

Taeho Jo

Machine Learning Foundations

Supervised, Unsupervised, and
Advanced Learning



Springer

Machine Learning Foundations

Taeho Jo

Machine Learning Foundations

Supervised, Unsupervised, and Advanced
Learning



Taeho Jo
Hongik University
Garosuro Cheongju, Korea (Republic of)

ISBN 978-3-030-65899-1 ISBN 978-3-030-65900-4 (eBook)
<https://doi.org/10.1007/978-3-030-65900-4>

© Springer Nature Switzerland AG 2021

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, 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 publisher, 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 publisher 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 publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

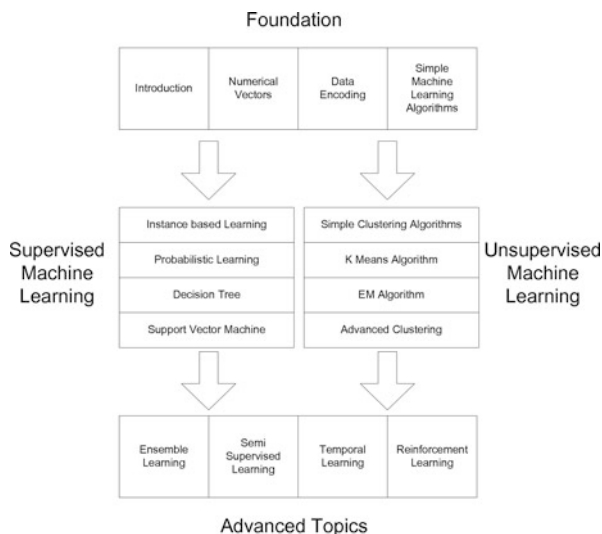
Preface

This book is concerned with the concepts, the theories, and the algorithms of machine learning. In Part **I**, we provide the fundamental knowledge about machine learning by exploring learning theories, evaluation schemes, and simple machine learning algorithms. In Parts **II** and **III**, we describe the supervised learning algorithms as the approaches to the classification and regression tasks, and the unsupervised ones as those to the clustering tasks. In Part **IV**, we cover the special types of learning algorithms, and the mixture of the supervised and unsupervised algorithms as the further study. Readers need the basic level of knowledge about linear algebra and vector calculus for understanding the machine learning algorithms where input data is always given as a numerical vector.

We mention some agenda that provide motivations for writing this book as the president in the company for consulting the artificial intelligence techniques, Alpha Lab AI. The machine learning algorithms became very popular approaches to data mining tasks, because of their flexibility and adaptability, compared with the rule based approaches. In the rule based systems, symbolic rules must be provided, manually, whereas the machine learning algorithms generate the rules automatically once previous examples are given; the machine learning based approaches are more automatic than the rule based ones. The machine learning algorithms are applied to tasks in various disciplines such as control system engineering, chemical engineering, and business administration and management. In this book, we describe the machine learning algorithms with the assumption that the input data is given as a numerical vector.

This book consists of four parts: foundation, supervised learning, unsupervised learning, and advanced topics. In the first part, we cover the linear algebra, the raw data types, and the primitive machine learning algorithms as the general aspects of machine learning. In the second part, we describe four kinds of supervised machine learning algorithms: K Nearest Neighbor, Probabilistic Learning, the Decision Tree, and the SVM (Support Vector Machine). In the third part, we focus on three kinds of unsupervised learning algorithms: AHC (Agglomerative Hierarchical Clustering) Algorithm, K Means Algorithm, and EM (Expectation Maximization) Algorithm. In the last part, we explore the advanced topics on the machine learning:

Fig. 1 Organization of the four parts



the combination of machine learning algorithms, the mixture of the supervised and unsupervised learning, the temporal sequence learning, and the reinforcement learning.

This book is written intended for three groups of readers: students, professors, and researchers. Senior undergraduate and graduate students will be able to study the contents of machine learning by themselves with this book. For lecturers and professors in universities, this book may be used as the main material for providing lectures on machine learning. This book is useful for researchers and system developers in industrial organizations for developing intelligent systems using the machine learning algorithms as tools. Therefore, this book is practical for people in both worlds: the academic world and the industrial one, to study the machine learning.

The four parts of the book are outlined in Fig. 1.

Part I: Foundation

The first part of this book is concerned with the linear algebra, the raw data types before encoding them into numerical vectors, and the simple machine learning algorithms. Because this part provides the foundation for studying the machine learning, we need to understand the linear algebra. We study the raw data types, such as the relational data, the textual data, and the image data, and the process of encoding them into numerical vectors. We study some primitive machine learning algorithms for warming up to study the main ones that are covered in Parts II and III.

Part I is intended to provide the basic foundations for understanding the machine learning algorithms.

Chapter 1 is concerned with the overview of machine learning algorithms in the general aspect. We begin with the overview of tasks, such as the data classification, the data clustering, and the regression, to which we are able to apply the machine learning algorithms in the functional view. We describe briefly the four types of machine learning algorithms: the supervised, the unsupervised, the reinforced, and the semi-supervised learning. We explore other areas that are related to the current area, by comparing it with them. Therefore, Chap. 1 covers the entire aspect of machine learning algorithms with their functions, types, and related areas.

Chapter 2 is concerned with the linear algebra, which is the mathematics that provides the foundation for understanding the machine learning algorithms. It is assumed that the input data is given as a numerical vector, while applying machine learning algorithms to real tasks. Chapter 2 covers mainly the operations and the mathematical properties of the vector and the matrix, and the relation between them. We will mention the two schemes of reducing the dimension of the input vector: the singular value decomposition and the principal component analysis. If you are familiar with the linear algebra, study only the two dimension reduction schemes.

Chapter 3 is concerned with the raw data types, which are in the status before encoding them into numerical vectors. The relational data that consists of records is the typical raw data type, and it is relatively easy to encode them into numerical vectors. We mention the textual data as the most popular raw data type in the real world and study the process of indexing a text into a list of words and encoding it into a numerical vector. We also study the image data, which is one more kind of raw data, and the process of encoding them into numerical vectors. Because it is assumed that a numerical vector is given as the input, while applying machine learning algorithms to real tasks, it is very important to encode the raw data into numerical vectors.

Chapter 4 is concerned with the binary classification task and some simple machine learning algorithms. While studying the machine learning algorithms, it is assumed that the given task is a binary classification, and the regression or the multiple classification may be decomposed into binary classifications. Some simple machine learning algorithms, which are given as threshold rules or hypercubes, will be mentioned for helping understanding the machine learning algorithms. As a typical type of machine learning algorithm, the linear classifier is expressed as a linear equation. This chapter is intended to provide the warm-up for understanding the main machine learning algorithms, which are covered in Parts II and III.

Part II: Supervised Learning

Part II is concerned with the four representative supervised machine learning algorithms: the KNN (K Nearest Neighbor), the decision tree, the probabilistic learning, and the SVM (Support Vector Machine). The concept learning that was

mentioned in the previous part is the only toy machine learning algorithm that provides the basic concept, rather than a real one, so we need to study the real versions that are applicable to real tasks. This part begins with the KNN as the simple and practical machine learning algorithm in Chap. 5. In the subsequent chapter, we describe the more advanced machine learning algorithms: the decision tree, the probabilistic learning, and the SVM. Therefore, this part is intended to study the four supervised learning algorithms, chapter by chapter.

Chapter 5 is concerned with the KNN (K Nearest Neighbor) as the simple and practical supervised learning algorithm. As its background, we mention the case based reasoning and lazy learning, which underlie in the KNN. Based on the background, we explain in detail the process of carrying out the classifications and regressions of data items by the KNN. We present also the modified versions of KNN as its variants. In this chapter, we assume that the KNN is the supervised learning algorithm, but we cover its unsupervised version in the next part.

Chapter 6 is concerned with the probabilistic learning, which is based on conditional probabilities of categories, given an example. We mention the probability theory, which is called Bayes Theorem, in order to provide the background for understanding the chapter. We describe in detail some probabilistic classifiers such as Bayes Classifier and Naive Bayes as the popular and simple machine learning algorithms. We cover the Bayesian Learning as the more advanced learning method than the two probabilistic learning algorithms. Among the probabilistic learning algorithms, the Naive Bayes is used as the most popular classifier in real classification Tasks, including text categorization.

Chapter 7 is concerned with the decision tree that is characterized as one of the symbolic machine learning algorithms. We begin this chapter by explaining the information gain as the important measure for implementing the decision tree. Afterward, we describe in detail the process of learning training examples for building the decision tree and classifying novice examples. We present some decision tree variants, such as the random forests, the decision list, and the decision graph. The decision tree is the most popular among machine learning algorithms in spite of its less performance than other machine learning algorithms because of its symbolic characterization.

Chapter 8 is concerned with the SVM that defines dual parallel linear boundaries among classes. We study the Perceptron that is a typical linear classifier and the basis for deriving the SVM. In the main part, we cover the classification process, the constraints, and the learning process of the SVM. We survey some variants of the SVM that are expansions of the standard SVM. The SVM is applicable to a nonlinear classification problem, robustly, as the most popular machine learning algorithm.

Part III: Unsupervised Learning

Part III is concerned with the unsupervised machine learning algorithms as clustering tools. As simple clustering algorithms, we study the AHC algorithm, the divisive clustering algorithm, and the online clustering algorithm. We continue the study of the unsupervised machine learning algorithms with the k means algorithm and its variants. The EM algorithm is considered as a clustering paradigm, rather than a specific algorithm, and some specific versions will be mentioned. We mention the clustering algorithms as the unsupervised learning algorithms and the clustering metric, which is the metric for evaluating them.

Chapter 9 is concerned with the three simple clustering algorithms. We mention the AHC algorithm where the clustering proceeds in the bottom-up direction. We mention the divisive algorithm where the clustering proceeds in the top-down direction as another kind of simple clustering algorithm. The online linear clustering, which is a fast and simple clustering algorithm, has almost linear complexity in case of making a small number of clusters. We explore other sophisticated clustering algorithms in the subsequent chapters.

Chapter 10 is concerned with the k means algorithm as the most popular clustering algorithm. This chapter begins with the unsupervised version of the KNN algorithm. With respect to the clustering process, we study the two main versions of the k means algorithms: the crisp k means algorithm and the fuzzy k means algorithm. The k medoid algorithm is mentioned as a variant of the k means algorithm, and the strategies of selecting representative items are focused. Note that the k means algorithm is the simplest version of EM algorithm, and it is covered in the next chapter.

Chapter 11 is concerned with the EM algorithm, which is a frame of clustering data items, rather than a specific algorithm. We will survey some probabilistic distributions as the basis for understanding the EM algorithm. Assuming that each cluster has its own normal distribution, we describe in detail the EM algorithm, which is extended from the k means algorithm. We mention the variations of EM algorithm and the semi-supervised learning algorithm with its combination with the Naive Bayes. In Chaps. 9–11, we cover the kinds of clustering algorithms, and in Chap. 12, we cover the clustering index as a clustering evaluation metric.

Chapter 12 is concerned with the clustering index, which is the metric for evaluating clustering algorithms. There are two kinds of similarities in evaluating clustering algorithms: the intra-cluster similarity that should be maximized and the inter-cluster similarity that should be minimized. The clustering index is the combination of these two kinds of similarities, following the style of combining the recall and the precision into the F1 measure. The clustering index may be used for tuning the external parameters of clustering algorithms, as well as for evaluating them. The clustering index was initially proposed for evaluating the current data organization.

Part IV: Advanced Topics

Part IV is concerned with the advanced topics in the machine learning. We explore various schemes of combining multiple machine learning algorithms. We consider some advanced supervised learning including the semi-supervised learning. We mention the two additional kinds of machine learning: the temporal learning and the reinforced learning. Part IV is intended to describe in detail the combination schemes and the special type of machine learning algorithms.

Chapter 13 is concerned with the schemes of combining multiple machine learning algorithms with each other. We mention the four main combination schemes, depending on the role of the coordinator as the start of this chapter. Afterward, we describe the meta-learning for reinforcing the combination schemes. We present some partitions for implementing the ensemble learning. This chapter focuses on the combination of supervised learning algorithms for classifying data items.

Chapter 14 is concerned with the advanced supervised learning, including the semi-supervised learning. We start with the three versions of the Kohonen Networks: the unsupervised version, the supervised version, and the semi-supervised version. Afterward, as another kind of semi-supervised learning algorithm, we present some models with the combination of the supervised learning algorithm and the unsupervised learning algorithm. We explore some techniques for advancing the supervised learning, including the resampling and the co-learning. There are two kinds of semi-supervised learning: a single model that is derived from an unsupervised learning algorithm and a combined model that consists of a supervised learning algorithm and an unsupervised one.

Chapter 15 is concerned with the Hidden Markov Model as a special type of machine learning algorithm. We describe Discrete Markov Model as the background for understanding the Hidden Markov Model. In Section 3, we describe entirely the Hidden Markov Model that performs the following three tasks: computing the probability of a state sequence, generating state sequence given an observation sequence, and estimating its parameters from a training set. We mention the text topic analysis as a typical task to which the Hidden Markov Model is applied. This chapter covers the special machine learning algorithm that considers the temporal sequence of input values, differently from other machine learning algorithms.

Chapter 16 is concerned with the reinforced learning, which is the alternative kind of machine learning to both the supervised learning and the unsupervised learning. The reinforced learning is one where the critics is given after making the decision to the input, instead of its target label. The table of entries, each of which consists of an input vector and reward scores corresponding to decisions, is called Q table. To an initial input, a decision is made at random, and it is done subsequently based on the Q table. We consider the case where the reward is variable to the same input, depending on the application area. This part provides the foundation for understanding the machine learning algorithms. In this part, we need to understand the linear algebra that is concerned with the vector and the matrix as

the mathematical foundation. Because the input of the machine learning algorithms is always given as a numerical vector, we need to understand how to encode the raw data such as relational data, textual data, and image data into numerical vectors. We will mention some simple machine learning algorithms assuming the simple binary classification as the given problem to which we apply them. This part is intended to study the foundations of the machine learning algorithms such as vector and matrix, encoding process, and the simple machine learning algorithms.

This part consists of four chapters. In Chap. 1, we present the outline of machine learning algorithms and the related areas. In Chap. 2, we study the vector and the matrix that are covered in the linear algebra as the means of modeling machine learning algorithms. In Chap. 3, because the input data is always given as a vector for the machine learning algorithms, we deal with the process of encoding the raw data into numerical vectors. In Chap. 4, we mention some primitive machine learning algorithms, assuming that the given problem is given as a binary classification.

President, Alpha Lab AI
Garosuro, Cheongju

Taeho Jo

Contents

Part I Foundation

1	Introduction	3
1.1	Definition of Machine Learning	3
1.2	Application Areas	4
1.2.1	Classification	4
1.2.2	Regression	6
1.2.3	Clustering	7
1.2.4	Hybrid Tasks	8
1.3	Machine Learning Types	11
1.3.1	Supervised Learning	11
1.3.2	Unsupervised Learning	12
1.3.3	Semi-supervised Learning	14
1.3.4	Reinforcement Learning	15
1.4	Related Areas	16
1.4.1	Artificial Intelligence	17
1.4.2	Neural Networks	18
1.4.3	Data Mining	19
1.4.4	Soft Computing	20
1.5	Summary and Further Discussions	21
	References	22
2	Numerical Vectors	23
2.1	Introduction	23
2.2	Operations on Numerical Vectors	24
2.2.1	Definition	24
2.2.2	Basic Operations	25
2.2.3	Inner Product	26
2.2.4	Linear Independence	28
2.3	Operations on Matrices	29
2.3.1	Definition	30
2.3.2	Basic Operations	31

2.3.3	Multiplication	32
2.3.4	Inverse Matrix	35
2.4	Vector and Matrix	37
2.4.1	Determinant	38
2.4.2	Eigen Value and Vector	40
2.4.3	Singular Value Decomposition	42
2.4.4	Principal Component Analysis	43
2.5	Summary and Further Discussions	45
3	Data Encoding	47
3.1	Introduction	47
3.2	Relational Data	48
3.2.1	Basic Concepts	48
3.2.2	Relational Database	50
3.2.3	Encoding Process	52
3.2.4	Encoding Issues	53
3.3	Textual Data	55
3.3.1	Text Indexing	55
3.3.2	Text Encoding	58
3.3.3	Dimension Reduction	60
3.3.4	Encoding Issues	61
3.4	Image Data	62
3.4.1	Image File Formats	63
3.4.2	Image Matrix	63
3.4.3	Encoding Process	65
3.4.4	Encoding Issues	66
3.5	Summary and Further Discussions	67
	References	67
4	Simple Machine Learning Algorithms	69
4.1	Introduction	69
4.2	Classification	70
4.2.1	Binary Classification	70
4.2.2	Multiple Classification	71
4.2.3	Regression	73
4.2.4	Problem Decomposition	74
4.3	Simple Classifiers	76
4.3.1	Threshold Rule	76
4.3.2	Rectangle	77
4.3.3	Hyperplane	79
4.3.4	Matching Algorithm	80
4.4	Linear Classifiers	82
4.4.1	Linear Separability	82
4.4.2	Hyperplane Equation	84
4.4.3	Linear Classification	86
4.4.4	Perceptron	87

4.5	Summary and Further Discussions	89
	References	90

Part II Supervised Learning

5	Instance Based Learning	93
5.1	Introduction	93
5.2	Primitive Instance Based Learning	94
5.2.1	Look-Up Example	94
5.2.2	Rule Based Approach	95
5.2.3	Example Similarity	98
5.2.4	One Nearest Neighbor	99
5.3	Classification Process	100
5.3.1	Notations	100
5.3.2	Nearest Neighbors	101
5.3.3	Voting	103
5.3.4	Attribute Discriminations	105
5.4	Variants	106
5.4.1	Dynamic Nearest Neighbor	106
5.4.2	Concentric Nearest Neighbor	108
5.4.3	Hierarchical Nearest Neighbor	110
5.4.4	Hub Examples	112
5.5	Summary and Further Discussions	113
	References	115
6	Probabilistic Learning	117
6.1	Introduction	117
6.2	Bayes Classifier	118
6.2.1	Probabilities	118
6.2.2	Bayes Rule	120
6.2.3	Gaussian Distribution	121
6.2.4	Classification	123
6.3	Naive Bayes	124
6.3.1	Classification	124
6.3.2	Learning	126
6.3.3	Variants	128
6.3.4	Application to Text Classification	129
6.4	Bayesian Learning	131
6.4.1	Bayesian Networks	131
6.4.2	Causal Relation	133
6.4.3	Learning Process	135
6.4.4	Comparisons	136
6.5	Summary and Further Discussions	138
	References	139

7	Decision Tree	141
7.1	Introduction	141
7.2	Classification Process	142
7.2.1	Basic Structure	142
7.2.2	Toy Examples	144
7.2.3	Text Classification	147
7.2.4	Rule Extraction	148
7.3	Learning Process	150
7.3.1	Preprocessing	151
7.3.2	Root Node	152
7.3.3	Interior Nodes	154
7.3.4	Pruning	155
7.4	Variants	156
7.4.1	Regression Version	156
7.4.2	Decision List	158
7.4.3	Random Forest	160
7.4.4	Decision Graph	162
7.5	Summary and Further Discussions	164
	Reference	165
8	Support Vector Machine	167
8.1	Introduction	167
8.2	Classification Process	168
8.2.1	Linear Classifier	168
8.2.2	Kernel Functions	170
8.2.3	Lagrange Multipliers	171
8.2.4	Generalization	173
8.3	Learning Process	174
8.3.1	Primal Problem	174
8.3.2	Dual Problem	176
8.3.3	SMO Algorithm	177
8.3.4	Other Optimization Schemes	180
8.4	Variants	181
8.4.1	Fuzzy SVM	181
8.4.2	Pairwise SVM	182
8.4.3	LMS SVM	183
8.4.4	Sparse SVM	185
8.5	Summary and Further Discussions	186
	References	187
 Part III Unsupervised Learning		
9	Simple Clustering Algorithms	191
9.1	Introduction	191
9.2	AHC Algorithm	192
9.2.1	Cluster Similarity	192

9.2.2	Initial Version	194
9.2.3	Fuzzy Clustering	195
9.2.4	Variants	198
9.3	Divisive Clustering Algorithm	200
9.3.1	Binary Clustering	200
9.3.2	Evolutionary Binary Clustering	202
9.3.3	Standard Version	204
9.3.4	Variants	205
9.4	Online Linear Clustering Algorithm	207
9.4.1	Representative Selection Scheme	208
9.4.2	Initial Version	209
9.4.3	Fuzzy Clustering	210
9.4.4	Variants	212
9.5	Summary and Further Discussions	214
	References	215
10	K Means Algorithm	217
10.1	Introduction	217
10.2	Supervised and Unsupervised Learning	218
10.2.1	Learning Paradigm Transition	218
10.2.2	Unsupervised KNN	219
10.2.3	Semi-supervised KNN	221
10.2.4	Dynamic Data Organization	222
10.3	Clustering Process	224
10.3.1	Initialization	224
10.3.2	Hard Clustering	225
10.3.3	Fuzzy Clustering	227
10.3.4	Hierarchical Clustering	229
10.4	Variants	230
10.4.1	K Medoid Algorithm	230
10.4.2	Dynamic K Means Algorithm	233
10.4.3	Semi-supervised Version	234
10.4.4	Constraint Clustering	236
10.5	Summary and Further Discussions	239
	References	240
11	EM Algorithm	241
11.1	Introduction	241
11.2	Cluster Distributions	242
11.2.1	Uniform Distribution	242
11.2.2	Gaussian Distribution	244
11.2.3	Poisson Distribution	245
11.2.4	Fuzzy Distributions	247
11.3	Clustering Process	249
11.3.1	Initialization	249
11.3.2	E-Step	250

11.3.3	M-Step	252
11.3.4	Issues	253
11.4	Semi-Supervised Learning: Text Classification	254
11.4.1	Semi-Supervised Learning	254
11.4.2	Initialization	255
11.4.3	Likelihood Estimation	256
11.4.4	Parameter Estimation	257
11.5	Summary and Further Discussions	259
	References	260
12	Advanced Clustering	261
12.1	Introduction	261
12.2	Cluster Index	262
12.2.1	Computation Process	262
12.2.2	Hard Clustering Evaluation	264
12.2.3	Fuzzy Clustering Evaluation	265
12.2.4	Hierarchical Clustering Evaluation	267
12.3	Parameter Tuning	268
12.3.1	Clustering Index to Unlabeled Items	268
12.3.2	Simple Clustering Algorithms	269
12.3.3	K Means Algorithm	270
12.3.4	Evolutionary Clustering	271
12.4	Clustering Governance	272
12.4.1	Cluster Naming	272
12.4.2	Cluster Maintenance	274
12.4.3	Multiple Viewed Clustering	276
12.4.4	Clustering Results Integration	278
12.5	Summary and Further Discussions	280
	References	282
 Part IV Advanced Topics		
13	Ensemble Learning	285
13.1	Introduction	285
13.2	Combination Schemes	286
13.2.1	Voting	286
13.2.2	Expert Gates	287
13.2.3	Cascading	289
13.2.4	Cellular Model	290
13.3	Meta-learning	291
13.3.1	Voting	292
13.3.2	Expert Gates	294
13.3.3	Cascading	296
13.3.4	Cellular Model	298
13.4	Partition	300
13.4.1	Training Set Partition	300

13.4.2	Attribute Set Partition	302
13.4.3	Architecture Partition	303
13.4.4	Parallel and Distributed Learning	304
13.5	Summary and Further Discussions	306
	References	307
14	Semi-supervised Learning	309
14.1	Introduction	309
14.2	Kohonen Networks	310
14.2.1	Initial Version	310
14.2.2	Learning Vector Quantization	313
14.2.3	Semi-supervised Version	315
14.2.4	Kohonen Networks vs. K Means Algorithm	317
14.3	Combined Learning Algorithms	318
14.3.1	Combination Paradigms	318
14.3.2	Simple Learning Algorithms	320
14.3.3	K Means Algorithm + KNN Algorithm	323
14.3.4	EM Algorithm + Naive Bayes	324
14.4	Advanced Supervised Learning	325
14.4.1	Resampling	326
14.4.2	Virtual Training Example	327
14.4.3	Co-Learning	329
14.4.4	Incremental Learning	331
14.5	Summary and Further Discussions	332
	References	334
15	Temporal Learning	335
15.1	Introduction	335
15.2	Discrete Markov Model	336
15.2.1	State Diagram	336
15.2.2	State Transition Probability	337
15.2.3	State Path Probability	338
15.2.4	Application to Time Series Prediction	340
15.3	Hidden Markov Model	341
15.3.1	Initial Parameters	342
15.3.2	Observation Sequence Probability	343
15.3.3	State Sequence Estimation	346
15.3.4	HMM Learning	349
15.4	Text Topic Analysis	351
15.4.1	Task Specification	351
15.4.2	Sampling	353
15.4.3	Learning	354
15.4.4	Topic Sequence	356
15.5	Summary and Further Discussions	357
	References	358

16	Reinforcement Learning	359
16.1	Introduction	359
16.2	Simple Reinforcement Learning	360
16.2.1	Single Example	360
16.2.2	Classification	362
16.2.3	Regression	364
16.2.4	Autonomous Moving	366
16.3	Q Learning	368
16.3.1	Q Table	368
16.3.2	Finite State	370
16.3.3	Infinite State	371
16.3.4	Stochastic Reward	374
16.4	Advanced Reinforcement Learning	375
16.4.1	Ensemble Reinforcement Learning	376
16.4.2	Reinforcement + Supervised	378
16.4.3	Reinforcement + Unsupervised	380
16.4.4	Environment Prediction	382
16.5	Summary and Further Discussions	384
	Index	385

Part I

Foundation

Chapter 1

Introduction



1.1 Definition of Machine Learning

The machine learning is defined as the computation paradigm where the capacity for solving the given problem is built by previous examples. The basic idea of inventing the machine learning is the case based reasoning which is the process of reasoning the problem by referring similar previous cases. The previous examples which are used for building the capacity are called training examples, and the process of doing so is called learning. The process of solving the real problems after learning the training examples is called generalization. This section is intended to study the overview of machine learning algorithm, with respect to the application areas and the learning paradigms.

