1, 3, 4, 9, 10, 13, 15, 36, 291****Reproducing Kernel Hilbert Space (RKHS), 160****Reversible Markov chain, 418–420, 423, 434–436****Reward, 9, 10****Reward function, 9****Right singular vector, 312, 321, 322, 372, 377****Risk function, 17–19, 69, 278**

- Root node, 60, 61, 63, 64, 78, 81, 82, 88, 96, 98–101, 191, 197
- S**
- Sample, 5, 7, 8, 12, 14, 18–21, 25, 27, 28, 30, 31, 33, 42, 50, 52, 70, 71, 79, 82, 85, 86, 88–90, 95–97, 100, 101, 110, 128, 130–132, 134–136, 141, 142, 144, 145, 149, 151, 152, 156, 165, 170, 180, 182, 183, 190, 191, 194, 208, 234, 274, 281–284, 286–288, 291, 293–308, 337–339, 352–358, 360–364, 401–406, 418–421, 423, 426–435, 444, 452, 455–457, 469, 495
- Scatter matrix, 298, 307
- Semi-supervised learning, 4, 10, 11
- Separating hyperplane, 39–41, 44–46, 48–51, 128–130, 132–137, 140, 141, 143, 145, 147–150, 156, 173–175, 274, 275
- Sequential Minimal Optimization (SMO), 127, 165, 166, 170–172, 175, 176, 275, 280
- S-fold cross-validation, 26
- Sigmoid curve, 104
- Signed distance, 131
- Similarity, 14, 57, 282, 293–297, 300, 303, 307–309, 336, 365–369, 371, 382–384, 445, 458, 470, 517
- Simplex, 109, 392, 393, 397, 398, 441
- Single-component Metropolis-Hastings, 427, 436
- Single linkage, 299
- Singular value, 312, 314, 318, 319, 321–326, 330, 331, 333–335, 360, 363, 372, 377
- Singular Value Decomposition (SVD), 287, 311–324, 326–331, 334–337, 355, 358, 360, 363, 365, 372, 374, 383, 384, 393, 493–495
- Soft clustering, 282, 283, 285, 286, 291, 297, 494
- Soft margin maximization, 127, 132, 144, 145, 150, 164, 275
- Span, 513
- Splitting point, 60, 61, 63, 94–98
- Splitting variable, 94
- Squared Euclidean distance, 303
- State, 9, 10, 107, 124, 221–225, 227–234, 238–243, 253–257, 260, 261, 263, 264, 270, 406–411, 413–421, 423, 425–427, 429, 434–436, 475, 476, 478, 480, 489
- State sequence, 33, 221–226, 228, 233, 234, 238, 239, 241, 243, 251, 252, 256, 257, 270, 273, 276
- State value function, 10
- Statistical learning, 176, 281, 291, 509
- Statistical machine learning, 1
- Statistical simulation method, 401
- Steepest descent, 196, 499
- Stochastic gradient descent, 13, 43, 51, 274
- Stochastic matrix, 407, 476, 479, 491
- Stochastic process, 406
- Strategy, 1, 3, 9, 10, 15, 17–20, 23, 24, 36, 39, 41, 42, 51, 127, 128, 150, 173, 199, 274–276, 278, 279, 285, 286, 302, 303, 397, 495
- String kernel function, 162, 163
- Strongly connected graph, 475
- Strongly learnable, 179, 198
- Structural Risk Minimization (SRM), 18, 19
- Sub-optimal, 80, 100
- Supervised learning, 1, 3–8, 10, 11, 13, 15–20, 29–31, 33, 34, 36, 104, 208, 233, 234, 273, 274, 278, 280, 281, 285–287, 291, 494
- Support vector, 135–137, 141–143, 149, 150, 164, 170–173, 175, 176
- Support Vector Machines (SVM), 11, 12, 14, 15, 30, 32, 36, 39, 48, 49, 127–129, 132, 133, 136, 137, 141–145, 148, 150, 152–154, 156, 162, 164, 165, 173–176, 199, 270, 273, 275, 277–280, 509
- Surrogate loss function, 152, 278
- Synonymy, 368
- T**
- Tagging, 1, 3, 6, 11, 31, 33, 34, 36
- Tensor, 335
- Term Frequency-Inverse Document Frequency (TF-IDF), 366, 367, 382
- Test data, 3, 5–7, 19, 36
- Test error, 20, 21, 23, 24, 26
- Test set, 6, 7, 25, 26
- Time homogenous Markov chain, 434
- Top-down, 293, 300, 308, 309
- Topic, 36, 288, 289, 291, 365, 368–375, 377, 379, 382–384, 387–394, 396–398, 439, 445–457, 461, 463–467, 469, 470, 491, 496, 497

- Topic document matrix, 370  
 Topic modeling, 281, 288, 289, 291, 365,  
   493–495  
 Topic vector space, 365, 368–374, 377,  
   379, 383  
 Topic vector space model, 365, 369, 383  
 Trace, 346, 347, 349  
 Training data, 1, 3, 5, 7, 8, 12, 19, 21, 23,  
   31, 35, 36, 39, 43, 48, 60, 71, 73, 77,  
   90, 91, 93, 97, 98, 100, 102, 110,  
   116, 117, 127, 132, 142, 144, 154,  
   173, 174, 179–184, 187, 190,  
   192–196, 198, 208, 233, 234, 261,  
   270, 281, 283–285, 291  
 Training error, 20, 21, 23, 24, 28, 36, 179,  
   185–187, 198  
 Training set, 5, 25, 26, 41, 44, 48–50, 56,  
   60, 65, 88, 90, 101, 141, 164, 170  
 Transition kernel, 413, 414, 420, 423, 430,  
   434  
 Transition probability, 9, 10, 221–225, 231,  
   239, 406–418, 428, 476, 477, 480,  
   481, 489, 490  
 Tree, 55, 60, 62, 77  
 Trial and error, 9  
 Truncated singular value decomposition,  
   317, 318, 333, 358–360, 363, 372,  
   373, 375, 383
- U**
- Unsupervised learning, 1, 3, 4, 7, 8, 11, 14,  
   15, 36, 208, 219, 220, 233, 234, 243,  
   277–279, 281, 282, 285, 286,  
   290–293, 337, 365, 387, 473,  
   493–495
- V**
- Validation set, 25  
 Value function, 10  
 Variational distribution, 458, 459, 461, 462,  
   470  
 Variational EM algorithm, 439, 458, 460,  
   461, 468–470, 495, 496  
 Variational inference, 451, 452, 458–460,  
   494, 495  
 Vector Space Model (VSM), 366  
 Viterbi algorithm, 238–240, 243, 244,  
   266–268, 270
- W**
- Weakly learnable, 179, 198  
 Weight, 40, 46, 47, 114, 123, 179–184, 190,  
   191, 198, 253–255, 267–269, 336,  
   366, 367, 369, 370, 372, 382, 383,  
   474  
 Weight vector, 40, 47, 104, 105, 117, 254,  
   255  
 Word-document matrix, 366  
 Word-topic matrix, 370, 383  
 Word vector space model, 365–369, 382