Let us mention the tasks to which we apply the machine learning algorithms. The classification is the process of classifying items into one or some among the predefined categories as the most popular task. The regression is the process of estimating a continuous value or values based on the current input factors, as another task. The clustering is the process of segmenting an entire group into subgroups based on similarities among items, as the task to which we apply the unsupervised learning algorithms. The reinforcement learning algorithms are needed for implementing an autonomous agent or agents.

The supervised learning may be mentioned as the first type of machine learning. In this learning type, it is assumed that all of previous examples, called training examples, are labeled. The supervised learning is the learning paradigm of building the capacity of solving problems based labels of training examples. The supervised learning is usually applied to the data classifications and the regressions. The supervised learning is the type which is mentioned most frequently in previous literatures.

The unsupervised learning is mentioned as the second type of learning paradigm and treated together with the supervised learning, mainly. It is assumed that all of training examples are unlabeled, in this type of learning. Because unlabeled examples are learned depending on their similarities, it is important to define the

similarity metric among them. The data clustering is the typical task to which the unsupervised learning algorithms are applied. It is possible to transit learning algorithms between the two learning types.

This chapter is intended to provide the overview of machine learning together with its related areas. We need to understand the tasks such as the classification, the regression, and the clustering, to which we apply the machine learning algorithms. We need to understand the machine learning paradigms, such as the supervised learning, the unsupervised learning, and the reinforcement learning. We also understand the areas which are related with the machine learning, such as data mining and neural networks. We will obtain the ability to do the further discussions about this chapter.

1.2 Application Areas

This section is concerned with the areas to which we apply the machine learning algorithms and consists of the four subsections. In Sects. 1.2.1 and 1.2.2, we cover the classification and the regression as the areas to which we apply the supervised learning algorithms. In Sect. 1.2.3, we describe functionally the clustering to which we apply the unsupervised learning algorithms. In Sect. 1.2.4, we mention some hybrid tasks which are mixed with the three kinds of tasks. This section is intended to describe the three tasks and hybrid ones, functionally.

1.2.1 Classification

The classification is defined as the process of assigning one or some among the predefined categories to each item. The binary classification where each item is classified into one of two categories is mentioned as the simplest classification type. The binary classification is expanded into a multiple classification by predefined more categories. If the given task is the soft classification which more than one category is allowed to be assigned to each item, the multiple classification is decomposed into binary classifications. This section is intended to describe briefly the binary classification, the multiple classification, and its decomposition into binary classifications.

The binary classification is illustrated functionally in Fig. 1.1. The two categories, the positive class and the negative class, are predefined and the samples

Fig. 1.1 Binary classification

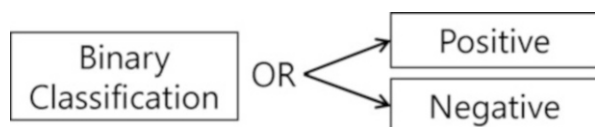


Fig. 1.2 Multiple classification

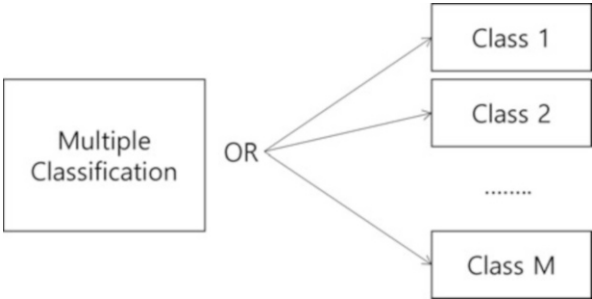
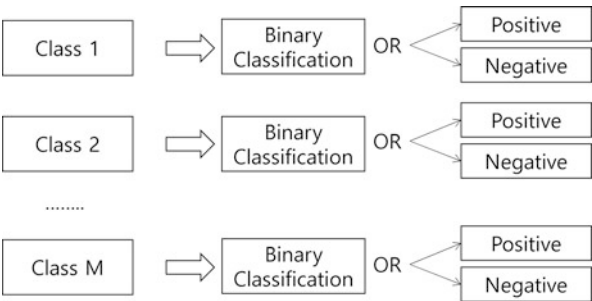


Fig. 1.3 Decomposition into binary classifications



are allocated to the both categories. The binary classification is to classify each item into the positive class or the negative class. Whenever studying initially the machine learning algorithms, it is assumed that the given problem is the binary classification. It is possible to decompose the multiple classification or the regression into binary classifications.

The multiple classification is illustrated in Fig. 1.2, functionally. More than two classes, M classes, $M > 2$, are predefined, and sample examples are allocated to each category. The multiple classification refers to the classification of each item into one or some among the M classes. If it is allowed to assign more than one category among the predefined ones, it belongs to the soft classification. It is possible to decompose the task into binary classification.

The process of decomposing the multiple classification into binary classifications is illustrated in Fig. 1.3. The multiple classification where M categories are predefined is initially given as the task which we must solve. The binary classifier which classifies each item into the corresponding category or not is assigned to each category. The multiple classification is decomposed into the binary classification tasks as many as categories; the task with the M categories is decomposed into M binary classification tasks as shown in Fig. 1.3. It is more suitable to decompose the multiple classification into binary classifications in the case of the soft classification.

Let us make some remarks on the classification tasks which the machine learning algorithms are applied popularly. The categories are fixed by the common sense, in the character recognition and spam mail filtering. It is complicated to predefined the categories in the topic based text categorization and the image classification.

The nested categories are allowed within a particular category in the hierarchical classification. A list or a tree of the predefined categories is called category system, and multiple category systems are allowed in the multiple viewed classification [1].

1.2.2 Regression

The regression is defined as the process of estimating an output value based on multiple factors. In the classification which was covered in Sect. 1.2.1, the output value is discrete, whereas in the regression which is covered in this section, the output value is continuous. There are two types of regression: the univariate regression where only one output value is estimated, and the multivariate regression where more than one output value is done. Typical regression instances are the nonlinear function approximation and the time series prediction. This section is intended to describe the regression, functionally.

Let us mention the univariate regression which is shown in Fig. 1.4 as the regression type where a single output variable is given. The nonlinear function approximation which estimates a single output variable with only single input variable is a typical example of the univariate regression. The prediction of a single future measure by analyzing previous measures and a current measure, called time series prediction, also belongs to the univariate regression. Other instances of the univariate regression are the prediction of a particular single stock price and that of a single exchange rate, such as EUR/USD. However, the prediction of increment or decrement of a value, rather than itself, belongs to the classification task.

The multivariate regression is illustrated functionally in Fig. 1.5. Multiple output values are generated to the input data which is given as an input vector, in this regression type. The process of predicting multiple future measures by analyzing previous measures and current ones is called multivariate time series prediction. The prediction of several stock prices and exchange rates of multiple currency pairs belong to the multivariate regression. It is possible to decompose the multivariate regression into independent univariate regressions.

The time series prediction which is the special type of regression is illustrated functionally in Fig. 1.6. It is the regression which is based on previous measures

Fig. 1.4 Univariate regression



Fig. 1.5 Multivariate regression



Fig. 1.6 Time series prediction



and a current one. The auto regression, the moving average, and the ARMA (Auto Regression and Moving Average) are traditional approaches to the time series prediction. The traditional models were replaced by the neural networks in 1990s [2] and by the SVM (Support Vector Machine) in 2000s [3]. Estimation of mid-terms was proposed for improving the times series prediction performance in using the neural networks [4, 5].

Let us make some remarks on the regression task. The regression is the task to which the supervised learning algorithms are applicable, together with the classification. In the regression, the output value is given as a continuous one, whereas in the classification, the output value is given as a discrete one. Because the inputs are given as in various scales, we need to normalize both input and output values between zero and one. The output value which is generated from the regression process should be de-normalized into its original scaled value.

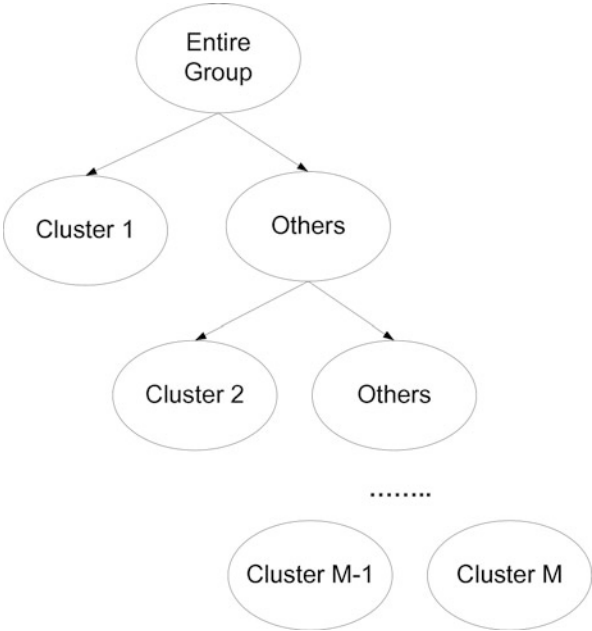
1.2.3 Clustering

The clustering is defined as the process of segmenting a group of items into subgroups each of which contains similar ones. The classification and the regression which are covered in the previous sections are the tasks to which the supervised learning algorithms are applied, but the clustering is the task which the unsupervised learning algorithms are applied. It is required to define a similarity metric between items for executing data clustering. The clustering is used for automating a collection of labeled examples, so the clustering may be integrated with the classification for organizing data items. This section is intended to describe the binary clustering and the multiple clustering in the functional view.

The binary clustering which partitions a group into two subgroups based on similarities among items is illustrated in Fig. 1.7. It is assumed that a single group of items is initially given as the input. A single item is selected from the group at random, and two subgroups are built: one is similar as the selected one, and the other is dissimilar. As an alternative way, two items may be selected at random as the initial cluster prototypes and the others are arranged into more similar group. The binary clustering is used in clustering data items by the divisive clustering algorithm which is covered in Sect. 9.3.

The multiple clustering which partitions a group into more than two subgroups at a time is illustrated in Fig. 1.8. A single group is initially given as the input like the binary clustering. The k means algorithm which is covered in Chap. 10 is a typical approach for implementing the multiple clustering. Items as many as clusters are

Fig. 1.9 Multiple clustering through binary clustering



classification with the text clustering [6]. Assigning a categorical score to each category as a continuous value is viewed as the combination of the classification and the regression. However, the system which provides the classification, the regression, and the clustering, as independent tasks, is never the system of hybrid tasks. This section is intended to describe the compound tasks by mixing the previous tasks.

The hybrid task which is the mixture of the classification and the regression is illustrated in Fig. 1.10. The task belongs to the only regression task in the function view; a vector which represents an item is given as the input and the continuous value is generated as the output. The item is classified into one of intervals and a value in the corresponding interval is estimated by the regression. In the task which is shown in Fig. 1.10, the classification becomes the supplementary task for reducing the estimation error, and the regression is the main task. The risk of this task is that the big error happens, if the item is misclassified.

The typical combination of the classification and the clustering is illustrated in Fig. 1.11. Unlabeled items are initially collected and are clustered based on their similarities. It results in a collection of items which are labeled with their own cluster identifiers, and they are used for training a classifier. Items which are given subsequently are classified by the classifier which learn clustered items. In this case, artificial training examples are generated by the clustering algorithm.

The results from the constraint clustering is illustrated in Fig. 1.12. The task refers to the process of clustering labeled examples as well as unlabeled ones. Both kinds of examples are initially given, initial clusters are constructed using the

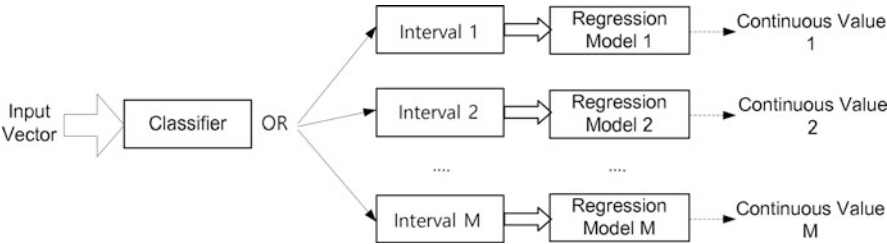


Fig. 1.10 Hybrid task: classification + regression

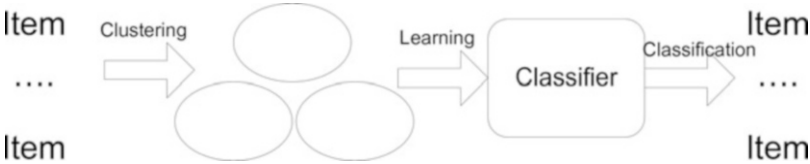


Fig. 1.11 Hybrid task: clustering + classification



Fig. 1.12 Constraint clustering

labeled examples, and unlabeled ones are arranged based on their similarities. In constructing the initial clusters, the labeled examples are arranged based on their labels. In the constraint clustering, we need to consider the possibility of other clusters beyond the target labels.

Let us mention the outlier detection as an instance of the hybrid task. It refers to the process of detecting whether individual items are exceptional, or not. When clustering data items, clusters of only one item or very small number of items belong to the outlier. The outliers are regarded as noises in processing data items, so it is necessary to remove them for cleaning data. The classification performance may be improved by the outlier detection.

1.3 Machine Learning Types

This section is concerned with the types of machine learning algorithms which are covered in this book. In Sect. 1.3.1, we describe the supervised learning which is covered in Part II. In Sect. 1.3.2, we explain the unsupervised learning which is covered in Part III. In Sects. 1.3.3 and 1.3.4, we mention the semi-supervised learning and the reinforcement learning, respectively. This section is intended to explore the four types of machine learning algorithms.

1.3.1 Supervised Learning

The definition of the supervised learning is illustrated as a block diagram in Fig. 1.13. We may consider the two kinds of output value in the supervised learning; the target output which is initially assigned to each training example and the computed output which is computed by the learning algorithm. The supervised learning is the learning paradigm where the parameters are optimized for minimizing the difference between the target output and the computed output. The KNN (K Nearest Neighbor), the Naive Bayes, and the decision tree belong to the supervised learning, and the classification and the regression are application areas of this type of learning algorithms. This section is intended to describe the supervised learning as the main learning paradigm which is covered in this book.

Let us mention the training examples which the supervised learning algorithms learn. Each training example is associated with its own category or a continuous value. The output which is initially labeled to each training example is called target output, and one which is computed by the learning algorithm is called computed output. The goal of the supervised learning is to minimize the misclassification or

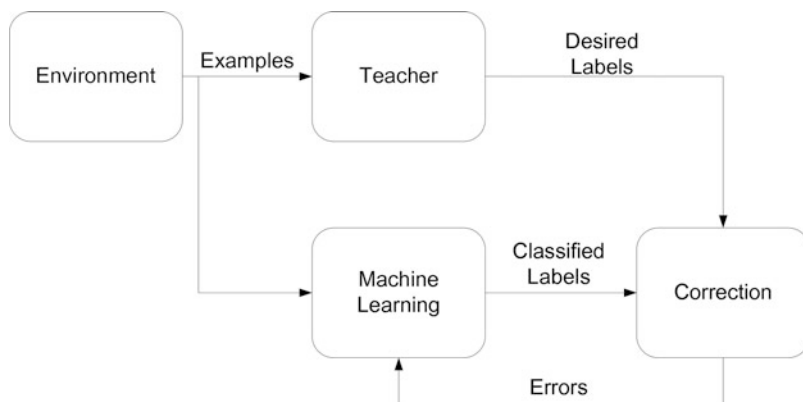


Fig. 1.13 Supervised learning

the error between the target output and the computed output. Unlabeled examples are added to the labeled which are prepared for the supervised learning, in the learning type, called semi-supervised learning.

Let us mention the process of learning the labeled training examples in the supervised learning algorithms. Their parameters are initialized at random and the labeled training examples are prepared. Their output values are computed and the parameters are updated to minimize the error between the target outputs and the computed outputs. The process is iterated to minimize the error or reach the convergence of the parameters. The learning process in the supervised learning is to optimize the parameters for minimizing the error.

Let us mention some areas to which the supervised machine learning algorithms are applied. The data classification which was described in Sect. 1.2.1 is the process of assigning one or some of the predefined categories to a novice item. The regression which is expanded from the classification and was described in Sect. 1.2.2 is the process of estimating a continuous output value or values to a novice input data. The time series prediction which is a special instance of the regression is the process of estimating a future value by analyzing the past values and a current value. The supervised learning algorithms may be applied to the outlier detection by mapping it into the classification task.

Let us make some remarks on the supervised learning which is covered in this section. It is required to associate each training example with its own label for the supervised learning. In the learning process, the parameters are optimized for minimizing the misclassification rate or the error. The classification and the regression are the tasks to which the supervised learning algorithms are applied. The case of utilizing unlabeled training examples, together with the labeled ones for the learning process is called semi-supervised learning.

1.3.2 Unsupervised Learning

The unsupervised learning as another learning paradigm is illustrated as a block diagram in Fig. 1.14. The training examples which are given for the learning paradigm are unlabeled and the clustering prototypes are initialized at random. The unsupervised learning is defined as the process of optimizing the cluster prototypes, depending on the similarities among the training examples. Clustering data items is the task to which the unsupervised learning algorithms are applied. This section is intended to explain the unsupervised learning as the opposite one to the supervised one which was covered in Sect. 1.3.1.

Let us mention the training examples which are prepared for the unsupervised learning. The fact that all of the training examples are labeled is the assumption in the supervised learning. In the unsupervised learning, all of the training examples are unlabeled, and the prototypes each of which characterizes its own cluster are initialized at random. The unlabeled training examples are used for optimizing

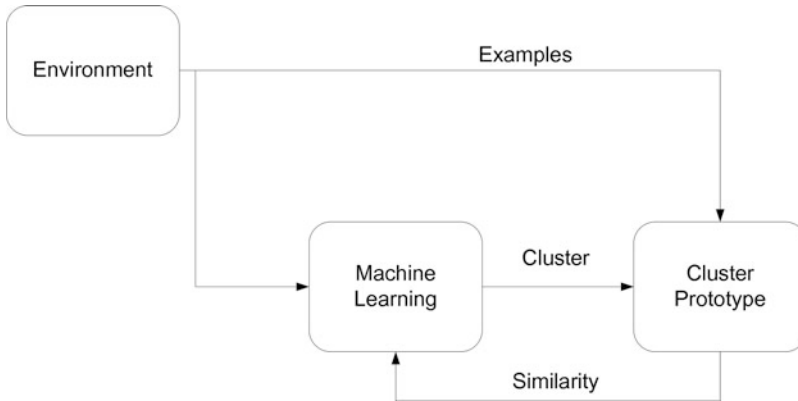


Fig. 1.14 Unsupervised learning

the cluster prototypes, depending on their similarities. The case of mixture of the unlabeled examples with some labeled ones for clustering data items is called constraint clustering.

Let us mention the learning process in the unsupervised learning algorithm as the frame rather than a specific algorithm. The number of clusters is decided and the cluster prototypes are initialized. Each data items in the group is arranged into the cluster whose prototype is most similar, and they are updated into more similar as their won arranged items. The arrangement and the update of the cluster prototypes are iterated until their convergences. The supervised learning is for minimizing the error or the misclassifications between the computed outputs and the target ones, whereas the unsupervised learning is for maximizing the similarities between the cluster prototypes and the data items.

Let us mention some cases of applying unsupervised learning algorithms to real tasks. The AHC algorithm which is covered in Sect. 9.2 was used for clustering texts by Jo in [1]. The Kohonen Networks were used for managing texts automatically in 2006, together with supervised learning algorithm [6]. The table based matching algorithm was used for clustering texts by Jo in [7]. The online linear clustering algorithm as fast clustering algorithm was used for detecting redundant national research projects in 2003 [8].

Let us make some remarks on the unsupervised learning. In the supervised learning, the training examples are labeled with their own output values, whereas in the unsupervised learning, they are unlabeled. The supervised learning depends on the target labels of the training examples, whereas the unsupervised learning depends on their similarities with the cluster prototypes. The supervised learning algorithms are applied to the classification and the regression, whereas the unsupervised learning algorithms are applied to the clustering. It is possible to modify the unsupervised learning algorithms into the supervised learnings; it will be explained in detail in Chap. 14.

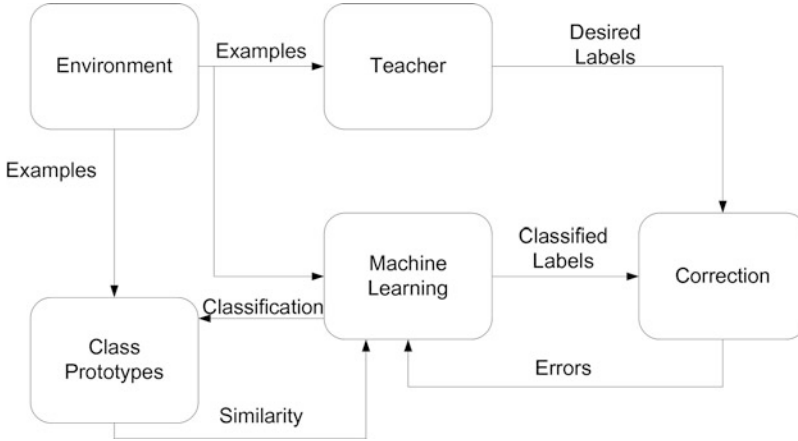


Fig. 1.15 Semi-supervised learning

1.3.3 *Semi-supervised Learning*

The semi-supervised learning is illustrated as a block diagram in Fig. 1.15. It is intended to utilize unlabeled examples which are very cheap to obtain as well as the labeled examples for training the learning algorithms. The machine learning learns the labeled examples by minimizing the error between their target label and the computed one and the unlabeled examples, depending on their similarities, as the mixture of the supervised learning and the unsupervised learning. By adding the unlabeled examples, the semi-supervised learning is applied to the classification tasks and the regression tasks. This section is intended to describe the semi-supervised learning as the mixture of both kinds of learning.

Let us mention the training examples which are prepared for the semi-supervised learning. The fact that the labeled examples are expensive to obtain, but the unlabeled ones are cheap to do is the motivation for proposing the semi-supervised learning. It is intended to add the unlabeled examples to the labeled ones for reinforcing the classification performance and the regression performance. In the semi-supervised learning, the training set is divided into the two sets: the set of labeled examples and the set of unlabeled ones. The originally unlabeled examples are artificially labeled by their similarities with the originally labeled ones or by the classifiers which learn them.

The semi-supervised learning is intended for the tasks to which the supervised learning algorithms are applied, such as the classification and the regression. The training example which are labeled originally are prepared and unlabeled ones are added. The parameters of the machine learning algorithms are updated by minimizing the error between the computed labels and the target labels to the labeled examples and by minimizing distance between the cluster prototypes and the input vectors to the unlabeled examples. The semi-supervised learning is implemented

by combining a supervised learning algorithm and an unsupervised one with each other; the unlabeled training examples are clustered by the unsupervised learning algorithm, based on the labeled ones, and both originally labeled examples and subsequently labeled ones are used for training the supervised learning algorithm. The semi-supervised learning is viewed as a compound or a combination of the supervised learning and the unsupervised one.

The semi-supervised learning is implemented by combining the unsupervised learning algorithm and the supervised one with each other. The labeled examples and the unlabeled examples are initially given as the training examples, and the unlabeled examples are clustered by the unsupervised learning algorithm, referring the labeled ones. The supervised learning algorithm is trained with both kinds of examples. A typical scheme of implementing the semi-supervised learning by the combination is to combine the Naive Bayes and the EM algorithm by Nigam et al. [9]. The scheme will be explained in detail in Chap. 14.

Let us mention the case of clustering data items using the both kinds of examples. In the unsupervised learning, the examples are assumed to be unlabeled. The data clustering where external parameters are optimized referring the labeled examples and the unlabeled examples are clustered, is called constraint clustering. Even if the constraint clustering is included as part in the process of the semi-supervised learning, it should be distinguished from the semi-supervised learning, with respect to their goals. The fact that the labeled examples are given as constraints in clustering data items is the reason of calling the task constraint clustering.

1.3.4 Reinforcement Learning

This section is concerned with the reinforcement learning in the functional view as the last type of machine learning, as illustrated in Fig. 1.16. The reinforcement

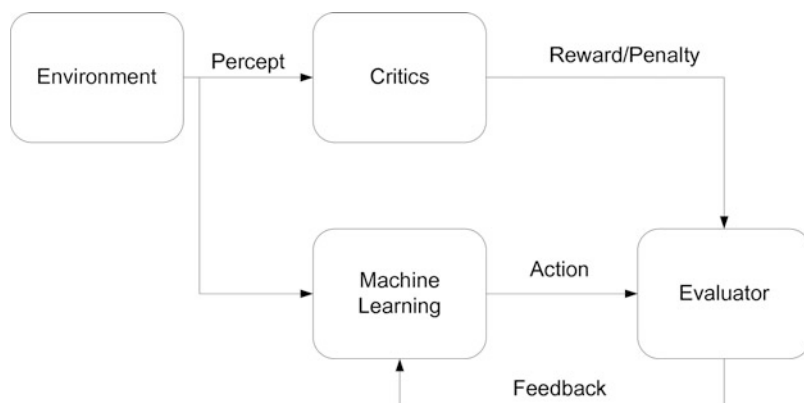


Fig. 1.16 Reinforced learning

learning is defined as the learning type where the parameters are updated for maximizing the reward and minimizing the penalty. In the reinforcement learning, the input is received from the external environment and the output is generated as the action. The reward or the penalty is given from the environment as the critics, and the parameters are updated for getting the reward and avoiding the penalty, as much as possible. This section is intended to describe the reinforcement learning as a learning frame, briefly.

We need to consider the external environment for studying the reinforcement learning. This learning type is viewed as the interaction between the environment and the learning agent. The reward or the penalty is decided by the environment after taking the action by the learning agent. The environment influences strongly on the decision of the learning agent which of the action is most desirable to the percept. It plays its role of the critic which judges the action from the learning agent.

Let us explain the reinforcement learning process, briefly. The learning agent accepts the percept from the external environment as the input and generates the action at random or from its table as the output. The reward or the penalty is given to the action. If the reward is given, the current output is set as the desired one, and otherwise, the output is updated. We will study this learning type, in detail in Chap. 16.

Let us mention some tasks to which the reinforcement learning is applied. The implementation of the autonomous moving agent is a typical area where the learning type is used. The leaning type is applied for implementing a game player which is an opposite to users. We may consider applying the reinforcement learning to the stochastic classification where the target labels of the training examples are not fixed. There is possibility of applying the reinforcement learning for implementing the autonomous trading agent in the financial area.

Let us make some remarks on the reinforcement learning. In the supervised learning and the unsupervised learning, it is assumed that the training examples are fixed, but in this learning type, the training examples are variable. As time goes, the reward and the penalty of even same input are variable. As the learning process, the table which is called Q table or policy table in Chap. 16 is updated depending on the reward or the penalty. The reinforcement learning will be studied in detail, in Chap. 16.

1.4 Related Areas

This section is concerned with the areas which are related with the machine learning. In Sect. 1.4.1, we explain the artificial intelligence from which the machine learning is derived as its branch. In Sect. 1.4.2, we mention the neural networks as the area which deals with the special type of machine learning algorithms. In Sect. 1.4.3, we mention the data mining which focuses on the tasks which should be solved by the machine learning algorithms. In Sect. 1.4.4, we consider the evolutionary computation which is used for the optimization tasks, as the additional related area.

1.4.1 Artificial Intelligence

The artificial intelligence is defined as the area for implementing a program or programs which execute like the behaviors of the human intelligence. The area was started in 1950s, intended for creating the artificial brain. It focuses on implementing the intelligent program, called autonomous agent, which does the tasks like human beings with the adaptability, rather than by the routine. The areas, the neural networks, the natural language processing, and the machine learning, are derived as its branches from it. In this section, we mention the main points of this area, the hill climbing, the knowledge representation, and the natural language processing.

The hill climbing is mentioned as a traditional optimization scheme in the area of the artificial intelligence. It is defined as the method of searching for the best solution by exploring the neighbors around the current position. It starts in a random point and moves into one of the neighbors with its better status than the current one. For example, in finding the root from a polynomial equation, if substituting a near value results in the output value which is closer to zero, it moves from the current value into the near value. The limit of this optimization method is to fall into a local minimum.

The knowledge representation which is a subfield of the artificial intelligence deals with the structured form of the knowledge. It is important material for reasoning, so it is necessary to represent the knowledge into a form which is understandable for the computer. The reasoning is the process of making a decision to the given task; the classification which is the decision of a category to a particular item is a typical example of reasoning. The ontology where the knowledge is viewed as a tree or a graph is the popular type of knowledge representation. The knowledge is given as a set of training examples, in context of the machine learning.

The natural language processing is the subarea of the artificial intelligence which deals with processing sentences which are written in a natural language, such as English, French, and Germany. The natural language means the language which is spoken and written commonly by human being, and the artificial language is one for expressing something more clearly, such as the programming language and the mathematical notation. The natural language understanding is the process of mapping sentences into structured knowledge representations, and the natural language generation is opposite to it. Both of them are used for implementing the machine translation between two different natural languages. The techniques of the natural language processing are utilized for reinforcing the information retrieval systems and the text mining systems.

Let us make some remarks on the artificial intelligence as the area which is related with this study. The artificial intelligence was established as an area in 1950s, and has existed as a main area of the computer science, until now. It is divided into the two types: the symbolic artificial intelligence which deals with the reasoning by the logic symbols and the connectionist artificial intelligence which deals with the computation models which simulate the neural system, called neural networks. The reasoning which is mentioned frequently in the artificial intelligence is the process

of making the conclusion based on the stored knowledge. The machine learning is the main part of the artificial intelligence.

1.4.2 Neural Networks

The neural networks are defined as area which deals with the special type of computation model which simulates the human nervous system. The area is intended to simulate the nervous system for understanding the natural neurons biologically and develop the solution to problems based on the nervous system. In the latter case, the neural networks are viewed as the special type of the machine learning algorithms based on the nervous system. The area begins with modeling the nervous system using a limited number of neurons by connecting them regularly. This section is intended to explore this area with respect to the relation with the machine learning.

In the neural networks, a single neuron is modeled as a computational unit. The artificial neuron is a basic unit which receives multiple values as its input and sends a single value as its output. Its net input is computed by summing the products of the values and their weights. The final output is computed by applying an activation function to the net input. The artificial neuron is viewed as a black box which computes an output value as a single value, by more than one input value.

Let us mention some neural networks for the supervised learning. The Perceptron was invented by Rosenblatt in 1950s as the initial neural networks for the supervised learning [10]. The MLP (Multiple Layer Perceptron) was involved by Rumelhart in 1980s; the invention opens the revival of the research on the neural networks [11]. The SVM (Support Vector Machine) was invented in 1990s and characterized to keep the linearity by mapping the training examples into ones in another space [12]. The NTC (Neural Text Categorizer) was invented by Jo in 2000s as one which is specialized for the text categorization [13].

The unsupervised neural networks belong to the alternative kind of the supervised ones which are mentioned above. The SOM (Self Organizing Map) which was intended by Kohonen in 1980s is a typical unsupervised neural network [14]. The SOM was modified into the supervised version which is called LVQ (Learning Vector Quantization) [15]. Based on the SOM, the neural gas was derived in 1990s [16]. The NTSO (Neural Text Self Organizer) was invented by Jo and Japkowicz in [17].

Let us make some remarks on the area of the neural networks which is covered in this section. The neural networks is viewed as the special type of the machine learning algorithm which simulates the biological nervous system. Although the neural networks may be included as a chapter in the text book on the machine learning, but it is not covered in this book, in order to avoid the redundant contents with the book on the neural networks; we author the book on the neural networks, separately from this study, so the neural networks will covered on the book. The neural networks are used for understanding the biological nervous system by simulating it, as well as the applications to the classification, the regression, and the

clustering. The neural networks belong to the soft computing which may generate different outputs to the same input by adding the randomness, together with the evolutionary computation and the fuzzy systems.

1.4.3 Data Mining

The data mining is defined as the area which deals with the process of extracting the implicit knowledge by analyzing data items. Its instances are the data classification, the data association, the regression, and the data clustering. The machine learning algorithms are mainly used as approaches to the data mining tasks. The data items are assumed as the records in the traditional data mining, but it is expanded into the various data mining types: the text mining, the web mining, and the bio mining. This section is intended to explain briefly the data mining as the area which is related with this study.

The data mining is the area which is expanded from the traditional computer science, database, by means of the data warehousing. The database is intended to access data in any application program by decoupling an application program and the data. The data items are accessed and manipulated by any application program, through the DBMS (Data Base Management System). The data warehouse is constructed from making decision by accessing the accumulated data. The data mining is expanded from the data warehousing for automating making the decision.

Let us mention the typical instances of the data mining. The data association which is motivated by the basket analysis means the process of extracting association rules from item sets. The data classification is the process of assigning automatically one or some among the predefined categories to each novice item. The regression is referred to the process of estimating continuous output values using the input factors. The data clustering is defined as the process of segmenting a group of data items into subgroups each of which consists of similar items.

Let us mention some data mining types, depending on the raw data types. The relational data mining as the traditional type is the process of extracting knowledge from the relational data. The text mining is the process of doing it from textual data. It may be expanded into the web mining which is the knowledge extraction from the web structures, web usages, and web documents. The bio mining is the knowledge extraction from the bio data such as DNA sequences.

Let us make some remarks on this area with respect to its relation with the machine learning. The data mining focuses on the tasks to which the machine learning algorithms are applied, such as the data classification, the data clustering, the regression, and the data association. The machine learning algorithms are used as the main approaches to the data mining tasks, even of other kinds of tools are available. The data mining and the machine learning tend to be coupled with each other, strongly. There are two routes for approaching to this area; one is from the data base by means of the data warehousing, and the other is from the artificial intelligence, focusing on the approaches.

1.4.4 Soft Computing

The soft computing is defined as the computing paradigm with the randomness. The hard computing which is opposite to the soft computing is referred to the computing with the same input and the same output. Because the randomness is involved in the soft computing, the output may be different even to the same input. The soft computing is applied to reinforce the performance of the machine learning algorithms. This section is intended to describe the soft computing with respect to its subareas, briefly.

Let us mention the three main areas of the soft computing. The neural networks which was mentioned in Sect. 1.4.2, is one of three main areas, in that their weights are initialized at random in any neural network model. The second main area is the evolutionary computation, in that the population of solution candidates is initialized at random, and random values are involved in the optimization process. The fuzzy computing and system belongs the soft computing as a main area, in that every value is mentioned as not a particular value but values around it including random values. In the text books on the soft computing, the three areas and the hybrid of them are covered.

Let us mention some hybrid computing which combines soft computing areas with each other. The evolutionary neural networks as the combination of the neural networks with the evolutionary computation are the neural networks where their weights are optimized by the evolutionary computation. Fuzzy distributions are defined by training neural networks using the training examples in the neuro-fuzzy system which combines the fuzzy system with the neural networks. The evolutionary fuzzy system which is the hybrid computing of the fuzzy system and the evolutionary computation is aimed to define fuzzy distributions by the evolutionary computation. The fuzzy neural networks are the neural networks where their weights are given as fuzzy values.

Let us consider the additional areas which are concerned with the soft computing. The swarm intelligence provides the optimization techniques which simulate the swarm behaviors. The artificial immune systems are also the area which solves the problem in the style of the key and the lock. The cultural algorithm is an additional area of the evolutionary computation which considers the belief space, in addition to the solution population. The rough set is the alternative area to the fuzzy set which considers the upper bound and the lower bound to each set.

Let us make some remarks on the soft computing with respect to its relations with the machine learning algorithms. The neural networks is the main area of the soft computing and was mentioned as an independent area in Sect. 1.4.2. The evolutionary computation may be used for tuning the parameters in executing the machine learning algorithms, and the parameter turning will be covered in Sect. 12.3. We may consider introducing the fuzzy concepts to the machine learning algorithms; the input values and the internal parameter values are treated as fuzzy values. It is possible to introduce the machine learning algorithms in the

fuzzy systems; the fuzzy distributions may be optimized by the machine learning algorithms.

1.5 Summary and Further Discussions

Let us summarize entirely the contents of this chapter. The classification, the regression, and the clustering are the tasks to which we apply the machine learning algorithms. There are four types of machine learning algorithms: the supervised learning, the unsupervised learning, the semi-supervised learning, and the reinforcement learning. We mentioned the typical four areas which are related with the machine learning: artificial intelligence, neural networks, data mining, and soft computing. This chapter covered the application areas, the machine learning types, and the related areas.

Let us mention the autonomous robot as the task to which we apply the machine learning algorithm. The reinforcement learning algorithm among the four types is used for implementing it. The front scene is given as the input and the direction is decided among forward, backward, left, and right. The reward or the penalty is given after making the decision and the reward scores are updated based on them. In Chap. 16, we will study it in detail.

Let us mention the associative memory as the map of a novice input into one of the training examples. This learning type is similar as One Nearest Neighbor which is covered in Chap. 5. A novice input is mapped into one of the labeled training examples and its label is decided by that of the mapped training example. The Hopfield Networks belong to this type of learning. This learning type is mainly used for restoring a damaged image into its original one.

Let us mention the evolutionary computation as an area which is related with the machine learning. It refers to the area which is concerned with the optimization techniques based on the evolutionary theory. The solution candidates are generated as the initial population at random, their offsprings are generated by applying the recombination and/or the mutation, and the next population is constructed by evaluating their fitness value and selecting some candidates. The population in subsequent generations consists of better solution candidates; the evolutionary computation is aimed to search for the solution by iterating the above process. In Chap. 12, we will mention the evolutionary algorithm which is used for clustering data items.

Let us mention the deep learning as the innovative machine learning type. It refers to the type of machine learning with multiple step from its input data to its output value, rather than with a single step. The deep learning is viewed into the trial of solving nonlinear problems with multiple steps of linear models, rather than with a single step of nonlinear model. The typical operations which are necessary for implementing the deep learning are pooling which generates more possible input data from a particular data item, and the convolution which averages over adjacent

input values. The convolutionary neural networks with its best performance in the image processing is the typical deep learning model.

References

1. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, Berlin, 2018)
2. T. Master, *Neural, Novel and Hybrid Algorithms for Time Series Prediction* (Wiley, New York, 1995)
3. K. Kim, Financial time series forecasting using support vector machines. *Neurocomputing* **55**(1–2), 307–319 (2003)
4. T. Jo, The effect of mid-term estimation on back propagation for time series prediction. *Neural Comput. Appl.* **19**(8), 1237–1250 (2010)
5. T. Jo, VTG schemes for using back propagation for multivariate time series prediction. *Appl. Soft Comput.* **13**(5), 2692–2702 (2013)
6. T. Jo, The implementation of dynamic document organization using text categorization and text clustering. PhD Dissertation of University of Ottawa, 2006
7. T. Jo, Table based single pass algorithm for clustering news articles. *Int. J. Fuzzy Log. Intell. Syst.* **8**(3), 231–237 (2008)
8. T. Jo, The application of text clustering techniques to detection of project redundancy in national R&D information system, in *The Proceedings of 2nd International Conference on Computer Science and Its Applications*, 2003
9. K. Nigam, A.K. McCallum, S. Thrun, T.M. Mitchell, Text classification from labeled and unlabeled documents using EM. *Mach. Learn.* **39**(2–3), 1–34 (2000)
10. F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**(6), 386–408 (1958)
11. D. Rumelhart, E. Geoffrey, E. Hinton, R.J. Williams, Learning internal representations by error propagation. No. ICS-8506, California Univ San Diego La Jolla Inst for Cognitive Science, 1985
12. C. Cortes, V. Vapnik, Support vector network. *Mach. Learn.* **20**(3), 237–297 (1995)
13. T. Jo, NeuroTextCategorizer: a new model of neural network for text categorization, in *The Proceedings of ICONIP*, 2000, pp. 280–285
14. T. Kohonen, Self-organized formation of topologically correct feature maps. *Biol. Cybern.* **43**(1), 59–69 (1982)
15. T. Kohonen, Learning vector quantization for pattern recognition. Rep. TKK-F-A601, Lab Computer and Inform Sci, 1986, 18 pp.
16. B. Fritzke, A growing neural gas network learns topologies, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, 1995), pp. 625–632
17. T. Jo, N. Japkowicz, Text clustering using NTSO, in *The Proceedings of IJCNN*, 2005, pp. 558–563

Chapter 2

Numerical Vectors



2.1 Introduction

The numerical vector is defined as a finite ordered set of numerical values. In each numerical vector, elements are given as ordered real values. The number of elements in each numerical vector is called its dimension. Every raw data should be converted into numerical vectors in using machine learning algorithms for real problems. This section is intended to understand the basic concepts about numerical vectors.

Let us consider the elements in a numerical vector. It is notated by $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$. The d values, $x_1, x_2, \dots, x_d$, become the elements of the vector, $\mathbf{x}$. The number of elements, d , is called the dimension of the vector, $\mathbf{x}$. Note that the elements, $x_1, x_2, \dots, x_d$, are ordered.

Let us mention the main operations on numerical vectors, and explain them in detail in Sect. 2.2. The addition and the deletion of two numerical vectors are simple operations on them. The scalar multiplication on a numerical vector is to multiply it by a constant value, and the norm of a numerical vector generates a scalar value which represents it. The inner product of two numerical vectors generates a scalar value which indicates the similarity between them, and the cosine similarity between them is a normalized similarity metric between zero and one. The outer product of two vectors generates a matrix as the opposite operation to the inner product.

A numerical vector is expanded into a matrix which consists of more than one vectors as rows or columns. The rows in the matrix are called row vectors, and the columns in it are called column vectors. A matrix is notated by Eq. (2.1),

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \mathbf{A} = [a_{ij}]_{m \times n} \quad (2.1)$$

The matrix where the number of rows is m , and the number of columns is n is called $m \times n$ matrix. When $m = n$, the matrix is called a square matrix.

The goal of this chapter is to understand the numerical vector and the matrix which are necessary for modeling the machine learning algorithms. We need to understand the numerical vector which is given as input data for the machine learning algorithms and the operations on them, mathematically. We need to understand the matrix which consists of numerical vectors as column vectors and row vector, and the operations on them. We will also understand the mathematical theories about the relations between numerical vectors and matrices. We will make further discussions about the mathematical foundations for understanding the machine learning algorithms.

2.2 Operations on Numerical Vectors

This section is concerned with the operations on numerical vectors. In Sect. 2.2.1, we define the numerical vectors, mathematically, and mention some basic operations on them in Sect. 2.2.2. In Sect. 2.2.3, we describe the inner product of numerical vectors which is most necessary for implementing the machine learning algorithms. In Sect. 2.2.4, we mention the linear combination of numerical vectors, and the linear independence as its property. This section is intended to cover the definition of numerical vectors, the operations on them, and the linear combination of them.

2.2.1 Definition

This section is concerned with the mathematical definition of numerical vectors. We already defined it in the previous section, conceptually, and it starts from a scalar value as a one dimensional vector. A numerical vector is a finite ordered set which contains more than one scalar value. It is possible to plot vectors visually until three dimensional space. This section is intended to define numerical vectors mathematically, before covering the operations on them.

Let us mention the scalar value, before discussing the numerical vector. The scalar value is a single numerical value and is notated by $x \in \mathfrak{R}$, an italic lower alphabetic character. A scalar value is viewed as a special type of numerical vector: a vector with only a single element. A scalar value is mentioned as a one dimensional vector. The norm of a vector is the unary operation on a numerical vector for expressing it as a scalar value.

The number of elements is finite in the numerical vector. It is called dimension and notated by d . When $d = 1$, it becomes a scalar value, and when $d = 2$ or $d = 3$, each numerical vector may be visualized in the 2 or 3 dimensional plane. When $d > 3$, it is impossible to visualize the numerical vector; it is viewed as an

ordered finite set of numerical values. If the dimension of the vector, $\mathbf{x}$, is d , the fact is notated by $x \in \mathbb{R}^d$.

Because the set and the vector look similar as each other in their appearance, we need to compare them with each other. In the vector, its elements are ordered, whereas in the set, its elements are unordered. In the vector, the number of elements is fixed, whereas in the set, the number of elements is variable. In the vector, its elements are always given as numerical vectors, whereas in the set, its elements are always given as various types of data. Operations on vectors are performed in the one to one style, whereas those on sets are performed in the style of all possible pairs.

Let us make some remarks on the numerical vectors which are given as structured data. It is assumed that the input is always given as a numerical vector in activating the machine learning algorithms for solving real problems. The dimension of the numerical vector is decided by number of attributes which characterize an item. The numerical vector may be expanded into a matrix or a tensor which are covered in Sect. 2.3.

2.2.2 Basic Operations

This section is concerned with the basic operations on numerical vectors. We need to define the operations, after doing the entities; for example, we defined the numerical vector in Sect. 2.2.1, and will do the operations on them in this section. The addition, the deletion, and the scalar multiplication are the binary operations on numerical vectors which generate a numerical vector as their output. The norm of numerical vector is a scalar value degree to the given numerical vector; it is a unary operation which generates a scalar value. This section is intended to describe the operations on numerical vectors, mathematically.

Let us mention the addition of two numerical vectors as a binary operation on them. Two vectors are notated as $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$ and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$. The addition of the two vectors is defined as Eq. (2.2),

$$\mathbf{x}_1 + \mathbf{x}_2 = [x_{11} + x_{21} \ x_{12} + x_{22} \ \dots \ x_{1d} + x_{2d}] \quad (2.2)$$

The sum of n vectors, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, is defined as Eq. (2.3),

$$\sum_{i=1}^n \mathbf{x}_i = \left[\sum_{i=1}^n x_{i1} \ \sum_{i=1}^n x_{i2} \ \dots \ \sum_{i=1}^n x_{id} \right] \quad (2.3)$$

The zero vector whose elements are zeros, $\mathbf{0} = [0 \ 0 \ \dots \ 0]$, is mentioned as the identity to this operation, and added to vector, $\mathbf{x}_1$ as Eq. (2.4),

$$\mathbf{x}_1 + \mathbf{0} = \mathbf{x}_1 \quad (2.4)$$

Let us mention another operation on two numerical vectors, deletion. It is assumed that two vectors are $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$ and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$. The deletion of the two vectors is defined as Eq. (2.5),

$$\mathbf{x}_1 - \mathbf{x}_2 = [x_{11} - x_{21} \ x_{12} - x_{22} \ \dots \ x_{1d} - x_{2d}] \quad (2.5)$$

The deletion is same to the addition of the negative vector as shown in Eq. (2.6),

$$\mathbf{x}_1 + (-\mathbf{x}_2) = \mathbf{x}_1 - \mathbf{x}_2 = [x_{11} - x_{21} \ x_{12} - x_{22} \ \dots \ x_{1d} - x_{2d}] \quad (2.6)$$

If a particular vector is assumed to be $\mathbf{x}_1$, $-\mathbf{x}_1$ is its reverse in the addition.

The scalar multiplication is mentioned as one more operation on a numerical vector. The scalar value and the vector are notated, respectively, by c and $\mathbf{x}_1$. The scalar multiplication on a vector is expressed by Eq. (2.7),

$$c \cdot \mathbf{x}_1 = [cx_{11} \ cx_{12} \ \dots \ cx_{1d}] \quad (2.7)$$

Applying the operation results in elements which are multiplied by the scalar value, c . The operands of this operation are given as a scalar and a vector, and the output is given as a vector.

Let us mention the unary operation on a numerical vector which is necessary for studying machine learning algorithms, called norm. It is used for representing a numerical vector as a scalar value which indicates the vector size. The norm of the vector, $\mathbf{x}_1$, is expressed into Eq. (2.8), as the general form,

$$\|\mathbf{x}_1\| = \left(\sum_{i=1}^d |x_{1i}|^r \right)^{\frac{1}{r}} \quad (2.8)$$

The dimension of the vector becomes a norm in $r = 0$, and in $r = \infty$, the maximum element in the vector becomes its norm. In studying the machine learning algorithms, r is usually set to one or two.

2.2.3 Inner Product

This section is concerned with the important operation on numerical vectors, called inner product. The operands to this operation are two same dimensional numerical vectors, and its output is a scalar value. The inner product is the summation of product of one to one element of two vectors. When two vectors are directed identically, the inner product is maximized, but when they are directed perpendicular to each other, the inner product becomes zero. This section is intended to treat the operation on two vectors, inner product.

Let us mention a normal vector before explaining the operation on numerical vector, called inner product. A normal vector is one whose norm is one. If a vector, $\mathbf{x}_1$, is a normal vector, its norm is one as shown in Eq. (2.9),

$$\|\mathbf{x}_1\| = 1 \quad (2.9)$$

A vector may be converted into its normal form by dividing its element by its norm. The multiplication of norm of a vector, $\mathbf{x}_2$, by the normal vector, $\mathbf{x}_1$ is same to norm of vector, $\mathbf{x}_2$, as shown in Eq. (2.10),

$$\|\mathbf{x}_1\| \cdot \|\mathbf{x}_2\| = \|\mathbf{x}_2\| \quad (2.10)$$

Let us mention the process of computing the inner product of two vectors. They are initially given as $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$ and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$. The inner product of two vectors is expressed as Eq. (2.11),

$$\mathbf{x}_1 \cdot \mathbf{x}_2 = \sum_{i=1}^d x_{1i} x_{2i} \quad (2.11)$$

A scalar value is generated as the output in any dimensional vectors. The relation between the inner product and the product of norms of two vectors is expressed as Eq. (2.12),

$$\|\mathbf{x}_1\| \cdot \|\mathbf{x}_2\| \geq |\mathbf{x}_1 \cdot \mathbf{x}_2| \quad (2.12)$$

Let us mention the orthogonality as the special results from the inner product of two vectors. In the geometrical view when two vectors are in their perpendicular direction, the inner product of them is given as zero value. Two vectors are notated by $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$ and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$, and if $\mathbf{x}_1 \cdot \mathbf{x}_2 = 0$ two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, are orthogonal. When the norms of two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are one as $\|\mathbf{x}_1\| = \|\mathbf{x}_2\| = 1$, and they are orthogonal each other, this case is called orthonormal. When two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, are with same direction, $\|\mathbf{x}_1\| \cdot \|\mathbf{x}_2\| = \|\mathbf{x}_1 \cdot \mathbf{x}_2\|$.

Let us mention the operation which is opposite to the inner product, called outer product. Two vectors are given as $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$ and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$. A matrix is generated from applying the outer product to the two vectors, as shown in Eq. (2.13),

$$\mathbf{x}_1 \otimes \mathbf{x}_2 = \begin{bmatrix} x_{11}x_{21} & x_{11}x_{22} & \dots & x_{11}x_{2d} \\ x_{12}x_{21} & x_{12}x_{22} & \dots & x_{12}x_{2d} \\ \dots & \dots & \dots & \dots \\ x_{1d}x_{21} & x_{1d}x_{22} & \dots & x_{1d}x_{2d} \end{bmatrix} \quad (2.13)$$

In the inner product, a scalar value is generated as the output, whereas a $d \times d$ matrix is generated in the outer product. In implementing the machine learning algorithms, the inner product is used more frequently than the outer product.

2.2.4 Linear Independence

This section is concerned with the linearly combination of vectors. A vector may be expressed by a linear combination of more than one product of a vector and a coefficient; the vectors which are involved in the linear combination are called spans. Only when all coefficients are zero, the linear combination of vectors becomes zero vector; this case is called linearly independent of spans. If the spans are linearly dependent on each other, a particular vector in the spans is expressed by a linear combination of the others. This section is intended to characterize mathematically the linear combination of vectors.

A particular d dimensional vector, $\mathbf{x} \in \mathcal{R}^d$, is expressed as a linear combination of vectors, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m, \mathbf{v}_i \in \mathcal{R}^d$. The set of vectors, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$, that is used for expressing the vector, $\mathbf{x}$, is called the spanning set of the vector, $\mathbf{x}$. The spanning set is notated by $S = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m\}$, and the linear combination of the vectors in the spanning set is expressed for the vector $\mathbf{x}$ into Eq. (2.14),

$$\mathbf{x} = \sum_{i=1}^m c_i \mathbf{v}_i \quad (2.14)$$

where c_i is a constant coefficient. For example, the three dimensional vector is $\mathbf{x} = [x_1 \ x_2 \ x_3]$ is, and the set, $S = \{[1 \ 0 \ 0], [0 \ 1 \ 0], [0 \ 0 \ 1]\}$, is expressed for the vector, $\mathbf{x}$, in Eq. (2.15),

$$\mathbf{x} = x_1[1 \ 0 \ 0] + x_2[0 \ 1 \ 0] + x_3[0 \ 0 \ 1] \quad (2.15)$$

The linear combination of the vectors in the spanning set is used for modeling a linear classifier.

The linear independence is considered from the linear combination of spans, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$, for expressing a particular vector, $\mathbf{x}$. The linear independence is viewed as no influence among the spans, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$; the vectors in the spanning set are independent of each other. It is defined mathematically as Eq. (2.16),

$$\sum_{i=1}^m c_i \mathbf{v}_i = 0; \text{ only if } c_1 = c_2 = \dots = c_m = 0 \quad (2.16)$$

The spanning set, $S = \{[1 \ 0 \ 0], [0 \ 1 \ 0], [0 \ 0 \ 1]\}$, is the linearly independent spans for expressing a three dimensional vector. If the vectors in the spanning set are linearly independent of each other, a particular vector in the spanning set is never expressed with a linear combination of the others.

Let us mention the linear dependency among the vectors in the spanning set, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$. In the spanning set, a vector, $\mathbf{v}_i$, exists, satisfying $\sum_{i=1}^m c_i \mathbf{v}_i = 0$, in non-zero coefficients. One among $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$, $\mathbf{v}_i$ is expressed by a linear combination of the others in the spanning set, as shown in Eq. (2.17),

$$\mathbf{v}_i = \sum_{j=1}^m c_j \mathbf{v}_j \quad (2.17)$$

For example, the spanning set, $S = \{[1 \ 0 \ 0], [0 \ 1 \ 0], [2 \ 0 \ 0]\}$, is a typical example of the linear dependency, since $\mathbf{v}_3 = 2 \cdot \mathbf{v}_1 + 0 \cdot \mathbf{v}_2$. The spanning set size should be reduced for more efficiency in this case.

Let us make some remarks on the linear combination of vectors. A particular vector is expressed by a linear combination of other vectors, and the set of the vectors is called spanning set. The d dimensional vector is expressed into a linear combination of vectors in the spanning set as Eq. (2.18),

$$\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_d \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \\ \dots \\ 0 \end{bmatrix} + \dots + x_d \begin{bmatrix} 0 \\ 0 \\ \dots \\ 1 \end{bmatrix} \quad (2.18)$$

The fact that the vectors in the spanning set are independent of each other for expressing a particular vector is called linear independence. The case where at least a vector in the spanning set is expressed with a linear combination of the others is called linear dependence.

2.3 Operations on Matrices

This section is concerned with the operations on matrices which are expanded from the matrices. In Sect. 2.3.1, we define the matrix mathematically. In Sect. 2.3.2, we mention the basic operations on matrices, such as the addition, the deletion, and the scalar multiplication. In Sects. 2.3.3 and 2.3.4, we describe the multiplication on two matrices and the inverse matrix, respectively. This section is intended to study the matrix as the fundamental for studying the machine learning algorithms.

2.3.1 Definition

A matrix is defined as an entry which consists of numerical values as rows and columns. It is viewed as a set of vectors which were covered in Sect. 2.2. A matrix is expressed as follows:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

where x_{ij} is given as its element. The above matrix is called $m \times n$ matrix where m is the number of rows and n is the number of columns, as its size. This section is intended to describe the matrix for understanding it conceptually.

Let us review the scalar and the vector which were covered in Sect. 2.2, before studying the matrix. The case where a single numerical value is given as an entry is called scalar; a scalar value corresponds to a 1×1 matrix. The case where more than one numerical value is given as a single row or a single column is called vector; a d dimensional vector corresponds to a $1 \times d$ or $d \times 1$ matrix. The scalar is expanded to the matrix by means of the vector; numerical values are given in a matrix with two axes as rows and columns. The case of numerical values with more than two axes is called tensor.

Let us describe the matrix with respect to its structure, mathematically. A matrix consists of more than one vectors, as rows and columns. The matrix, $\mathbf{X}$, is notated as follows:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

where x_{ij} is the element in row i and column j . The number of rows is m and the number of columns is n as the matrix size; the above matrix is called $m \times n$ matrix. Therefore, the above matrix has m row vectors and n column vectors.

Let us mention the special types of matrix. When the number of columns is same to the number of rows, the matrix is called square matrix. When $x_{ij} = x_{ji}$, the matrix is called symmetry matrix. When the diagonal elements, x_{ii} , are non-zeros and off diagonal elements, x_{ij} , $i \neq j$, the matrix is called diagonal matrix. When diagonal elements are ones among diagonal matrices, the matrix is called identity matrix.

Let us make some remarks on the matrix which is described conceptually in this section. The scalar corresponds to a 1×1 matrix, and the vector does to a $1 \times d$ or $d \times 1$ matrix. The matrix size is expressed as the product of the number of columns and the number of rows. The element of the matrix is x_{ij} where i is a row index

and j is a column index. In the matrix, its row indicates a row vector, and its column indicates a column vector.

2.3.2 Basic Operations

This section is concerned with the basic operations on the matrices. In Sect. 2.3.1, we defined and characterized mathematically the matrix. We will mention the addition, the deletion, the scalar multiplication, and the vector multiplication, as the basic operations on matrices. The matrix size should be identical for applying the addition and the deletion. This section is intended to review the basic operations on matrices, briefly.

Let us mention the addition of the two matrices as a basic operation. Let us notate two matrices as follows:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

The addition of the two matrices, $\mathbf{A}$ and $\mathbf{B}$, is defined as Eq. (2.19),

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2n} + b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{bmatrix} \quad (2.19)$$

The addition of the two matrices is defined as the addition of their elements as shown in Eq. (2.19). The same size of the two matrices is required for applying the addition; the number of rows and the number of columns in the two matrices should be identical to each other.

Let us mention the deletion of matrix from another. It is assumed that two matrices are given like above. The deletion of matrix, $\mathbf{B}$, from the matrix, $\mathbf{A}$, is expressed by Eq. (2.20),

$$\mathbf{A} - \mathbf{B} = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} & \dots & a_{1n} - b_{1n} \\ a_{21} - b_{21} & a_{22} - b_{22} & \dots & a_{2n} - b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} - b_{m1} & a_{m2} - b_{m2} & \dots & a_{mn} - b_{mn} \end{bmatrix} \quad (2.20)$$

The deletion is viewed as the addition of negative second term, as shown in Eq. (2.21),

$$\mathbf{A} - \mathbf{B} = \mathbf{A} + (-\mathbf{B}) \quad (2.21)$$

The same size of the two matrices is the requirement like the addition of them.

Let us mention one more basic operation on a matrix, scalar multiplication. A scalar value, c , and a matrix, $\mathbf{A}$, are given as the operands for this operation. The scalar multiplication by c to the matrix, $\mathbf{A}$, is expressed by Eq. (2.22),

$$c\mathbf{A} = \begin{bmatrix} ca_{11} & ca_{12} & \dots & ca_{1n} \\ ca_{21} & ca_{22} & \dots & ca_{2n} \\ \dots & \dots & \dots & \dots \\ ca_{m1} & ca_{m2} & \dots & ca_{mn} \end{bmatrix} \quad (2.22)$$

The multiplication of the matrix, $\mathbf{A}$, by a scalar value c , is shown in Eq. (2.22). The distributive law is applicable to the addition, the deletion, and the scalar multiplication, as shown in Eq. (2.23),

$$c(\mathbf{A} \pm \mathbf{B}) = c\mathbf{A} \pm c\mathbf{B} \quad (2.23)$$

Let us mention the multiplication of a matrix with a vector. The number of columns in the matrix and the dimension of the vector should be consistent with each other for performing the operation. The multiplication of the matrix with the vector is expressed as Eq. (2.24),

$$\mathbf{Ax} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n a_{1i}x_i \\ \sum_{i=1}^n a_{2i}x_i \\ \dots \\ \sum_{i=1}^n a_{mi}x_i \end{bmatrix} \quad (2.24)$$

The multiplication of the $m \times n$ matrix with the n dimensional vector generates the n dimensional vector, as shown in Eq. (2.24). The multiplication of two matrices will be covered in Sect. 2.3.3.

2.3.3 Multiplication

This section is concerned with the multiplication of two matrices. In Sect. 2.3.2, we studied the multiplication of a matrix by a scalar value and a vector. In this section, two matrices are multiplied by each other, by applying the inner product to each row vector of the left matrix and each column vector of the right matrix. The commutative law is not applicable to the multiplication of two matrices. This section is intended to describe the multiplication of two matrices, mathematically.

Let us mention the multiplication of two trivial matrices, in order to warm up one of two general matrices. Two 2×2 are notated as follows:

$$\mathbf{A}_{2 \times 2} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \mathbf{B}_{2 \times 2} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

The multiplication of the two 2×2 is expressed as Eq. (2.25),

$$\mathbf{A}_{2 \times 2} \mathbf{B}_{2 \times 2} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} \quad (2.25)$$

The two 3×3 are given as follows:

$$\mathbf{A}_{3 \times 3} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \mathbf{B}_{3 \times 3} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

The multiplication of the two matrices is expressed as Eq. (2.26),

$$\begin{aligned} & \mathbf{A}_{3 \times 3} \mathbf{B}_{3 \times 3} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{bmatrix} \end{aligned} \quad (2.26)$$

The element of resulted matrix is viewed as the inner product of a row vector of the left matrix and a column vector of the right matrix.

Let us consider the multiplication of two $n \times n$ square matrices as the general form. The two $n \times n$ matrices are notated as follows:

$$\mathbf{A}_{n \times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, \mathbf{B}_{n \times n} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}$$

The multiplication of the two matrices is expressed by Eq. (2.27),

$$\mathbf{A}_{n \times n} \mathbf{B}_{n \times n} = \begin{bmatrix} \sum_{k=1}^n a_{1k}b_{k1} & \sum_{k=1}^n a_{1k}b_{k2} & \dots & \sum_{k=1}^n a_{1k}b_{kn} \\ \sum_{k=1}^n a_{2k}b_{k1} & \sum_{k=1}^n a_{2k}b_{k2} & \dots & \sum_{k=1}^n a_{2k}b_{kn} \\ \dots & \dots & \dots & \dots \\ \sum_{k=1}^n a_{nk}b_{k1} & \sum_{k=1}^n a_{nk}b_{k2} & \dots & \sum_{k=1}^n a_{nk}b_{kn} \end{bmatrix} \quad (2.27)$$

Each element in the multiplication of the two square matrices is inner product of i th n dimensional row vector and j th n dimensional column vector, as expressed in Eq. (2.28),

$$\mathbf{r}_i \cdot \mathbf{c}_j = \sum_{k=1}^n a_{ik} b_{kj} \quad (2.28)$$

where $\mathbf{r}_i$ is the i th row vector in the matrix $\mathbf{A}$, and $\mathbf{c}_j$ is the j th column vector, in the matrix, $\mathbf{B}$. The dimension of the row vector in the left matrix should be identical to the dimension of the column vector in the right matrix, in order to apply the multiplication of two matrices.

Let us mention the multiplication of two non-square matrices as the more general case. The requirement for performing the multiplication of two matrices is that the number of columns in the left matrix should be identical to the number of rows in the right matrix. The two matrices are given as follows:

$$\mathbf{A}_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \mathbf{B}_{n \times l} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1l} \\ b_{21} & b_{22} & \dots & b_{2l} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nl} \end{bmatrix},$$

and the multiplication of them is expressed as Eq. (2.29),

$$\mathbf{A}_{m \times n} \mathbf{B}_{n \times l} = \begin{bmatrix} \sum_{k=1}^n a_{1k} b_{k1} & \sum_{k=1}^n a_{1k} b_{k2} & \dots & \sum_{k=1}^n a_{1k} b_{kl} \\ \sum_{k=1}^n a_{2k} b_{k1} & \sum_{k=1}^n a_{2k} b_{k2} & \dots & \sum_{k=1}^n a_{2k} b_{kl} \\ \dots & \dots & \dots & \dots \\ \sum_{k=1}^n a_{mk} b_{k1} & \sum_{k=1}^n a_{mk} b_{k2} & \dots & \sum_{k=1}^n a_{mk} b_{kl} \end{bmatrix} \quad (2.29)$$

The $m \times n$ matrix in the left and the $n \times l$ matrix in the right are multiplied with each other into a $m \times l$, and the commutative law is not applicable to the case. Swapping the two matrices in Eq. (2.29) is not applicable; it does not satisfy the requirement which was mentioned above.

Let us make some remarks on the multiplication of two matrices which is covered in this section. The commutative law is not applicable to the operation on matrices, as expressed in Eq. (2.30),

$$\mathbf{AB} \neq \mathbf{BA} \quad (2.30)$$

The matrix which is result from multiplying two matrices consists of elements each of which is an inner product of the row vector of the left matrix and the column vector of the right matrix. The same dimension of the row vectors in the left matrix to that of the column vectors in the right matrix is required for applying the multiplication to them. In next section, we will study the identity matrix, $\mathbf{I}$, where $\mathbf{AI} = \mathbf{A}$ and the inverse matrix of the matrix, $\mathbf{A}^{-1}$, where $\mathbf{AA}^{-1} = \mathbf{I}$.

2.3.4 Inverse Matrix

This section is concerned with the special kinds of matrix: the identity matrix and the inverse matrix. In the previous section, we studied the multiplication of the two matrices as the main operation. The identity matrix is the matrix which is multiplied to a particular matrix for generating the same matrix, and the inverse matrix is the matrix which is multiplied to a matrix for generating the identity matrix. The inverse matrix is used for solving a linear equation system. This section is intended to describe the identity matrix, the inverse matrix, and its application to solving a linear equation system.

The identity matrix is defined as a special type of square matrix which cases a square matrix to be itself by its multiplication. The identity matrix is notated by $\mathbf{I}$ and is multiplied with a matrix, $\mathbf{AI} = \mathbf{A}$. In the identity matrix, its off-diagonal elements are given as zeros, and its diagonal elements are given as ones, as shown in Eq. (2.31),

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix} \quad (2.31)$$

The elements of the identity matrix are notated as Eq. (2.32),

$$\mathbf{I}_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (2.32)$$

The identity matrix is characterized as diagonal, square, and symmetry.

The inverse matrix of a particular matrix is one which generates the identity matrix by multiplying them with each other. The inverse matrix of a matrix $\mathbf{A}$ is notated by $\mathbf{A}^{-1}$, assuming that both matrices are square ones. If the two matrices, $\mathbf{A}$ and $\mathbf{A}^{-1}$, are multiplied with each other, the identity matrix is generated as shown in Eq. (2.33),

$$\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \quad (2.33)$$

In case of the non-square matrix, $\mathbf{A}_{m \times n}$, the identity matrix, $\mathbf{I}_{n \times n}$, is generated as shown in Eq. (2.34),

$$\mathbf{A}_{n \times m}^+ \mathbf{A}_{m \times n} = \mathbf{I} \quad (2.34)$$

the matrix, in Eq. (2.34), is called the pseudo inverse matrix of the matrix, $\mathbf{A}_{m \times n}$. Not all square matrix have its inverse matrix; if its inverse matrix is available, the matrix is invertible.

Let us mention the Gaussian-Jordan elimination as the process of deriving the inverse matrix from a particular matrix. The augmented matrix which is a pair of a matrix, $\mathbf{A}$, and the identity matrix is constructed as shown in Eq. (2.35),

$$[\mathbf{A}|\mathbf{I}] = \left[\begin{array}{cccc|cccc} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{array} \right] \quad (2.35)$$

The matrix, $\mathbf{A}$, is assumed to be a $n \times n$ matrix, the n row vectors of the matrix, $\mathbf{A}$, $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n$, and the n row vectors of the identity matrix, $\mathbf{r}'_1, \mathbf{r}'_2, \dots, \mathbf{r}'_n$, and the above augmented matrix is changed into the augmented matrix which is presented in Eq. (2.36), by updating each row of both matrices, $\mathbf{A}$ and $\mathbf{I}$, by Eq. (2.36),

$$\begin{aligned} \mathbf{r}_k &\leftarrow \mathbf{r}_k - \frac{a_{k1}}{a_{11}} \mathbf{r}_1 \\ \mathbf{r}_k &\leftarrow \mathbf{r}_k - \frac{a_{k2}}{a_{22}} \mathbf{r}_2 \\ &\dots \\ \mathbf{r}_k &\leftarrow \mathbf{r}_k - \frac{a_{kk-1}}{a_{k-1k-1}} \mathbf{r}_{k-1} \end{aligned} \quad (2.36)$$

$$[\mathbf{B}|\mathbf{C}] = \left[\begin{array}{cccc|cccc} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ 0 & b_{22} & \dots & b_{2n} & c_{11} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & a_{11}^{-1} & a_{12}^{-1} & \dots & 1 \end{array} \right] \quad (2.37)$$

where a_{ij}^{-1} is the element of the inverse matrix, $\mathbf{A}^{-1}$. The augmented matrix which is presented in Eq. (2.37) is changed into one which is presented in Eq. (2.39), by updating each row of both, $\mathbf{B}$ and $\mathbf{C}$, using Eq. (2.38),

$$\begin{aligned} \mathbf{r}_k &\leftarrow \mathbf{r}_k - \frac{a_{kn}}{a_{nn}} \mathbf{r}_n \\ \mathbf{r}_k &\leftarrow \mathbf{r}_k - \frac{a_{kn-1}}{a_{n-1n-1}} \mathbf{r}_{n-1} \\ &\dots \\ \mathbf{r}_k &\leftarrow \mathbf{r}_k - \frac{a_{kk-1}}{a_{k-1k-1}} \mathbf{r}_{k-1} \end{aligned} \quad (2.38)$$

$$[\mathbf{I}|\mathbf{A}^{-1}] = \left[\begin{array}{ccc|ccc} 1 & 0 & \dots & 0 & a_{11}^{-1} & a_{12}^{-1} & \dots & a_{1n}^{-1} \\ 0 & 1 & \dots & 0 & a_{21}^{-1} & a_{22}^{-1} & \dots & a_{2n}^{-1} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & a_{n1}^{-1} & a_{n2}^{-1} & \dots & a_{nn}^{-1} \end{array} \right] \quad (2.39)$$

From the Gaussian-Jordan elimination process, we know that if one zero row vector is available at least in the given matrix, the matrix is not invertible.

The inverse matrix is used for solving a linear equation system. The linear equation system is expressed as Eq. (2.40),

$$\begin{aligned} a_1 1x_1 + a_1 2x_1 + \dots + a_1 nx_n &= c_1 \\ a_2 1x_1 + a_2 2x_1 + \dots + a_2 nx_n &= c_2 \\ \dots & \\ a_n 1x_1 + a_n 2x_1 + \dots + a_n nx_n &= c_n, \end{aligned} \quad (2.40)$$

and find the solution as the values of $x_1, x_2, \dots, x_n$, satisfying Eq. (2.40). The linear equation is expressed as the product of a matrix and a vector, as shown in Eq. (2.41),

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{bmatrix} \quad (2.41)$$

Equation (2.41) is expressed into its simplified version as shown in Eq. (2.42),

$$\mathbf{Ax} = \mathbf{c} \quad (2.42)$$

and find the solution by getting the inverse of the matrix, $\mathbf{A}$, as shown in Eq. (2.43),

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{c} \quad (2.43)$$

The inverse matrix is constructed by the Gaussian-Jordan elimination as mentioned above.

2.4 Vector and Matrix

This section is concerned with the mathematical theory about the vector and the matrix. In Sect. 2.4.1, we explain the determinant which is a scalar value representing a matrix. In Sect. 2.4.2, we describe the Eigen values and the Eigen vectors for representing a matrix with a vector. In Sect. 2.4.3, we mention the SVD (Singular Value Decomposition) as a tool for reduce the dimension. In Sect. 2.4.4,

we deal with the principal component analysis as one more dimension reduction scheme.

2.4.1 Determinant

The determinant of a particular matrix is defined as a scalar value which represents itself. The invertibility of a matrix is determined by its determinant value; if its determinant is non-zero, it is invertible. The determinant of a matrix, $\mathbf{A}$, is expressed as Eq. (2.44),

$$|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{vmatrix} \quad (2.44)$$

The determinant is applicable only to a square matrix. This section is intended to describe the process of computing the determinant of a matrix.

Let us mention of the process of finding the determinant from 2×2 matrix as the simple case. The 2×2 is expressed by Eq. (2.45),

$$|\mathbf{A}_{2 \times 2}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \quad (2.45)$$

The determinant is expressed as Eq. (2.46),

$$|\mathbf{A}_{2 \times 2}| = a_{11}a_{22} - a_{12}a_{21} \quad (2.46)$$

The inverse matrix of $\mathbf{A}$ is given as Eq. (2.47),

$$|\mathbf{A}_{2 \times 2}^{-1}| = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \quad (2.47)$$

If the determinant of the matrix is zero, it is not invertible.

Let us consider the determinant of $n \times n$ matrix as a general square matrix. $\mathbf{A}$ $n \times n$ matrix is expressed by Eq. (2.48),

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad (2.48)$$

and we define the submatrix, $\mathbf{A}_{ij}$, where i th row and j th column are deleted. The determinant of the matrix, $\mathbf{A}$, is defined as Eq. (2.49),

$$|\mathbf{A}| = \sum_{k=1}^n (-1)^{k+1} a_{1k} |\mathbf{A}_{1k}| \quad (2.49)$$

The determinant of the $n \times n$ matrix is computed by downsizing the matrix until 2×2 , iteratively. It takes the quadratic complexity in computing the determinant of the matrix in implementing it.

Let us consider finding the determinant from a diagonal matrix. It is one where its diagonal elements are non-zero values and its off-diagonal ones are zero values, and expressed as Eq. (2.50),

$$\mathbf{A}_{diag} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \quad (2.50)$$

In the diagonal matrix, Eq. (2.49) for finding the determinant is simplified into Eq. (2.51),

$$|\mathbf{A}_{diag}| = \prod_{i=1}^n a_{ii} \quad (2.51)$$

Because it is easier to compute the determinant in the diagonal matrix, it is necessary to map a matrix into its diagonal matrix through SVD (Singular Value Decomposition) which is covered in Sect. 2.4.3. The diagonal matrix may be preferred to the general matrix for modeling the machine learning algorithms.

Let us mention some properties of the determinant of the matrix, $\mathbf{A}$, $|\mathbf{A}|$. If at least, a row or a column is a zero vector, the determinant of the matrix is zero. If the matrix, $\mathbf{B}$, is obtained by interchanging any two columns or any two rows of the matrix, $\mathbf{A}$, the determinant of the matrix, $\mathbf{B}$, becomes the negative determinant of the matrix, $\mathbf{A}$, as shown in Eq. (2.52),

$$|\mathbf{B}| = -|\mathbf{A}| \quad (2.52)$$

If, at least, two columns or two rows are identical to each other, the determinant of the matrix becomes zero. If the matrix, $\mathbf{B}$, is obtained by multiplying any row or any column of the matrix, $\mathbf{A}$, by a scalar value, k , the determinant of the matrix, $\mathbf{B}$, is expressed as in Eq. (2.53),

$$|\mathbf{B}| = k|\mathbf{A}| \quad (2.53)$$

2.4.2 Eigen Value and Vector

The Eigen vector is used for characterizing a matrix by multiplying itself with a scalar which is called Eigen value. The multiplication of a $n \times n$ matrix and a $n \times 1$ Eigen vector is expressed as the multiplication of an Eigen value and an Eigen vector. The Eigen value, λ , and the Eigen vector, $\mathbf{v}$, are expressed as Eq. (2.54), to the matrix, $\mathbf{A}$, mathematically,

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \quad (2.54)$$

The matrix, $\mathbf{A}$, in the left side of Eq. (2.54), is assumed as a square matrix, and Eq. (2.54) is used for diagonalizing a matrix into a simplified one. This section is intended to describe the Eigen vectors and values in the context of the linear algebra.

Let us consider the Eigen vectors and values in a 2×2 matrix, for the simplicity. The 2×2 matrix is expressed by Eq. (2.55),

$$\mathbf{A}_{2 \times 2} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad (2.55)$$

and its Eigen vector is expressed by Eq. (2.56),

$$\mathbf{v} = \begin{bmatrix} v_1 & v_2 \end{bmatrix} \quad (2.56)$$

The Eigen vector and the Eigen vector to the 2×2 matrix are expressed as Eq. (2.57),

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \lambda \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad (2.57)$$

Equation (2.57) is expanded into Eq. (2.58),

$$\begin{bmatrix} a_{11}v_1 + a_{12}v_2 \\ a_{21}v_1 + a_{22}v_2 \end{bmatrix} = \begin{bmatrix} \lambda v_1 \\ \lambda v_2 \end{bmatrix} \quad (2.58)$$

and Eq. (2.59) is derived from Eq. (2.58),

$$\begin{bmatrix} (a_{11} - \lambda)v_1 + a_{12}v_2 \\ a_{21}v_1 + (a_{22} - \lambda)v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (2.59)$$

The left side in Eq. (2.59) is expressed as Eq. (2.60),

$$\begin{bmatrix} (a_{11} - \lambda)v_1 + a_{12}v_2 \\ a_{21}v_1 + (a_{22} - \lambda)v_2 \end{bmatrix} = \left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = (\mathbf{A}_{2 \times 2} - \lambda \mathbf{I})\mathbf{v} \quad (2.60)$$

Let us consider the Eigen vectors and values to a general square matrix. Equation (2.61) is derived from Eq. (2.60), representing the Eigen vector and value as Eq. (2.54),

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = \mathbf{0} \quad (2.61)$$

The determinant of the matrix, $\mathbf{A} - \lambda \mathbf{I}$, is zero for finding the Eigen vectors and values, as is expressed as Eq. (2.62),

$$|\mathbf{A} - \lambda \mathbf{I}| = 0 \quad (2.62)$$

The determinant of the matrix, $\mathbf{A}_{2 \times 2} - \lambda \mathbf{I}_{2 \times 2}$, is zero in the case of 2×2 matrix as shown in Eq. (2.63),

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = (a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} = 0 \quad (2.63)$$

The Eigen vector, $\mathbf{v}$, is found by applying Eq. (2.54), after finding the Eigen values, λ .

Let us mention some mathematical theorems on the Eigen vectors and values of a matrix. A triangular matrix, $\mathbf{A}$, is given as Eq. (2.64),

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad (2.64)$$

and its diagonal elements are given as its Eigen values. If the Eigen values are non-zeros, the matrix is invertible. If an Eigen value of the matrix, $\mathbf{A}$, is given as λ , the Eigen value of its inverse matrix is given as $\frac{1}{\lambda}$. If the Eigen value of the matrix, $\mathbf{A}$ is a diagonal matrix, are given as its Eigen values. If the Eigen values are non-zeros, the matrix is invertible. If an Eigen value of the matrix, $\mathbf{A}$, is given as, the Eigen value of the matrix, $\mathbf{A}^n$, is λ^n .

Let us make some remarks on the Eigen vectors and values of a square matrix which is covered in this section. The matrix is assumed to be a square matrix for finding its Eigen vectors and values. In the usual case, from a $n \times n$ matrix, the n Eigen values, $\lambda_1, \lambda_2, \dots, \lambda_n$, and the n Eigen vectors, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$. Each Eigen vector is associated with its own Eigen value as shown in Eq. (2.54). The Eigen vectors and values of the matrix are used for diagonalizing itself.

2.4.3 Singular Value Decomposition

The SVD (Singular Value Decomposition) is defined as the process of decomposing a particular matrix into the three matrices: the matrix whose column vectors are the Eigen vectors, the diagonal matrix whose diagonal elements are the Eigen values, and the matrix whose row vectors are the Eigen vectors. In Sect. 2.4.2, we studied the Eigen vectors and values of a matrix. A matrix is decomposed into the three matrices as expressed as Eq. (2.65),

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V} \quad (2.65)$$

where $\mathbf{A}$, $\mathbf{U}$ is the matrix where the Eigen vectors are given as its column vectors by $\mathbf{A}^T\mathbf{A}$, $\mathbf{D}$ is the diagonal matrix whose diagonal elements are the Eigen values of the matrix, and $\mathbf{V}$ is the matrix where the Eigen vectors are given as its row vectors by $\mathbf{A}\mathbf{A}^T$. The SVD is used for reducing the dimension of the numerical vector which represents a data item. This section is intended to describe the SVD which is based on the Eigen vectors and values.

The N training examples, each of which is given as a d dimensional vector, are viewed as a $N \times d$ matrix. The training set is notated as a set, $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and each training example is given as a d dimensional vector, $\mathbf{x}_i = [x_{i1} \ x_{i2}, \dots \ x_{id}]$. The training set is represented as a $N \times d$ matrix as shown in Eq. (2.66),

$$\mathbf{Tr} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ x_{N1} & x_{N2} & \dots & x_{Nd} \end{bmatrix} \quad (2.66)$$

where each column corresponds to an attribute, and each row corresponds to a training example. By swapping the column and the row with each other, we consider its transpose matrix, $\mathbf{Tr}^T$, as shown in Eq. (2.67),

$$\mathbf{Tr}^T = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2N} \\ \dots & \dots & \dots & \dots \\ x_{d1} & x_{d2} & \dots & x_{dN} \end{bmatrix} \quad (2.67)$$

the $d \times d$ matrix is generated by the multiplication, $\mathbf{Tr}^T\mathbf{Tr}$. As it is focused by this section, the matrix, $\mathbf{Tr}$, is decomposed into its three component matrices.

Let us mention the steps of performing the SVD to a particular matrix. The N Eigen vectors are computed from the $N \times N$ matrix, $\mathbf{Tr}\mathbf{Tr}^T$, and the $N \times N$ matrix, $\mathbf{U}$, is constructed by arranging the Eigen vectors as its column vectors. The N Eigen values are computed from the Eigen vectors to the matrix, $\mathbf{Tr}\mathbf{Tr}^T$, and the $N \times d$

matrix, is constructed by arranging the Eigen values as its diagonal elements. The d Eigen vectors are computed from the $d \times d$ matrix, $\mathbf{Tr}^T \mathbf{Tr}$, and the $d \times d$ matrix, $\mathbf{V}$, is constructed by arranging the Eigen vectors as its row vectors. The item-attribute matrix which represents the training examples, $\mathbf{Tr}$ is decomposed into the three matrices as shown in Eq. (2.68),

$$\mathbf{Tr} = \mathbf{U} \mathbf{\Sigma} \mathbf{V} \quad (2.68)$$

This decomposition is called SVD (Singular Value Decomposition).

The training examples with their reduced dimensions are generated by selecting maximum d Eigen values from the matrix, $\mathbf{\Sigma}$. It is given as a $N \times d$ matrix as shown in Eq. (2.68). The matrix, $\mathbf{\Sigma}'$, is constructed by filling the maximal d' Eigen values, $d' < d$ from the matrix, $\mathbf{\Sigma}$, and zero values. The matrix, $\mathbf{Tr}'$, where the training examples with their reduced size are given as its row vectors, is constructed by Eq. (2.69),

$$\mathbf{Tr}' = \mathbf{U} \mathbf{\Sigma}' \mathbf{V} \quad (2.69)$$

In the matrix, $\mathbf{Tr}'$, the d' non-zero column vectors are given and the $d - d'$ zero column vectors are given.

Let us make some remarks on the SVD which is covered in this section. Understanding the Eigen vectors and values of a matrix becomes the basis for studying the SVD. The SVD is the decomposition of a matrix into the three matrices which consist of its Eigen vectors and values, as shown in Eq. (2.68). In Eq. (2.68), $\mathbf{U}$ and $\mathbf{V}$ are the matrices which consist of the Eigen vectors, respectively, as the column vectors and the row vectors, and $\mathbf{\Sigma}$ is the diagonal matrix whose diagonal elements are given as the Eigen values. The dimension is reduced, using the SVD, by removing the minimal Eigen values in the matrix, $\mathbf{\Sigma}$.

2.4.4 Principal Component Analysis

This section is concerned with the PCA (Principal Component Analysis) where the theory on the matrix is applied to the dimension reduction. The training set which consists of the d dimensional vectors is viewed as a matrix in the SVD (Singular Value Decomposition). In the PCA, the covariance matrix on the training examples is set as the basis matrix. The Eigen vectors which are associated with the maximal Eigen values are selected as many as the reduced dimension as the dimension reduction process. This section is intended to describe the PCA as the alternative scheme of reducing the dimension of numerical vectors to the SVD.

The first step of the PCA is to compute the covariance matrix of the training examples. The training set is notated by $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, each training example is notated by $\mathbf{x}_i = [x_{i1} \ x_{i2}, \dots \ x_{id}]$, as a d dimensional numerical vector,

and the mean vector of the training examples is notated by $\bar{\mathbf{x}} = [\bar{x}_1 \ \bar{x}_2, \dots \ \bar{x}_d]$. Each element of the covariance matrix, var_{ij} , is computed by Eq. (2.70),

$$var_{ij} = \frac{1}{N} \sum_{k=1}^N (\bar{x}_i - x_{ki})(\bar{x}_j - x_{kj}) \quad (2.70)$$

The $d \times d$ covariance matrix is constructed as shown in Eq. (2.71), by iterating computing all elements by Eq. (2.70),

$$\mathbf{Cov} = \begin{bmatrix} var_{11} & var_{12} & \dots & var_{1d} \\ var_{21} & var_{22} & \dots & var_{2d} \\ \dots & \dots & \dots & \dots \\ var_{d1} & var_{d2} & \dots & var_{dd} \end{bmatrix} \quad (2.71)$$

The dimension is reduced by the PCA based on the Eigen vectors and values of the covariance matrix.

The Eigen vectors and values are computed from the matrix, $\mathbf{Cov}$. The Eigen values are $\lambda_1, \lambda_2, \dots, \lambda_d$, and the Eigen vectors are $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_d$. The Eigen values are ranked in the descending order and the d' maximal Eigen values are selected as $\lambda_1^{\max}, \lambda_2^{\max}, \dots, \lambda_{d'}^{\max}, d' < d$. The d' Eigen vectors which correspond to the selected Eigen values are retrieved as $\mathbf{v}_1^{\max}, \mathbf{v}_2^{\max}, \dots, \mathbf{v}_{d'}^{\max}$, and the matrix, $\mathbf{W}$ is constructed by setting the Eigen vectors as its row vectors, as shown in Eq. (2.72),

$$\mathbf{W} = \begin{bmatrix} \mathbf{v}_1^{\max} \\ \mathbf{v}_2^{\max} \\ \dots \\ \mathbf{v}_{d'}^{\max} \end{bmatrix} = \begin{bmatrix} v_{11}^{\max} & v_{12}^{\max} & \dots & v_{1d}^{\max} \\ v_{21}^{\max} & v_{22}^{\max} & \dots & v_{2d}^{\max} \\ \dots & \dots & \dots & \dots \\ v_{d'1}^{\max} & v_{d'2}^{\max} & \dots & v_{d'd}^{\max} \end{bmatrix} \quad (2.72)$$

The matrix, $\mathbf{W}$, is used for generating the reduced dimensional vector in the next step.

The reduced dimensional vectors are generated by projecting the original training examples with the above matrix, $\mathbf{W}$. The mean vector and the covariance matrix are computed by the above process, and the matrix, $\mathbf{W}$, is constructed by finding the Eigen vectors and values from the covariance matrix. A d' dimensional vector is computed by the product of the matrix, $\mathbf{W}$, and the difference between the mean vector and a training example, as shown in Eq. (2.73),

$$\mathbf{z}_i = \mathbf{W}(\mathbf{x}_i - \bar{\mathbf{x}}) \quad (2.73)$$

The projected examples, $Tr_p = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N\}$, are retrieved from the training examples, $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ by applying Eq. (2.73) to each training example. The set of the projected examples, whose dimension is d' , is used actually for training the machine learning algorithms.

Let us make some remarks on the PCA which is presented as a scheme of reducing the dimension of the numerical vectors. The computation is very expensive for reducing the dimensionality by the PCA. The covariance matrix is constructed from the N training examples with the quadratic complexity, $O(N^2)$. It is very expensive to compute the Eigen vectors and values from the covariance matrix. This dimension reduction scheme is actually not adopted in the text mining tasks where texts are encoded into huge dimensional numerical vectors, because of very much computation time.

2.5 Summary and Further Discussions

Let us summarize what concerns this chapter. We studied the vector which represents a raw data item in using the machine learning algorithms with respect to its definition and the operations on them. We studied the matrix which is expanded from a vector with respect to identical ones. We covered the relation between a matrix and a vector and the two schemes of reducing the dimensionality. This chapter covers the mathematical definition and characterization of a vector and a matrix as the basis for studying the machine learning algorithms.

Let us view the matrix as a vector set. A given matrix is assumed as a $m \times n$ matrix and viewed as a vector set with the two directions: the column based direction and the row based direction. The matrix consists of n m dimensional vectors as column vectors, or m n dimensional vectors as row vectors. If k d dimensional vector is represented as a set, it is viewed as a $k \times d$ matrix or a $d \times k$. A vector is viewed as a finite ordered set of scalar values.

Let us mention the tensor as the expansion from the matrix. A vector is an array of numerical values in one dimension, and a matrix is an array of such values in two dimensions. A tensor is an array of values in more than two dimensions. A tensor as a three dimensional array is viewed as a list of matrices. In using the convolutionary neural networks as a deep learning algorithm, an input or an encoded one is represented as a tensor.

Let us consider the inner product between two vectors as the non-normalized similarity between them. The cosine similarity is normalized by product of norms of two vectors between zero and one. If two vectors are directed to perpendicular, the inner product between them becomes zero; the similarity between them becomes minimum. The similarity measurement based on the inner product depends on the angle between two vectors. The Euclidean distance depends on the distance in the Cartesian coordination system between them.

Let us consider the matrix, called submatrix, which is a part of a particular matrix. Let us assume the matrix with $n \times m$ size which is initially given as the original matrix. The size of its submatrix is $n' \times m'$ where $n' \leq n$ and $m' \leq m$. All elements of the submatrix are included in the original matrix. We may extract more than one submatrix from the original matrix where $m > 1$ and $n > 1$.

Chapter 3

Data Encoding



3.1 Introduction

The data encoding is defined as the process of converting raw data into numerical vectors as structured data. A numerical vector is always given as the input in using machine learning algorithms for real problems. The raw data is almost always given differently from numerical vector, and it is necessary to convert the raw data into numerical vectors, in order to apply machine learning algorithms. In this chapter, we mention the three representative types of raw data: relational data, textual data, and image data. This section is intended to describe briefly the three types of raw data.

The data is given as a list of records, called a table in the area of database, and this kind of data is called relational data. Each record consists of fields that correspond to attributes, and a table consists of more than one record. Each field is given as a single value such as a Boolean, an integer, a real value, or a short string, or a composite value. The primary key is the unique field value that identifies its own record, and the secondary key is one that characterizes an individual record or records. The relational data is the traditional data type and the form that is converted easily into numerical vectors.

The textual data has become dominant since 2000s with the expansion of Internet. A textual data item is given as an article that consists of more than one paragraph, which is written in a natural language. A word, a sentence, or a paragraph may belong to textual data in the broad meaning. A sentence is constructed by combining words, following the grammatical rules, and a paragraph is built by combining sentences, semantically. We need techniques of encoding textual data items into structured forms such as numerical vectors, in order to apply machine learning algorithms to text mining tasks.

The image data is dominant, together with the textual data in the real world. An image is viewed as a big sized matrix that consists of elements called pixels. Each pixel is given as a numerical value or numerical values that indicate a gray scale or color information, which consists of the three values, namely, red, green, and blue.

The image looks easy for encoding itself into a numerical vector, but sophisticated feature selection methods are required for more efficient representations. Various standard formats of image file are available as gif, png, and bmp.

The goal of this chapter is to understand the three types of raw data, that is, the relational data, the textual data, and the image data, together with their encoding processes. We need to understand the relational data as the traditional one and the process of encoding them into numerical vectors. We will understand the textual data such as dominant and unstructured one and the process of encoding them. We will also understand the image data that is dominant, together with the textual data, and the process of mapping them into numerical vectors. The assumption that is inherent in the machine learning algorithms is that their input data is always given as a numerical vector.

3.2 Relational Data

This section is concerned with the relational data that is initially given as raw data and the process of encoding them into numerical vectors. We mention the basic concepts of relational data and relational database, respectively, in Sects. 3.2.1 and 3.2.2. We describe the process of mapping the relational data into numerical vectors, entirely in Sect. 3.2.3. We mention some of the issues of encoding in Sect. 3.2.4. This section focuses on encoding of raw data, relational data, into numerical vectors.

3.2.1 Basic Concepts

The relational data is defined as a table that is given as the collection of records. Each record that consists of several fields indicates as an information about item, and each field corresponds to an individual value. Individual records are identified by their own primary keys that are unique. Index records each of which consists of its own primary key and its storage location are constructed for accessing data records easily. This section is intended to describe data records, primary keys, and index records as the basic concepts of relational data.

The data records are illustrated in Fig. 3.1 as examples. A data record refers to one that represents a real item with its full fields. As shown in Fig. 3.1, each record represents a research paper, and the four data records are given as a table. The five fields in each record are paper no., author, title, session, and affiliation. We need the two special characters that are not used as field values, as the field boundary and the record boundary, for storing the records as a long string in a file.

The first column in the table that is presented in Fig. 3.2 belongs to the primary key. It refers to the field for identifying uniquely individual records. All values in the primary key are required to be unique. The field “Paper No.” corresponds to the

Paper No	Author	Title	Session	Affiliation
ICN_001	Taeho Jo	Understanding of Text Mining	Data Mining	Hongik University
ICN_002	Michael Jordan	Understanding of Neural Networks	Artificial Intelligence	Stanford University
ICN_003	Taeho Jo	Understanding of Machine Learning	Artificial Intelligence	Hongik University
ICN_004	Kai Wang	Understanding of Machine Learning	Artificial Intelligence	Beijing University

Fig. 3.1 Data record

Primary Key



Paper No	Author	Title	Session	Affiliation
ICN_001	Taeho Jo	Understanding of Text Mining	Data Mining	Hongik University
ICN_002	Michael Jordan	Understanding of Neural Networks	Artificial Intelligence	Stanford University
ICN_003	Taeho Jo	Understanding of Machine Learning	Artificial Intelligence	Hongik University
ICN_004	Kai Wang	Understanding of Machine Learning	Artificial Intelligence	Beijing University

Fig. 3.2 Primary key

Fig. 3.3 Index record

Primary Key	Data Record Location
ICN_001	0
ICN_002	851
ICN_003	1721
ICN_004	2916

primary key in this example. There is the case of combining several key values into a primary key.

The index records are illustrated in Fig. 3.3. The index record is the record that consists of the two fields: the primary key and its location. The index records are used for finding data records easily, and the field location indicates the position of its corresponding data record in the file. This type of index records is called primary index records, and data records are retrieved by the primary key that is given as the query. In the reality, searching for data records by the primary key is rare.

Let us make some remarks on the relational data. It is the collection of records that characterizes items individually and is called a table. The integrity in the relational data indicates the correctness degree; lower noise in the data indicates the higher integrity. The data type of each field is usually a primitive data type: integer, Boolean, real value, or short string. It is easier to encode the relational data into numerical vectors, compared with other types of raw data.

3.2.2 Relational Database

The relational database is defined as the collection of relational data that is separated from application programs. The database is the data collection that may be accessed by any application program through the DBMS (Database Management System). The relational algebra is defined as the basic theory for manipulating the relational data, and the SQL is defined as the query language for doing it. The SQL state is processed by the DBMS for accessing the relational data. This section is intended to describe the relational database, briefly.

The file structure before proposing the database is illustrated in Fig. 3.4. The K application programs exist as shown in Fig. 3.4. Each application program provides accesses to their data file which is dependent on it, in the file structure. Redundant data may be created in accessing the same by the different application programs. The inefficiency in the file structure becomes the motivation for proposing the database that is accessed by the application programs.

The DBMS and its relation with the application programs are presented in Fig. 3.5. The application programs that are executed by a user share the database.

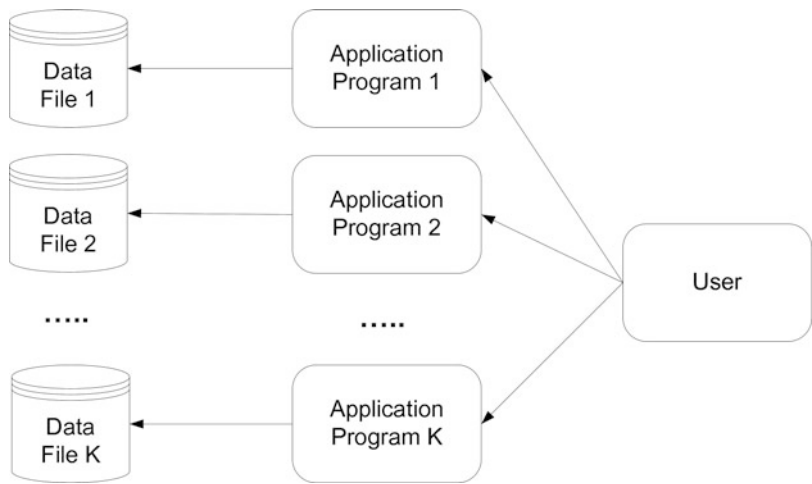


Fig. 3.4 File structure

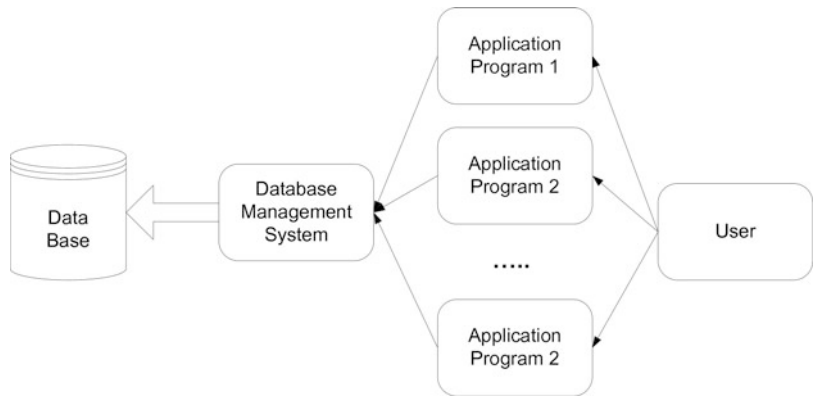


Fig. 3.5 Database Management System

Fig. 3.6 SQL statements

```
SELECT fieldList from tableName
where attributeConditionList

SELECT Author, Title, from paperList
where Author = "Taeho Jo"
```

The DBMS plays the role of the interface between the database and the applications. Various forms of queries from the application programs exist, and they are mapped into the SQL statements as the query language for accessing the database. The SQL statements are studied in detail in [1].

The SQL statement frame and its example are illustrated for accessing data items in the relational database in Fig. 3.6. The top statement indicates the SQL statement frame, and the bottom statement indicates a specific example. SELECTION in the statement is the command for accessing data items in the relational database, fieldList is the list of fields of data items that are accessed, tableName is the name of table that is the source of data items, and attributeConditionList is the conditions that satisfy data items, in the frame. The example of SQL statement means accessing data items that satisfy the value of the field author, which is equal to the string “TaehoJo” from the table, named by “paperList,” with the fields: author and title. The SQL statement that is the query for accessing the relational data base is generated by an application program, rather than by users.

Let us make some remarks on the relational database that is covered in this section. The table name, the field list, and the data type and the constraint in each field are necessary for defining the relational database. Because the field values are given in various scaled ones, we need to normalize them for encoding the relational data into numerical vectors. In the string-valued field, the numerical code is defined and assigned to each string. A long string-type field value will be treated as the textual data in Sect. 3.3.

3.2.3 Encoding Process

This section is concerned with the process of encoding the relational data into numerical vectors. In studying the machine learning algorithms, it is absolutely assumed that the input data is given as a numerical vector. Even if the relational data looks similar as a numerical vector, it should be encoded into a complete numerical vector. Not all field values are numerical vectors; values of some fields are given as string values. This section is intended to describe the necessary steps for encoding relational data into numerical vectors.

It is necessary to remove noises in relational data, which is shown in Fig. 3.7, for encoding relational data into numerical vectors. The value 38.45 in the field age violates against the fact that an age should be given as an integer value that is greater than or equal to zero. The value -15,000 in the field salary is given as a negative value; it violates against the fact that the income should be as a positive value by our common sense. The value 480 in the field age is too large against the life expectancy. The values should be treated as noises in the relational data.

The process of normalizing field values into ones between zero and one is illustrated in Fig. 3.8. In this case, it is assumed that each record consists of fields whose values are continuous. In each record, we take the maximum and the minimum, and the rate of the difference between a current value and the minimum and to the difference between the maximum and the minimum becomes its normalized value. In the current value, zero indicates the minimum and one indicates the maximum; the output value is generated as a normalized value, in the regression. Mapping the normalized value into the original value is called denormalization.

No	Name	Age	Salary	Position
1	Tom Smith	35.45	\$40,000	Junior Manager
2	Mary Tailor	38	-\$15,000	Manager
3	Taeho Jo		\$40,000	Technical Advisor
4	Judy King	480	\$90,000	CEO

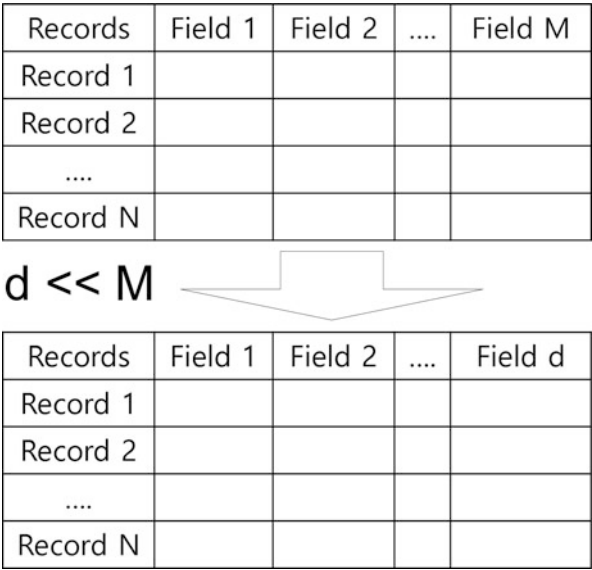
Fig. 3.7 Noises in relational data

No	Field
1	Value 1
2	Value 2
...	
N	Value N

$$Normalized_Value = \frac{Value - Min_Value}{Max_Value - Min_Value}$$

Fig. 3.8 Normalization of field value

Fig. 3.9 Dimension reduction



The process of reducing the dimension from M to d is illustrated in Fig. 3.9. Not all attributes of each item are necessary for doing the classification, the regression, or clustering. The correlations of attribute values with the categories or the output values are analyzed, and only the attributes with their correlation efficient close to -1 or 1 are selected, in doing the classification or the regression. The attributes with very small number of discrete values or continuous values with their small ranges are removed in doing the clustering. The process of generating attributes is called feature extraction, and the process of selecting only some of them is called feature selection.

Let us make some remarks on the process of encoding the relational data into numerical vectors. The relational data item looks similar as a numerical vector; it is possible to use a relational data item as a numerical vector by itself. In the reality, the noise removal, the field value normalization, and the dimension reduction are required for encoding so. The field value whose value is given as a string should be mapped into a numerical value; this process is called codification. It is easier to encode relational data into numerical vectors, than to do textual data or image data that are covered in the subsequent sections.

3.2.4 Encoding Issues

This section is concerned with some issues in encoding relational data into numerical vectors. In the previous sections, we mention the relative easiness of encoding, compared with encoding the textual data or the image data. Because the

No	Name	Age	Salary	Position
1	Junior Manager	Tom Smith	35	\$30,000
2	Mary Tailor	38	\$15,000	Manager
3	Taeho Jo		\$40,000	Technical Advisor
4	Judy King	48	\$90,000	CEO

Fig. 3.10 Unreliable values

No	Name	Age	Salary	Position
1	Tom Smith	35	\$30,000	
2	Mary Tailor		\$15,000	Manager
3	Taeho Jo		\$40,000	Technical Advisor
4	Judy King	48	\$90,000	CEO

Fig. 3.11 Missing values

integrity of the relational database is always incomplete, it is impossible to avoid problems in encoding even relational data. We need to solve the issues that are mentioned in this section, for gathering reliable training set and reliable test set. This section is intended to present the three main issues that are considered in encoding relational data.

The unreliable values that are presented in Fig. 3.10 are issues that are considered in the relational data into numerical vectors. The person name “Tom Smith” in the field age of the first record in Fig. 3.10 is an instance of the unreliable data. The numerical value 35 without the dollar sign in the field salary is another instance. The value \$30,000 in the field position that requires a string is the third example. In the first record, the values of the record should be arranged into fields by shifting one.

The relational data with missing values is illustrated in Fig. 3.11. The values in the field position in the first record and the field age in the second and the third are not available. If attributes are not used for the classification, the clustering, or the regression, there is no problem. If they are considered, we need to estimate the missing values. They are approximated by taking the most similar record.

Let us consider the case of encoding records from different tables shown in Fig. 3.12. The two tables are merged into one table, and the relational data may be encoded into numerical vectors. Different field names with the same meaning should be discovered in both the tables; the different field names “salary” and “monthly payment” are assigned to the two tables. One-to-one matching of fields is needed for merging the tables into their standard form. We need to develop a similarity metric between fields with respect to the synonym between different field names, formats, and value scales.

Not all fields are necessary while doing the data mining tasks such as the data classification, the regression, or the clustering. While encoding the relational data,

No	Name	Age	Salary	Position
1	Tom Smith	35	\$30,000	Assistant Manager
2	Mary Tailor	34	\$45,000	Manager
3	Taeho Jo	49	\$40,000	Technical Advisor
4	Judy King	48	\$90,000	CEO

No	Given + Sur	Age	Monthly Payment	Status
1	Tom Smith	35	\$30,000	Assistant Manager
2	Mary Tailor	34	\$45,000	Manager
3	Taeho Jo	49	\$40,000	Technical Advisor
4	Judy King	48	\$90,000	CEO

Fig. 3.12 Different field names

we need to remove unnecessary fields for more efficient processing. For each field, its correlation with its output value should be analyzed, and the fields whose correlations are close to zero should be removed. The information gain that is based on the information theory is used for detecting unnecessary fields. Removing unnecessary fields is very useful in the case of encoding relational data with many fields into small dimensional numerical vectors.

3.3 Textual Data

This section is concerned with the textual data as another kind of raw data. In Sect. 3.3.1, we describe the process of indexing a text into a list of words. In Sect. 3.3.2, we explain the process of encoding a text into a numerical vector as the next step of text indexing. In Sects. 3.3.3 and 3.3.4, we mention the dimension reduction and the issues in encoding, respectively. In this section, we focus on the process of mapping a text that is given as raw data into a numerical vector as structured data.

3.3.1 Text Indexing

The text indexing is defined as the process of mapping a text or texts into a list of words. A text is an article that consists of more than or equal to one paragraph and is

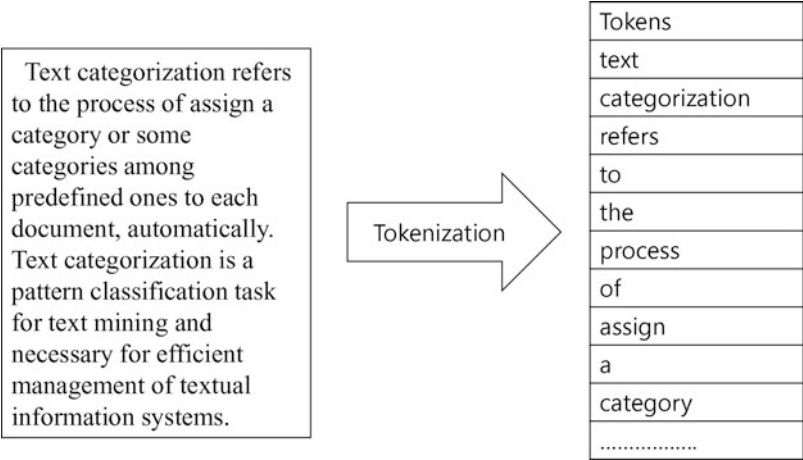


Fig. 3.13 Tokenization

written in a natural language. There are three basic steps of indexing a text or texts; the tokenization, the stemming rule, and the stop word removal are presented in Figs. 3.13, 3.14, and 3.15, respectively. Nouns, verbs, and adjectives become outputs of the text indexing through the three steps. This section is intended to describe the steps of indexing a text or texts.

The tokenization process is illustrated with a simple example in Fig. 3.14. It refers to the process of segmenting a text into a list of tokens by white space, punctuation marks, and other special characters. As shown in Fig. 3.14, the text that is written in English is partitioned into a list of tokens. All capital characters are converted into smaller ones, and tokens that include numbers or special characters are removed in this step. Because the grammatical variations are remaining in the results from the tokenization, more processing steps are needed for getting the words.

The process of stemming each token is illustrated in Fig. 3.14. A text is partitioned into tokens in the previous step, and each token in Fig. 3.14 is marked by the slash. Each token is mapped into its root form; plural nouns are mapped into singular ones, and verbs with their variations are mapped into their root forms. Adverbs with their postfix “ly” are mapped into adjectives by removing the postfix. Nouns with the postfix “tion” are mapped into verbs by removing the postfix, optionally.

The marks of stop words in the text are illustrated in Fig. 3.15. The stop word means the word that functions only grammatically regardless of text contents; pronouns, prepositions, conjunctions, and so on belong to stop word. The process of removing stop words for more efficiency is called stop word removal. This step may be swapped with the previous step, in indexing a text or texts. Nouns, verbs, and adjectives are remaining after the three basic steps.

Text/ categorization/ refers→refer/ to/ the/ process/ of/ assign/ a/ category/ or/ some/ categories→category /among /predefined→predefine/ ones→one/ to/ each/ document/, automatically→automatic/. Text/ categorization/ is/ a/ pattern/ classification/ task/ for/ text/ mining/ and/ necessary/ for/ efficient/ management/ of/ textual/ information /systems→system/. In/ the/ academic/ world/, research/ on/ text/ categorization/ has→have/ been→be/ progressed→progress/ very/ much/, and/ we/ will/ survey/ it/ in/ next/ section/. In/ the/ industrial/ world/, text/ categorization/ systems→system /were→be/ already/ developed→develop/ as/ an/ independent/ system/ or/ a/ module/ for/ textual/ information/ systems→system/. /Although/ research/ and/ development/ on/ text/ categorization/ have/ been→be / progressed→progress/ like/ this/, /we/ need/ further→far/ research/ on/ it/ to/ improve/ techniques→technique/ and/ implementations→implementation/ of/ text/ categorization/.

Fig. 3.14 Stemming rule

Text/ categorization/ refers→refer/ to/ the/ process/ of/ assign/ a/ category/ or/ some/ categories→category /among /predefined→predefine/ ones→one/ to/ each/ document/, automatically→automatic/. Text/ categorization/ is/ a/ pattern/ classification/ task/ for/ text/ mining/ and/ necessary/ for/ efficient/ management/ of/ textual/ information /systems→system/. In/ the/ academic/ world/, research/ on/ text/ categorization/ has→have/ been→be/ progressed→progress/ very/ much/, and/ we/ will/ survey/ it/ in/ next/ section/. In/ the/ industrial/ world/, text/ categorization/ systems→system /were→be/ already/ developed→develop/ as/ an/ independent/ system/ or/ a/ module/ for/ textual/ information/ systems→system/. /Although/ research/ and/ development/ on/ text/ categorization/ have/ been→be / progressed→progress/ like/ this/, /we/ need/ further→far/ research/ on/ it/ to/ improve/ techniques→technique/ and/ implementations→implementation/ of/ text/ categorization/.

Fig. 3.15 Stop word removal

Let us mention the additional steps to the three basic ones for indexing a text or texts. The weight that indicates the importance of the text is assigned to each of the remaining words. The additional filtering of terms by their weights may be considered; in addition, terms with their lower weights may be removed. Towards the words with their higher weights, their semantically similar words are added as their associated words from external sources; this is called index expansion. We need to classify each word into one of the three categories: expansion, inclusion, and removal; this is called index optimization in [2].

3.3.2 Text Encoding

This section is concerned with the next step after the text indexing for processing textual data. The text encoding is defined as the process of mapping texts into structured data, such as numerical vectors. A text is encoded into a numerical vector with the three steps: the feature extraction, the feature selection, and the feature value assignment. Some essential words in the corpus are used as features for encoding texts. This section is intended to explain the three steps for encoding a text into a numerical vector.

The process of extracting feature candidates from the corpus is illustrated in Fig. 3.16. The process is the same as that of indexing a text, which is illustrated in Figs. 3.13, 3.14, and 3.15. It consists of the tokenization that segments a text into tokens, the stemming of each token, and the removal of stop words. According to the previous literature, more than ten thousand words are extracted as feature candidates. It is not feasible to use all words as features; we need to select only some among them.

The criteria for selecting some among the feature candidates as features are illustrated in Fig. 3.17. The given task is assumed to be a binary classification, and the total frequency for each feature candidate in the positive class and the negative class may be considered. The entropy of both classes may be computed for each feature candidate. Some words with their less entropy are selected among them. According to the previous literature, about 300 words are usually selected as features [3, 4].

The three schemes of assigning a value to each feature are illustrated in Fig. 3.18. The first scheme is to assign 0 that indicates the absence of the features in the text, or 1 that indicates the presence. The second scheme is to assign a relative frequency of the word in the given text. The third scheme is to compute TF-IDF (Term Frequency-Inverse Document Frequency) by the equation that is presented in Fig. 3.18, and to assign the value to each feature. The process of encoding a text into a numerical vector is accomplished by assigning values to features with one among the three schemes.

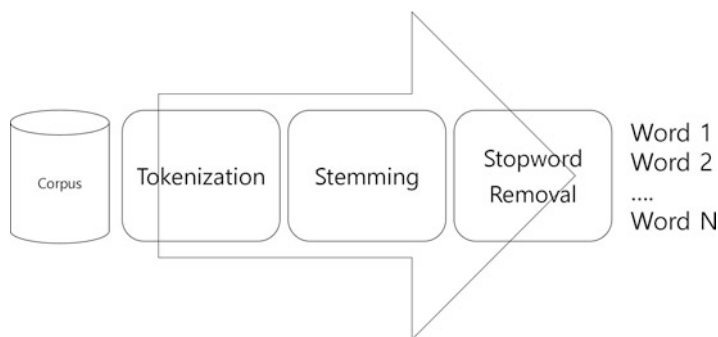


Fig. 3.16 Feature extraction

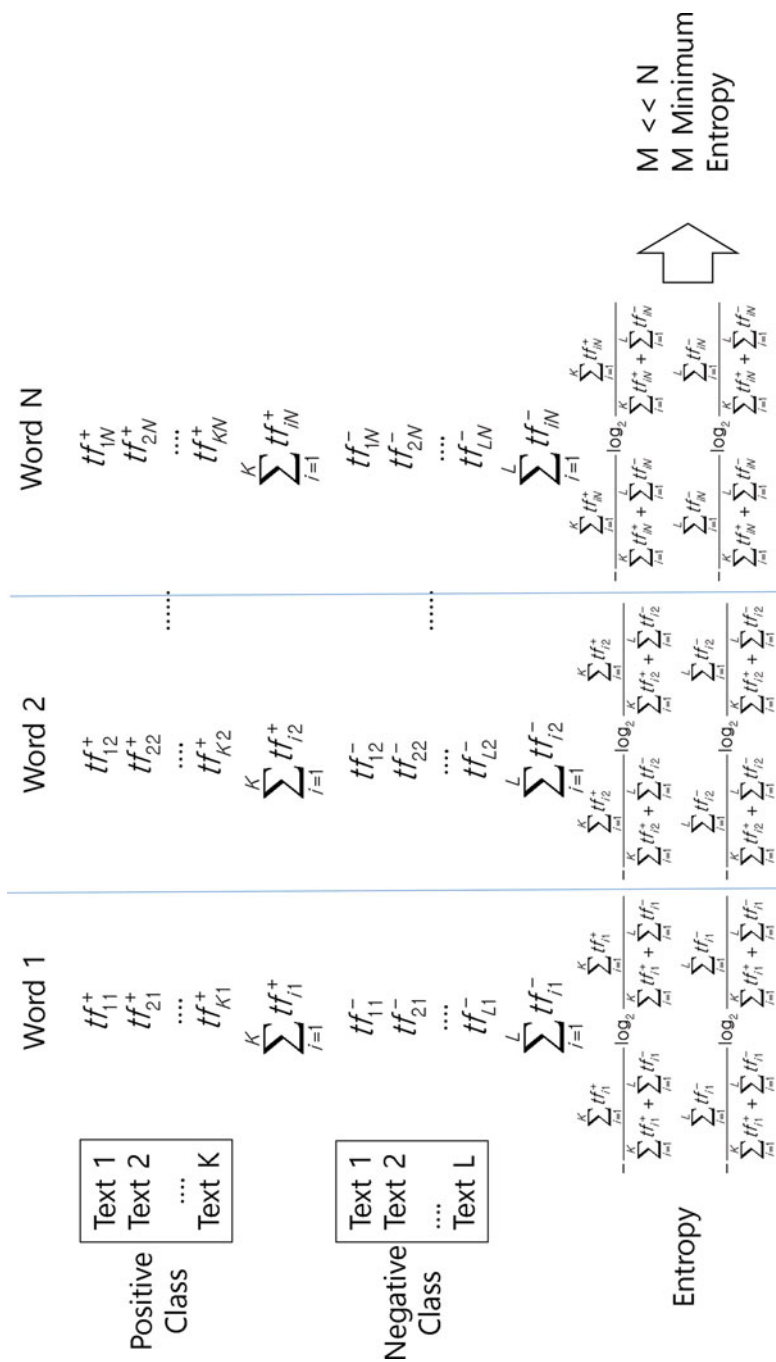


Fig. 3.17 Feature selection

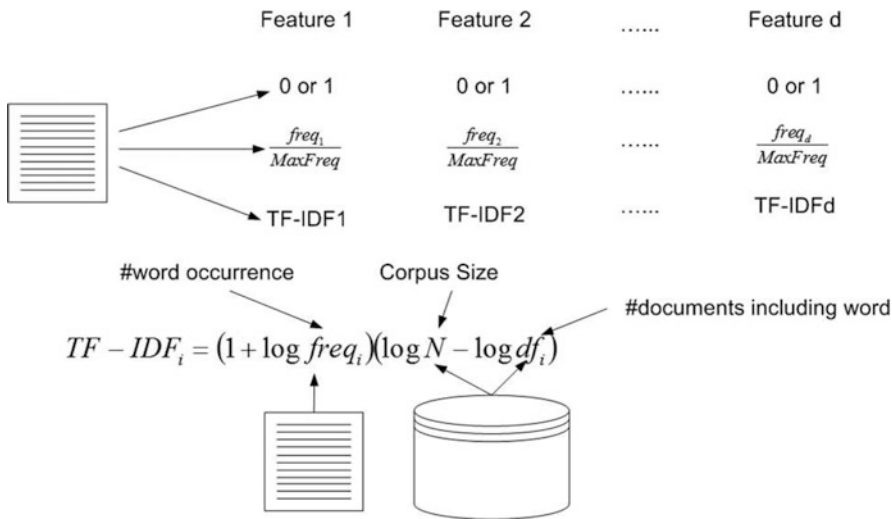


Fig. 3.18 Feature value assignment

Let us consider some issues in encoding a text into a numerical vector. Three hundred features are usually selected among ten thousand feature candidates; this dimension is still huge for processing the numerical vectors. The distribution in each numerical vector representing a text is very sparse; zero values are dominant with over 90%. The numerical vector that represents a text is not symbolic by itself. The references to the corpus are required for performing the above steps of the text encoding.

3.3.3 Dimension Reduction

This section is concerned with the schemes of reducing the dimension of numerical vectors that represent texts. Ten thousand feature candidates are extracted from the corpus as usual case, by indexing it. In the previous works, ten thousand dimensional vectors in using all of the feature candidates were reduced to about three hundred dimensional vectors for representing texts. Many kinds of dimension reduction schemes have been proposed. This section is intended to mention the three typical schemes of reducing the dimension: the singular value decomposition, the principal component analysis, and the independent component analysis.

Let us mention the scheme of reducing the dimension of a vector, called singular value decomposition. The set of n training examples each of which is represented in a N dimensional vector is viewed as $n \times N$ matrix. The matrix is divided into three components: the $n \times n$ square matrix that consists of n Eigen vectors as its column vectors, the $n \times N$ diagonal matrix that consists of singular values as its diagonal

elements, and the $N \times N$ square matrix that consists of N Eigen vectors as its row vectors. In the second component, d columns are selected by ranking its singular values. The selected columns correspond to the selected features.

Let us mention the second traditional scheme of reducing the dimension, called principal component analysis. The training set is expressed as a matrix in the above scheme. In this scheme, the mean vector and the covariance matrix are computed from the training examples. The Eigen vectors and the Eigen values are extracted from the covariance matrix, the d maximum Eigen values where d is much less than N are selected, and $n \times d$ matrix is constructed with the Eigen vectors that are associated with the selected Eigen values. The d dimensional vectors are extracted by multiplying the above matrix by n dimensional difference vector between the original training example and the mean vector, as the downsized training examples.

Let us explain the third traditional scheme of reducing the dimension, called independent component analysis. The scheme is based on the Eq. (3.1):

$$\mathbf{x} = \mathbf{A}\mathbf{x}', \quad (3.1)$$

where $\mathbf{x}$ is the N dimensional vector, $\mathbf{A}$ is the $N \times d$ matrix, and $\mathbf{x}'$ is the d dimensional vector. The pseudo inverse matrix is computed, instead of the inverse matrix of $\mathbf{A}$, by Eq. (3.2):

$$\mathbf{x}' = \mathbf{A}^+ \mathbf{x}, \quad (3.2)$$

where $\mathbf{A}^+$ is the $d \times N$ pseudo inverse matrix of $\mathbf{A}$. The matrix $\mathbf{A}$ is constructed by clustering the training examples into d subgroups and arranging the cluster prototypes as the column vectors. By the Gaussian elimination, we find the pseudo inverse matrix $\mathbf{A}^+$.

In this section, we mentioned the three traditional schemes of reducing the dimension of numerical vectors, which represents the training examples. In the singular value decomposition, the training set is represented into an example-attribute matrix, and the dimension is reduced through the matrix decomposition. In the principal component analysis, the mean vector and the variance matrix are computed from the training examples, and the dimension is reduced by obtaining the Eigen values and Eigen vectors of the covariance matrix. In the independent component analysis, an example-cluster matrix is constructed, and the dimension is reduced by the pseudo inverse of the matrix. Refer the literature for more detail explanation about the dimension reduction schemes [5].

3.3.4 Encoding Issues

This section is concerned with some issues in encoding a text into a numerical vector. In the previous section, we studied the three steps of encoding: the feature extraction, the feature selection, and the feature value assignment. Some issues

may be considered in encoding. These issues become the motivation for encoding texts into their alternative structured forms to the numerical vectors. This section is intended to point out some issues in encoding texts into numerical vectors.

Let us point out the huge dimensionality as the issue in encoding texts into numerical vectors. According to the previous work, several ten thousand words are extracted from a corpus as the feature candidates [4]. Even if only some are selected among the feature candidates by a feature selection method, the dimension is usually three hundreds. When selecting less than one hundred features, poor performance is caused in the text classification and the text clustering [6]. Although the ten thousand words is cut down to the several hundred features, the dimension is still big.

Let us mention the sparse distribution as the tendency of the numerical vectors that represent texts, as the second issue. The sparse distribution means the dominance of zero values over non-zero values in a numerical vector with more than 90%. Zero values are generated easily by applying the cosine similarity to sparse vectors; the dominance over zero values in each numerical vector results in the fact the discrimination among numerical vectors becomes very poor, while applying a machine learning algorithm to the text classification or the text clustering. Very small coverage of each feature to a corpus is the cause of the sparse distribution of the vectors. It is impossible to avoid both the issues in encoding texts into numerical vectors, at same time; either of the first or the second is faced in doing so.

The fact that numerical vectors that represent texts are non-symbolic representations is one more issue in encoding. In the numerical vectors, their contents are not visible. Each numerical vector whose attributes are words consists of numerical values, which indicates their importance degrees in the text. The huge dimensionality and the sparse distribution in representing texts into numerical vectors blind the text contents, further. There is no way of guessing the text content by a numerical vector, as its representation.

The corpus is required for encoding texts into numerical vectors. The feature candidates are extracted by indexing the corpus, and some are selected among them. In using a different corpus, different features are expected for encoding the texts. Since the corpus may be changed by adding more texts or deleting some texts, we need to consider updating the features, accordingly. The requirement of the corpus and the strong dependency on it become the issues in encoding texts into numerical vectors.

3.4 Image Data

This section is concerned with the image data as another type of raw data. In Sect. 3.4.1, we explore the various image file formats. In Sect. 3.4.2, we describe the matrix that represents an image. In Sect. 3.4.3, we explain the process of encoding an image into a numerical vector. In Sect. 3.4.4, we point out some issues in encoding images into numerical vectors.

3.4.1 *Image File Formats*

This section is concerned with some image file formats, before mentioning the image process. Until now, we studied the preprocessing of the relational data and the textual data for applying the machine learning algorithms, respectively, in Sects. 3.2 and 3.3. An image exists as a file, and several standard file formats are available. Each image file format indicates a scheme of storing and representing a digital image. This section is intended to study the four standard image file formats as the representative ones, before discussing the image processing.

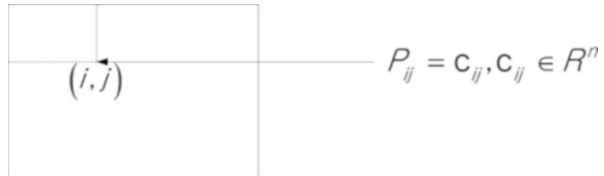
Let us mention the TIFF (Tagged Image File Format) as a popular format of the image file. The gray scaled image and the true color image are within the scope of what this format supports. The linked list of the IFD (Image File Directory) where each IFD links its next one with the IFH (Image File Header) is the data structure of the TIFF. The last tag links the image data in a list of tags in each IFD. The merit of this format is the flexibility in loading images in any application program, but its demerit is that the error may occur frequently in doing so.

Let us mention another file format, GIF (Graphics Interchange Format). The PNG is used as the image file format in the web, together with the GIF, and it tends to be replaced by the PNG, recently. There are the three kinds of images that are supported by this format: the gray scaled image, the true color image, and the indexed image. The advantage of this image file format is the possibility of compressing the file format into a zip file with its high rate and without any loss. Because the PNG has more merits than the GIF, the former is preferred to the latter, while including it in the web.

Let us introduce the JPEG (Joint Photographic Expert Group) not as an image file format, but as a compression method. The proposal of the JPEG is intended to compress the image data with the high compression ratio, 1:16, in average. The images that follow the JPEG are compressed with the three steps: the color conversion, the image conversion, and the compression by the Huffman encoding. The image file formats that are standardized by the JPEG are JFIF (JPEG File Interchange Format), EXIF (Exchangeable Image File Format), and JPEG 2000. Many variants of the JPEG compression algorithm are developed, and multiple algorithms are combined for compressing the image data with each other.

3.4.2 *Image Matrix*

An image is represented into a matrix where each element is given as a pixel. Each pixel indicates a color information or a gray scale. The size of the matrix that represents an image is called resolution, and the image file size depends strongly on the resolution. Only important pixels are used, or the image is compressed into a small sized one, for processing it. This section is intended to study the matrix as an image representation.

Fig. 3.19 Image pixel**Fig. 3.20** Gray scaled image matrix**Fig. 3.21** Color image matrix

In the matrix, an individual cell or element is called pixel, as shown in Fig. 3.19. A pixel in the i th row and the j th column is notated by P_{ij} . There are three value types in the pixel, P_{ij} , depending on the image type: the binary value for the black or white image, the single integer for the gray scaled image, and the multiple bits for the color image. $P_{(i+1)(j+1)}$, $P_{(i+1)(j-1)}$, $P_{(i-1)(j+1)}$, and $P_{(i-1)(j-1)}$ are the neighbor pixels that are adjacent to the pixel P_{ij} and are referred frequently for processing images. P_{ij} with its big differences from its neighbor pixels tends to be recognized as a boundary for segmenting the image.

The pixel value in the gray scaled image is illustrated in Fig. 3.20. In this case, the pixel is given as a continuous value between zero and one. Each pixel indicates a gray degree between the white and the black; the zero indicates the black and the one indicates the white. The gray scaled image is represented as a matrix where each element is given as a continuous value between zero and one. The gray scaled image is viewed as the fuzzification of the white–black image.

A single pixel in the color image, which is given as a vector, is illustrated in Fig. 3.21. It is given as the three dimensional vector that indicates the RGB (Red, Green, and Blue), in the image, which is given as a bmp file. The dimension of the vector is variable, depending on the file format. The cosine similarity or the Euclidean distance is used for computing the difference of a pixel from its neighbors. Because the size of the color image is usually very big, we need to compress the image matrix.

Let us make some remarks on the matrix that represents an image. It is given as a matrix that consists of its pixels. The pixel is a binary value for the black–

white image, a continuous value between zero and one for the gray scaled image, and a RGB combination for the color image. The difference between the pixel and its neighbors becomes the important indicator for extracting the image outline and segmenting the image. It is necessary to select only essential pixels for processing the image more efficiently.

3.4.3 Encoding Process

This section is concerned with the process of encoding an image into a numerical vector. The raw image that is mentioned in Sect. 3.4.2 is given as matrix where its pixel is a binary value for the black–white image, a continuous value for the gray scaled image, and a RGB vector for the color image. The naive scheme of encoding an image is to gather pixels from it, sequentially. Because the naive scheme is characterized as a very wasteful one, we need more efficient encoding methods. This section is intended to describe the process of encoding images into numerical vectors efficiently.

The raw image is given as a matrix that consists of pixels, as mentioned in the previous section. The number of rows and columns of the image matrix is mentioned as the image resolution. A raw image is represented as a numerical vector whose elements are given as pixel values by taking them sequentially. The dimension of the numerical vector that is encoded from the image naively is given as the product of the number of columns and the number of rows. A very huge dimensionality of the vector becomes the issue in encoding images, so it is necessary to find schemes for reducing the dimension for processing image more efficiently.

Let us mention some schemes for reducing the dimension of the numerical vector that represents raw data. The principal component analysis is the dimension reduction scheme that is based on the Eigen values from the covariance matrix of the input vectors. The SVD (Singular Value Decomposition) that is mentioned in Sect. 2.4.3 is based on the Eigen vectors and values of the item–attribute matrix that represents the training examples. The independent component analysis is the scheme of reducing the d dimensional vector into the c dimensional vector where c is the number of clusters, by clustering data items into the $c \times d$ item–cluster matrix that consists of the cluster prototype vectors as its row vectors and multiplying the item–cluster matrix by the original input vector. The three main dimension reduction schemes are described in detail in [5].

We need to compress the image matrix into its smaller size for reducing additionally the dimension of the numerical vector. It is assumed that the original image is given as a $N \times M$ matrix, and it is encoded into a $N \times M$ dimensional vector, naively. The $k \times k$ pixels are merged into a pixel by averaging over their values, and the $N \times M$ matrix is compressed into a $N/k \times M/k$ image matrix. The $N \times M$ dimensional vector is reduced into a $N/k \times M/k$ dimensional vector by $1/(k^2)$. The information loss happens while compressing the image by doing so.

Let us make some remarks on the process of encoding an image into a numerical vector. An image that is given as a matrix, which consists of pixels, looks like more structured data than the textual data. In the naive scheme, the image is encoded into a numerical vector by extracting pixels sequentially. The image compression and the dimensionality reduction are required for encoding images more efficiently. The information loss happens as the payment of the efficiency in the image compression and the dimensionality reduction.

3.4.4 *Encoding Issues*

This section is concerned with some issues in encoding images into numerical vectors. An image is viewed as a matrix whose cell is given as a pixel. Even if an image is encoded into a numerical vector by taking its pixels sequentially, we need to consider some issues in doing so. The naive encoding of the image where all of the pixels are taken is very efficient. This section is intended to point out some issues in encoding images into numerical vectors.

The huge dimensionality is the issue in encoding any type of raw data into numerical vectors. The issue was pointed out in encoding texts into numerical vectors in Sect. 3.3.4. The huge dimensionality is inevitable in encoding image with its high resolution by the naive scheme. If the image with about 1000×1000 is encoded, square of one thousand dimensional vector is required. The dimensionality in encoding an image into a numerical vector is bigger than that in encoding a text into a numerical vector.

The redundant pixel values in the image matrix are mentioned as an issue in processing images. The pixel value is given as a single continuous value for the gray scaled image and as the RGB vector for the color image. Many same pixel values are discovered in the image matrix. The image may be compressed by replacing redundant pixel values by a single pixel value and its frequency. We need to remove the area the redundant pixel values that occur frequently in the image.

The size of the image character has been become an issue in the optical character recognition. Same image characters with their different sizes belong to the same category. The vectors that represent the images are too different ones, so they are classified easily into their different categories. The different pixel values from the images on the same foreground with different sizes are treated as the images with their different foregrounds. We need to standardize the foreground size in the preprocessing.

The rotation of an image is an issue in processing it. The rotated character image may be misclassified easily in the optical character recognition. The image with its rotated foreground tends to be encoded into a very different numerical vector from one that is encoded from the unrotated image. This very different representation of the rotated image becomes the cause of degrading the performance of the image classification and retrieval. The recent proposal of the convolutionary neural networks as a deep learning algorithm is the solution to the problem.

3.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. The relational data is treated most popularly in databases as the traditional type of data. The textual data is the collection of texts each of which consists of sentences that are written in a natural language, and they are encoded into numerical vectors, through the text indexing process. The image data is viewed as a matrix of pixels each of which is given as a binary value, a continuous values, or RGB information. In this chapter, we studied the process of encoding the three types of raw data into numerical vectors as the preprocessing for applying the machine learning algorithms.

It is necessary to consider missing values in processing relational data items. Treating missing values as zero values causes wrong data analysis such as misclassification or wrong clustering. Missing values are estimated by analyzing correlations with other attributes. Relational data item may be analyzed by ignoring attributes with missing values as the alternative scheme. We may apply the machine learning algorithms separately for estimating missing values, viewing it as a regression task.

Let us consider the threads that are attached to news articles as comments, as a kind of text. The thread is characterized as a short text that is written in a colloquial language. There was previously the trial of classifying this kind of texts into one of the three categories: positive, neutral, and negative; this task is called sentimental analysis. Because this kind of text tends to be indexed into a small number of words, there is limit in processing them lexically. The index expansion that uses associated texts is needed for doing the task more robustly.

Let us mention the signal data as one more type of raw data. Even if the signal data is actually given as continuous temporal data, it is represented as a temporal sequence like the time series data by discretizing it. The time interval is decided in advance, and values are gathered in the temporal points with the decided interval. Each signal pattern in the given interval may be codified into a numerical value. The signal data that is given as a temporal sequence is used for evaluating time series prediction model.

The data compression may be needed in processing and analyzing data items. It refers to the process of reducing the data size by encrypting data items into another form. The Huffman coding is the typical data compression scheme where the data is compressed based on character frequencies and it is restored without any loss. Reducing dimensions of the training examples for doing the learning and the generalization more efficiently belongs to the data compression. It is intended to downsize the data file and restore it into its original form with no loss.

References

1. T. Connolly, C. Begg, *Database Systems: A Practical Approach to Design Implementation, and Management* (Addison Wesley, Essex, 2005)
2. T. Jo, Graph based KNN for optimizing index of news articles. *J. Multimed. Inf. Syst.* **3**(3), 53–62 (2016)

3. Y. Yang, An evaluation of statistical approaches to text categorization. *Inf. Retr.* **1**(1), 69–90 (1999)
4. F. Sebastiani, Machine learning in automated text categorization. *ACM Comput. Surv.* **34**(1), 1–47 (2002)
5. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, Berlin, 2018)
6. T. Jo, Normalized table matching algorithm as approach to text categorization. *Soft Comput.* **19**(4), 839–849 (2015)

Chapter 4

Simple Machine Learning Algorithms



4.1 Introduction

We mention some simple machine learning algorithms for understanding the concepts about the machine learning and warming up for studying it. The problem to which we apply the simple machine learning algorithms is assumed to be given as a binary classification where each item is classified into positive or negative. Each item that we classify is given as a two dimensional numerical vector. We present some simple machine learning algorithms that are given hypercubes or hyperspheres as the approaches to the binary classification. This section is intended to describe a toy problem that is the binary classification of two dimensional vectors.

The binary classification is mentioned as a toy problem for explaining machine learning algorithms. It refers to the classification of each item into one of the two categories. Whenever we explain machine learning algorithms, the problem is assumed to be given as a binary classification. The multiple classification may be decomposed into binary classification tasks. In the binary classification, we predefine the two categories: the positive class and the negative class.

Let us mention the collection of two dimensional data for visualizing them easily. Each data item consists of two values, x_1 and x_2 , and is notated by $\mathbf{x} = [x_1 \ x_2]$. Each data item is plotted in the rectangular coordinate system where the horizontal axis corresponds to x_1 and the vertical one does to x_2 . The set of training examples is notated by $\mathbf{Tr} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and a particular training, $\mathbf{x}_i$, is notated by $\mathbf{x}_i = [x_{i1} \ x_{i2}]$. Each training example is labeled with the positive class or the negative class for the binary classification.

Let us mention the most primitive supervised learning algorithm and do other kinds of simple ones in the subsequent sections. Let us assume the training examples as individual scalar values; the positive scalar values are labeled with the positive class and the negative scalar values are labeled with the negative class. The classifier becomes the boundary between the two classes; it is set as the zero point. The boundary is initially set at an arbitrary point, and the learning is to update it to minimize the misclassification of the training examples. Note that noises in the

training examples are negative values that are labeled with the positive class, and vice versa.

The goal of this chapter is to understand the simple machine learning algorithms, given the problem as a single binary classification. We need to understand the expansion of the binary classification into a multiple classification and a regression. We will understand some simple machine learning algorithms that are applicable to the toy binary classifications and are given as a rectangle, a threshold, and a hypersphere. We will understand the linear classifier that is given as a hyperplane in the space, as another kind of simple machine learning algorithms. We will make the further discussions about the simple machine learning algorithms as the pre-study of the advanced machine learning algorithms that are covered in Parts II and III.

4.2 Classification

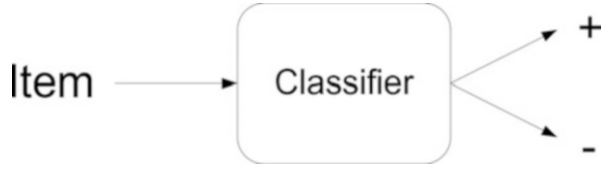
This section is concerned with the classification tasks to which we apply the supervised learning algorithms. In Sects. 4.2.1 and 4.2.2, we mention the two types of classification: the binary classification and the multiple classification. In Sect. 4.2.3, we describe the regression that is expanded from the classification. In Sect. 4.2.4, we describe, in detail, the process of decomposing the given problem, the multiple classification or the regression, into binary classifications. This section is intended to present the process of mapping the given problem into binary classification, and we assume that the given problem is the binary classification, in subsequent sections.

4.2.1 Binary Classification

The binary classification is defined as the simplest classification where each item is classified into one of two categories. Currently, it is assumed that a problem is given as a binary classification for explaining the process of executing machine learning algorithms, easily. The predefined categories are the positive class and the negative class, and the error rate is defined as a cost function. Examples each of which is labeled with one of the two categories are collected as training examples, and the learning is the process of minimizing the cost function on the training examples. This section is intended to describe the binary classification, the cost function, and the learning process, briefly.

The binary classification that is assumed to be a given task for studying simple machine learning algorithms is illustrated in Fig. 4.1. A single item, which is represented into a numerical vector, is given as an input. The item is classified into the positive class or the negative class, by analyzing its relations with the training examples. Each item and its label are notated, respectively, by $x \in \mathbb{R}^d$, which is given as a d dimensional numerical vector, and $y \in \{+, -\}$, which is given as

Fig. 4.1 Functional view of binary classification



one of the two elements. The two labels that are presented as + or −, are called, respectively, the positive class and the negative class.

The cost function is the function for quantifying the loss in classifying an item or estimating an output value. The cost function is needed for observing the loss in making the classification and the regression on the training examples. The cost function is defined in the binary classification as the rate of misclassifying the training examples. The target output and the actual output are notated, respectively, by y_i and $\hat{y}_i$ to the input vector, $\mathbf{x}$, and in the classification, the cost function is expressed by Eq. (4.1):

$$f[y \neq \hat{y}] = \frac{\#(y \neq \hat{y})}{|\mathbf{Tr}|} \quad (4.1)$$

where $\#(y \neq \hat{y})$ is the number of misclassified examples in the training examples, and $|\mathbf{Tr}|$ is the total number of training examples. The direction of learning process is to minimize the cost function value.

Let us mention the outline of learning process based on the cost function. It is set as degree of misclassifying the training examples. The learning process is interpreted into the process of minimizing the cost function in the context of the machine learning algorithm. The parameters of machine learning algorithms are initialized at random and optimized for minimizing the cost function. As they are optimized iteratively, their convergence is reached.

Let us make some remarks on the binary classification. It is one where each item is classified into one of the two categories. It is assumed as the simplest task in studying the machine learning algorithms. It is assumed that each item is given as a two dimensional vector that is able to be plotted in the coordinate system. The hypercube and the hyperplane are assumed to be a classification boundary in the simple machine learning algorithms.

4.2.2 Multiple Classification

This section is concerned with the multiple classification in the functional view. The binary classification is described in Sect. 4.2.2, and it is expanded to the multiple classification in this section. The three categories should be predefined at least in the multiple classification. In the binary classification, we mentioned the two categories: the positive class and the negative class, whereas in the multiple classification,

Fig. 4.2 Functional view of multiple classification



we mention the categories: category 1, category 2, ... category M . This section is intended to describe the multiple classification, functionally.

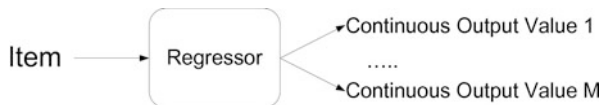
The functional view of the multiple classification is illustrated in Fig. 4.2. A single data item is given as the input, and one or some categories among the M predefined ones are decided as the output. In the hard classification, only one category is generated, and in the soft classification, more than one category is allowed to be generated. In the flat classification, the M categories are predefined as a list, and they are done as a tree, in the hierarchical classification. The decomposition of the multiple classification into binary classification is more desirable in the soft classification.

The cost function is considered in the task of multiple classification. The misclassification rate of the training examples is the cost function like the case of in the binary classification and is expressed in Eq. (4.1). The multiple classification is decomposed into binary classifications, and the cost function may be defined for each binary classification. The risk function is defined as $R(C_i \rightarrow C_j)$ as the risk degree when an item that is labeled with C_i is classified into the category, C_j for all possible pairs of the predefined categories in the real applications. Equation (4.1) is usually defined as the cost function in the multiple classification.

Whether the given task is a binary classification or a multiple classification, the learning process is directed to minimize the misclassifications on the training examples. The cost function is defined as the misclassification rate of the training examples. The parameters of learning algorithms are initialized at random and are updated for minimizing the misclassification rate, continually. When the misclassification rate changes very little, updating the parameters is terminated. In some cases, the risk function that is different among misclassification cases is adopted for learning machine learning algorithms from the training examples.

Let us make some remarks on the multiple classification that is mentioned in this section. In the binary classification, only two categories are predefined, whereas more than two categories are predefined in the multiple classification. Only a single classifier which assigns one among the predefined categories to each item is applicable to the hard classification with a small number of categories. The multiple classification is decomposed into binary classifications as many as categories in the soft classification with a large number of categories. The binary classifier that judges whether an item belongs to its corresponding category, or not, is assigned to each category, in this case.

Fig. 4.3 Functional view of regression



4.2.3 Regression

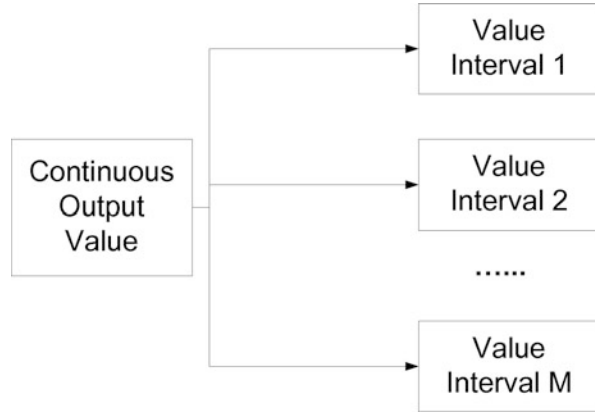
This section is concerned with the expansion of the classification into the more advanced task, called regression. It was already described in Sect. 1.2.2. The regression will be mentioned in the view that is different from one in Sect. 1.2.2; this section covers the learning frame for the task. The target values of the training examples are continuous in the regression, and the error that is the difference between the target values and the computed ones always exists. This section is intended to describe the regression with respect to the cost function and the learning process.

The functional view of the regression is illustrated in Fig. 4.3. The regression was already mentioned in Sect. 1.2.2, functionally. An item is given as a numerical vector, and a continuous value is predicted as its output. The task is modeled as a linear equation, a nonlinear one, neural networks, or a machine learning algorithm. The time series prediction was mentioned as a special regression type, in Sect. 1.2.2.

The cost function is defined in the regression as a loss function. The difference between the target value and the estimated value is the basis for defining the cost function. The square of difference between them that is divided by two is the usual cost function. The division by two is for differentiating the cost function, easily. The learning process is to find the parameters for the zero value of the differentiated cost function.

Let us mention the frame of learning the training examples for defining the regression model. The cost function to the training examples is defined based on the error between the target values and the estimated ones. The parameters are updated by decreasing gradually the differentiation of the cost function. The learning direction is to minimize the cost function value. The possibility of falling into a local minimum, instead of the global minimum, exists in this type of learning.

Let us make some remarks on the regression task that is mentioned in this section. The difference between the classification and the regression is that the output value is discrete in the classification, whereas it is continuous in the regression. In the univariate regression, a single continuous value is given as the output, whereas in the multivariate regression, multiple continuous output values are given as the output. The regression performance has been improved since 1990s by replacing the statistical models by the machine learning algorithms such as neural networks and the SVM [1, 2].

Fig. 4.4 Discretization

4.2.4 Problem Decomposition

This section is concerned with the interpretation of the regression, which was mentioned in Sect. 4.2.3, into binary classification tasks. The binary classification was expanded into the regression from Sects. 4.2.1 to 4.2.3, by mentioning the multiple classification. As the opposite direction, the regression is decomposed into binary classifications, in this section. The regression is mapped into the multiple classification through the discretization, and it is mapped into binary classifications by the decomposition. This section is intended to describe the process of mapping the regression into binary classifications.

The discretization of the entire continuous value range into several intervals is illustrated in Fig. 4.4. The discretization is the process of mapping the regression task where its output is given as a continuous value into the classification task where its output is given as a discrete value. A particular item is classified into one of the intervals, instead of estimating a continuous value. Each value interval is called bin, and the bin size is called bin granularity. In the fine bin granularity, the low prediction error but the high misclassification happens, whereas in the coarse granularity, the situation becomes opposite.

The process of decomposing the multiple classification into binary classifications is illustrated in Fig. 4.5. The process of discretizing continuous output values into discrete value intervals in the left part of Fig. 4.5 was already mentioned above. It is assumed that a binary classifier is assigned to each value interval, the training examples are relabeled with the positive class that indicates the output value corresponds to the interval, or the negative class that indicates not, and each binary classifier is learned with the relabeled examples. In the generalization, a novice vector is given as the input, it is classified into the positive class or the negative class, and the value interval where it is classified into the positive class is the final answer. If it is classified into more than one positive class, the final interval may be decided by the categorical scores.

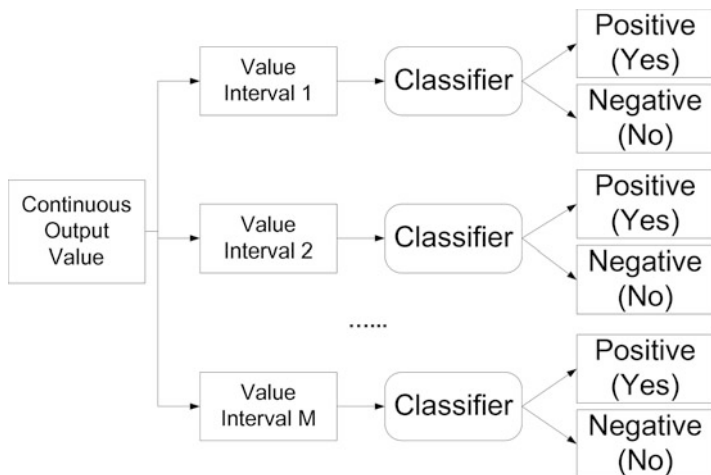


Fig. 4.5 Decomposition into binary classifications

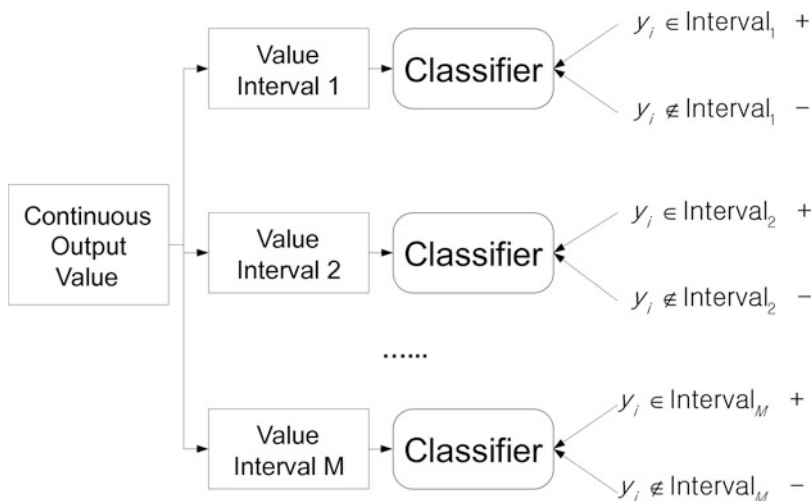


Fig. 4.6 Rearrangement of training examples

The process of relabeling and arranging the training examples is illustrated in Fig. 4.6. The training examples that are prepared for the regression task are initially labeled with their continuous values. For each category that indicates a value interval, the training examples whose output values correspond to the given interval are relabeled with the positive class, and the others are done with the negative class. Only some of the negative examples as many as the positive examples are used for keeping the balance between both classes. Relabeling and rearranging the training examples become the overhead for decomposing the task into binary classifications.

There is the trade-off between the decomposition into binary classification tasks and direct application to the regression or the multiple classification. Much overhead for relabeling and reallocating the training examples happens in decomposing the task into binary classification tasks. Especially, in decomposing the regression into binary classifications, we need decision rules for only a single category, in classifying an item into more than one category. It is useful to decompose the multiple classification into binary classifications in the fuzzy multiple classification where each item is allowed to be classified into more than one category. The direct application to the multiple classification is recommended in the case where a small number of categories is defined and it is crisp.

4.3 Simple Classifiers

This section is concerned with the four simple classification algorithms. In Sect. 4.3.1, we mention the threshold rules that are used for classifying items. In Sect. 4.3.2, we deal with a rectangle as a classifier in the two dimensional space, together with its learning process. In Sects. 4.3.3 and 4.3.4, we mention the hypersphere and the matching algorithm as the simple classification algorithms. This section is intended to study the four simple machine learning algorithms, in order to warm up for studying main machine learning algorithms.

4.3.1 Threshold Rule

This section is concerned with a simple machine learning algorithm that consists of threshold rules. A given problem is assumed to be a binary classification where each item is classified into the positive class or the negative class. The learning process is viewed into the process of searching for optimal threshold rules for minimizing misclassifications. The number of threshold rules becomes the complexity of this type of machine learning algorithm. This section is intended to describe this type of machine learning algorithm for understanding easily the learning process.

The set of most simple training examples is illustrated in Table 4.1. This problem is assumed to be a binary classification, and each training example is labeled with one of the two classes. Each training example is given as a scalar value: one dimensional vector, and labeled with the positive class or the negative class. In Table 4.1, the training examples are given in the first row, and their labels are given in the second row; each column indicates an individual training example. The

Table 4.1 Training examples in one dimensional space

-3	-2	-1	1	2	3
-	-	-	+	+	+

boundary between the positive class and the negative class locates in zero by the intuitive view, but it is assumed to be unknown initially.

In this case, the learning process is regarded as searching for the optimal threshold. The threshold is initialized at random; it is assumed that the threshold is initialized at $+1.5$. The values, -3 , -2 , and $+1$, are classified into the negative class, and the others are classified into the positive class. The value, $+1$, is misclassified into the negative class, so the threshold is updated from 1.5 to 0.5 . After doing that, all of the values are classified correctly after updating so.

Let us consider more complicated ones where we need to define multiple thresholds. We add more training examples, -4 , which is labeled with the positive class, and 4 , which is labeled with the negative class to those in Table 4.1. Accordingly, we add more rules, as follows:

$$\begin{aligned} &\text{if } x \leq -4, \text{ then } + \\ &\text{if } 0 < x < -4, \text{ then } + \\ &\text{otherwise } - \end{aligned}$$

Keeping the simple rule, if $x \geq 0$, then $+$ causes the misclassification of the two examples, -4 and 4 . We may allow the misclassifications for avoiding high complexity from many rules.

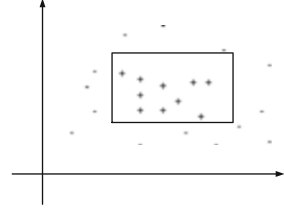
Let us consider the case of the alternative distributions of the positive interval and the negative interval. All thresholds are defined for classifying the training examples, correctly. Classification thresholds may be perfect on the training examples, but very poor on novice examples; this case is called overfitting. We need to reduce classification thresholds, allowing some misclassifications on the training examples for improving the performance on novice examples. There is a trade-off between the number of thresholds as the complexity and the misclassification of the training examples as the error.

4.3.2 Rectangle

This section is concerned with a simple classifier in the two dimensional space, shaped as a rectangle. It is assumed that the area in the rectangle is the positive class area and one out of it is the negative class area. When the training examples that are labeled with the positive class or the negative class are scattered in the two dimensional space, and the learning process is to define the optimal rectangle that minimizes the misclassification rate on the training examples. The rectangle in the two dimensional space is characterized as the two points: the left bottom corner point and the right top corner point. This section is intended to visualize the learning process by the rectangle as a simple classifier.

Figure 4.7 illustrates the training examples that are labeled with the positive class or the negative class and scattered in the two dimensional space. Each example is

Fig. 4.7 Training examples in two dimensional space



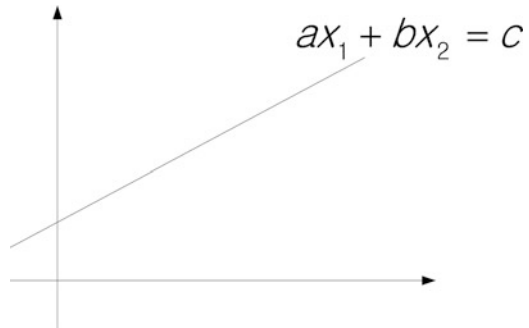
given as a two dimensional vector, and the two axes are assumed as x_1 axis and x_2 axis. The rectangle is defined as the boundary between the positive class and the negative class; the examples inside the rectangle belong to the positive class, and ones out of it belong to the negative class. The rectangle is defined by the left bottom corner, $(\alpha_{\min}, \beta_{\min})$, and the right top corner, $(\alpha_{\max}, \beta_{\max})$, as its parameters. The learning process is executed by manipulating the two corner points for minimizing the misclassifications on the training examples.

The rectangle that is given as the classification boundary is characterized by two threshold rules. The positive class within the rectangle and the negative class out of it are set as the conceptual classification rules. The area of the positive class inside the rectangle is defined as the two rules: $\alpha_{\min} \leq x_1 \leq \alpha_{\max}$ and $\beta_{\min} \leq x_2 \leq \beta_{\max}$. The four values, $\alpha_{\min}$, $\alpha_{\max}$, $\beta_{\min}$, and $\beta_{\max}$, are manipulated for minimizing the misclassifications over the training examples, as the learning process. The increment of $\alpha_{\min}$ and $\beta_{\min}$ and/or the decrement of $\alpha_{\max}$ and $\beta_{\max}$ are for downsizing the rectangle.

Let us explain the learning process of the rectangle that is given as the classifier. Each item is given as a two dimensional vector and classified into the positive class or the negative class. The training examples are scattered in the two dimensional space as shown in Fig. 4.7, and the four parameters, $\alpha_{\min}$, $\alpha_{\max}$, $\beta_{\min}$, and $\beta_{\max}$, are initialized at random. The four values are manipulated for minimizing the misclassification of the training examples, as the learning process. Until no training example is misclassified, they are updated gradually.

Let us make some remarks on the rectangle that is given as a binary classifier in the two dimensional space. The problem is assumed as a toy problem where each item is represented into a two dimensional numerical vector, and classified into the positive class or the negative class. The examples inside the rectangle, which are labeled with the positive class, are called positive examples, and ones outside it, which are labeled with the negative class, are called negative examples. The rectangle is optimized by manipulating the four parameters as the learning process for minimizing the misclassifications. We may consider different amounts of risks, depending on application areas, between the misclassification of positive examples into the negative class and that of negative examples into the positive class.

Fig. 4.8 Linear equation in two dimensional space



4.3.3 Hyperplane

This section is concerned with the hyperplane that is given as a classification boundary. The hyperplane is given as a line in the two dimensional space or a plane in the three dimensional one. It is expressed as a linear combination of products, each of which consists of a variable and a coefficient, and becomes a boundary between the positive class and the negative class. The learning process in the hyperplane is to manipulate the coefficients for minimizing the misclassification or the error. This section is intended to describe the hyperplane classifier, as a simple machine learning algorithm.

A line equation in the two dimensional space is illustrated in Fig. 4.8. The two axes are assumed as x_1 and x_2 . The line in the two dimensional space is expressed as Eq. (4.2):

$$ax_1 + bx_2 = c \quad (4.2)$$

$ax_1 + bx_2 > c$ is the area of the positive class, and $ax_1 + bx_2 < c$ is the area of the negative class in viewing the line as a classification boundary. The learning process is to manipulate the two coefficients, a and b , for minimizing the misclassification.

The line equation in the two dimensional space is expanded into the plane equation in the three dimensional space. The three axes are assumed to be x_1 , x_2 , and x_3 . The plane is the boundary between the positive class and the negative class and is expressed as Eq. (4.3):

$$ax_1 + bx_2 + cx_3 = d \quad (4.3)$$

The areas of the positive class and the negative class are defined, respectively, as Eqs. (4.4) and (4.5):

$$ax_1 + bx_2 + cx_3 > d \quad (4.4)$$

$$ax_1 + bx_2 + cx_3 < d \quad (4.5)$$

a , b , and c in the above equation become the weights of the three variables, and d is given as the bias.

The plane equation in the three dimensional space is expanded into the hyperplane equation in the d dimensional space, further. The axes are assumed as $x_1, x_2, \dots, x_d$. The hyperplane equation is expressed as Eq. (4.6):

$$w_1x_1 + w_2x_2 + \dots + w_dx_d = c \quad (4.6)$$

The positive class area and the negative class area are expressed, respectively, by Eqs. (4.7) and (4.8):

$$w_1x_1 + w_2x_2 + \dots + w_dx_d > c \quad (4.7)$$

$$w_1x_1 + w_2x_2 + \dots + w_dx_d < c \quad (4.8)$$

Equation (4.6) is the general form of the linear classifier equation that is applied to a real problem.

We explained the hyperplane equation that expresses a linear classifier. The problem is assumed as a binary classification where each item is classified into the positive class or the negative class. The vector space is divided into the two areas by the hyperplane. The case where it is possible to separate the training set into the positive group and the negative group by the hyperplane perfectly is called linear separability. In Sect. 4.4, we will study the linear classifier that is viewed as a hyperplane, with respect to the classification.

4.3.4 Matching Algorithm

This section is concerned with one more kind of simple machine learning algorithm, called matching algorithm. It is assumed that the training examples are initially labeled. The matching algorithm is based on the case based reasoning that is the judgment of a particular case by its most similar one among previous cases. What is provided by this section is the basis for studying the supervised machine learning algorithms that are covered in the next part. This section is intended to mention the three matching algorithms.

The classification algorithm by the mean vector matching is illustrated as a pseudo code in Fig. 4.9. The mean vector is computed for each category, by averaging the identically labeled examples. The similarities of the novice vector with the mean vectors are computed. The category of the novice vector whose mean vector is most similar is decided. The mean vector matching becomes the basis for studying the probability learning in Chap. 6.

The process of classifying a data item by the one nearest neighbor is illustrated as a pseudo code in Fig. 4.10. A novice item is given as the input. The training example

Fig. 4.9 Mean vector matching

```

classifyItemByMeanVectors(Item example){
    maxSimilarity = 0;
    for each category in categoryList{
        meanVector = category.getMeanVector();
        similarity = example.getSimilarity(meanVector);
        if(similarity > maxSimilarity){
            maxSimilarity = similarity;
            maxCategoryName = category.getName();
        }
    }
    return maxCategoryName;
}

```

Fig. 4.10 One nearest neighbor

```

classifyItemByOneNearestNeighbor (Item example){
    maxSimilarity = 0;
    for each trainingExample in trainingExampleList{
        similarity = example.getSimilarity(trainingExample);
        if(similarity > maxSimilarity){
            maxSimilarity = similarity;
            label = trainingExample.getLabel();
        }
    }
    return label;
}

```

that is most similar as the novice item is retrieved. The label of the retrieved one becomes the label of the novice one. This algorithm may be expanded into the KNN (K Nearest Neighbor) that is covered in Chap. 5.

The process of classifying an item by the symbolic rule is illustrated in Fig. 4.11. This approach was used for implementing the medical diagnosis system that is called expert system in 1980s. The symbolic rules are manually defined, depending on the prior knowledge in the system, and each item is classified by searching its matching rule. Each rule consists of the conditional part that consists of the attribute and its value, $a_i = v_i$, and the causal part, as the category. The excellent classification performance but the poor flexibility of this approach becomes the motivation for replacing it by the machine learning algorithms.

What is studied in this section provides the basis for doing the supervised learning algorithms that is entirely covered in Part II. The mean vector matching algorithm that was mentioned in the first simple classification algorithm becomes the basis for studying the probabilistic learning that is covered in Chap. 6. The One Nearest Neighbor is the primitive version of the instance based learning that is covered in Chap. 5. The rule based approach becomes for constructing the decision tree from the training examples, which is mentioned in Chap. 7. This kind of the classification algorithm is used as the base for evaluating other classification algorithms.

Fig. 4.11 Rule based approach

```

classifyItemByRules (List ruleList, Item example){
    rule = searchRule(ruleList, example);
    if(rule == null) return "Unclassified";
    return rule.getCategoryName();
}

searchRule(List ruleList, Item example){
    for each rule in ruleList{
        if(rule.isMatch(example)
            return rule;
    }
    return null;
}

isMatch(Rule rule, Item example){
    for each attribute in example{
        value = attribute.getValue();
        if((attribute == rule.getAttribute()) and
            (value == rule.getValue()))
            return true;
    }
    return false;
}

```

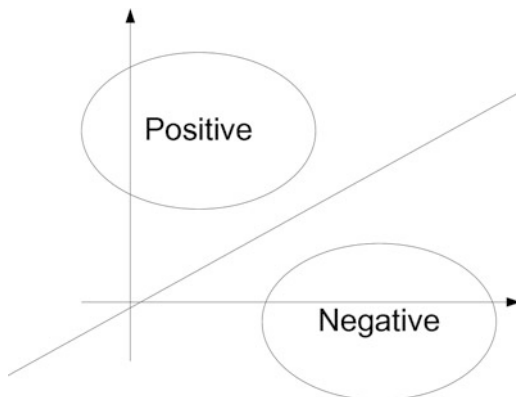
4.4 Linear Classifiers

This section is concerned with the linear classifier that is a kind of simple machine learning algorithms. In Sect. 4.4.1, we explain the linear separability that characterizes the classification by visualizing it in the two dimensional space. In Sect. 4.4.2, we present the hyperplane equation mathematically for studying the linear classifier. In Sect. 4.4.4, we treat the linear classifier as the approach to the binary classification task. In Sect. 4.4.4, we mention the Perceptron as the specific version of linear classifier.

4.4.1 Linear Separability

This section is concerned with the linear separability that characterizes a classification program. The problem is assumed as a binary classification where each item is classified into the positive class or the negative class. Each example is given as a two dimensional vector, the training examples are plotted in the two dimensional space, and the distribution over them is separable linearly; this situation is called

Fig. 4.12 Linearly separability



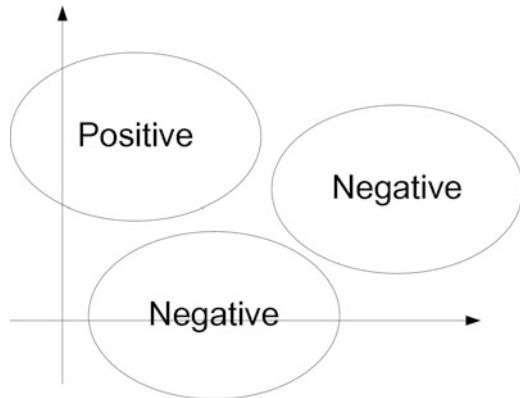
linear separability in the two dimensional space. The linear separability is defined as the situation where examples are separable category by category by a hyperplane that is expressed as a linear combination, in the d dimensional space. This section is intended to study the linear separability for characterizing the classification task, before applying the linear classifier.

The linear separability of the training examples is visualized in the two dimensional space as shown in Fig. 4.12. It is assumed that the given problem is a binary classification and each item is given as a two dimensional vector. The training examples are labeled with the positive class or the negative class and plotted into the two dimensional space as illustrated in Fig. 4.12. They are separated with only a line into the two classes. When the two dimensional space is expanded into the d dimensional space, the training set is separated by the $d - 1$ dimensional hyperplane.

In the linearly separability that is presented in Fig. 4.12, we find the hyperplane that is a classifier without training error. It expresses the area of the positive class that is larger than the hyperplane equation and the area of the negative class that is smaller than the equation. We may express an infinite number of hyperplanes that classify the training examples without any error. Even if the training examples are classified perfectly, there is no guarantee that novice examples are classified correctly. This issue becomes the motivation for making the idea of the SVM (Support Vector Machine) that is covered in Chap. 8.

Let us visualize the nonlinear separability in the two dimensional space as the opposite case to what is illustrated in Fig. 4.12. As shown in Fig. 4.13, it is impossible to separate the training set into the positive class and the negative class by a hyperplane. There is no hyperplane that defines the boundary between the two classes, completely; it is impossible to optimize the hyperplane for no misclassification on the training examples. When applying it for the data classification, the hyperplane should be optimized for minimizing the misclassification on the training examples, rather than eliminating it, completely. In this case, we consider using the multiple layer Perceptron, which defined a quadratic boundary directly, or the SVM, which maps the training examples into ones in another space.

Fig. 4.13 Non-linearly separability



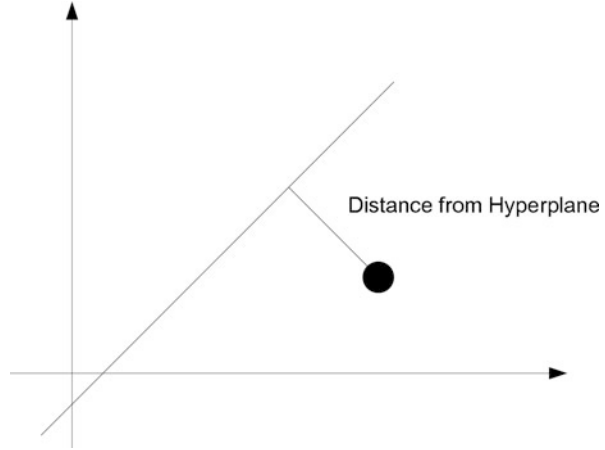
Let us make some remarks on the linear separability that is visualized in Fig. 4.12. In this section, it is assumed that the task is given as a binary classification, and each item is given as a two dimensional vector. In reality, the problem is usually given as a multiple classification or a nonlinear regression, and each item is given as a big dimensional vector. It is impossible to characterize the given problem as whether the training examples are linearly separated by the categories, in advance. In general, somewhat error in the training examples is allowed in training the linear classifier.

4.4.2 Hyperplane Equation

The hyperplane in the d dimensional space is expressed into a linear equation of the d products of the weights and the variables. The hyperplane in the two dimensional space is viewed as a line, and one in the three dimensional space is viewed as a plane. The hyperplane in the d dimensional space is given as a $d - 1$ dimensional space, and the variable values on the hyperplane indicate whether the hyperplane equation is satisfied, or not. The hyperplane in the d dimensional space is used as the classification boundary; the training set is separated by the labels. This section is intended to study the hyperplane equation in the d dimensional space.

Let us explore the equation that expresses the $d - 1$ dimensional hyperplane, in the d dimensional space. The line in the two dimensional space and the plane in the three dimensional space are expressed, respectively, as linear equations, Eqs. (4.2) and (4.3). The hyperplane in the d dimensional space is expressed as Eq. (4.6). In Eq. (4.6), the coefficients, $w_1, w_2, \dots, w_d$, are elements of the vector, $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]$, which is perpendicular to the given hyperplane. $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$ is the input vector that represents a data item, and $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]$ is the weight vector that is optimized through the learning.

Fig. 4.14 Distance from hyperplane



Let us consider the weights that are given as the coefficients of the hyperplane equation in the d dimensional space. The weights, $w_1, w_2, \dots, w_d$ in Eq. (4.6), are expressed as the vector, $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]$. The direction of the weight vector is perpendicular to the hyperplane, which is expressed as Eq. (4.6), with the right angle. The vectors that are parallel to the hyperplane are orthogonal to the weight vector; the inner product of the vectors, $\mathbf{u}$, which is parallel to the hyperplane, and $\mathbf{w}$, which is perpendicular to the hyperplane is zero, as shown in Eq. (4.9):

$$\mathbf{w} \cdot \mathbf{u} = 0 \quad (4.9)$$

The vector of the hyperplane is cross with the weight vector in the right angle.

The distance between the hyperplane and a particular point is illustrated in Fig. 4.14. The above hyperplane equation is simplified into the product of the weight vector and the input vector, as shown in Eq. (4.10):

$$\mathbf{w} \cdot \mathbf{x} - \mathbf{b} = 0 \quad (4.10)$$

and the point and a point on the hyperplane are notated, respectively, by $\mathbf{x}_p$ and $\mathbf{x}_o$. The projection of the vector, $\mathbf{x}_o - \mathbf{x}_p$ between the two points, into the weight vector, $\mathbf{w}$ is expressed as Eq. (4.11):

$$dist = \frac{\mathbf{w} \cdot (\mathbf{x}_o - \mathbf{x}_p)}{\mathbf{w} \cdot \mathbf{w}} \|\mathbf{w}\| = \left\| \frac{\mathbf{w} \cdot \mathbf{x}_o - \mathbf{w} \cdot \mathbf{x}_p}{\mathbf{w}} \right\| \quad (4.11)$$

The norm of the weight vector, $\|\mathbf{w}\|$, is minimized by maximizing the distance from the hyperplane in applying it to the SVM. The distance from the hyperplane follows the weight vector that is perpendicular to the hyperplane.

Let us make some remarks on the hyperplane equation that is covered in this section. The hyperplane is given as a line and a plane, respectively in the

two dimensional space and the three dimensional one. The hyperplane in the d dimensional space, is expressed as a linear combination of products of the variables and the coefficients. The coefficients in the hyperplane equation are called weights, and the weight vector directs perpendicularly to the hyperplane. The distance from the hyperplane is maximized by minimizing the norm of the weight vector as shown in Eq. (4.11).

4.4.3 Linear Classification

This section is concerned with the linear classifier that defines a linear classification boundary. The linear classifier that is expressed as a hyperplane equation is mentioned in Sect. 4.4.2. The linear classifier defines a linear boundary that is given as a hyperplane and separates the d dimensional space into the two areas; it is originally designed for a binary classification. The linear classifier is expanded for using it for the multiple classification or the regression, by decomposing it into binary classification tasks. This section is intended to mention the process of applying the linear classifier to the binary classification, the multiple classification, and the regression.

Let us consider performing the binary classification using the linear classifier. It is expressed into Eq. (4.6), and the input vector is given as a d dimensional vector, $\mathbf{x} = [x_1 \ x_2 \ \dots x_d]$. The input vector, $\mathbf{x}$, is classified into the positive class, if $w_1x_1 + w_2x_2 + \dots + w_dx_d > 0$, and otherwise, it is classified into the negative class. The hyperplane becomes the classification boundary between the two classes. The case where the training examples are classified with the hyperplane with no error, it is called linearly separability.

If the given problem is a multiple classification, it may be decomposed into binary classifications. The given task is to classify each item into one or some among the m categories, and each training example is labeled with one or some among them. The classification task is decomposed into the m binary classifications, and in each binary classification, the positive class indicates that an individual item belongs to the corresponding category, and the negative class indicates that it does not. In each binary classification task, the training examples that are labeled with the corresponding category should be relabeled with the positive class, and the rest should be relabeled with the negative class. In order to keep the balanced distribution over the positive class and the negative class, only some examples that are labeled with the negative class as many as ones with the positive class are used.

It is possible to map the regression into binary classification tasks. By discretizing the continuous output value, the regression is mapped into a multiple classification. It is decomposed into binary classification tasks by the process that is mentioned above. The linear classifier is allocated for each binary classification. The regression is solved by the multiple linear classifiers.

Let us make some remarks on the application of a linear classifier to the binary classification, the multiple classification, and the regression. The linear classifier is

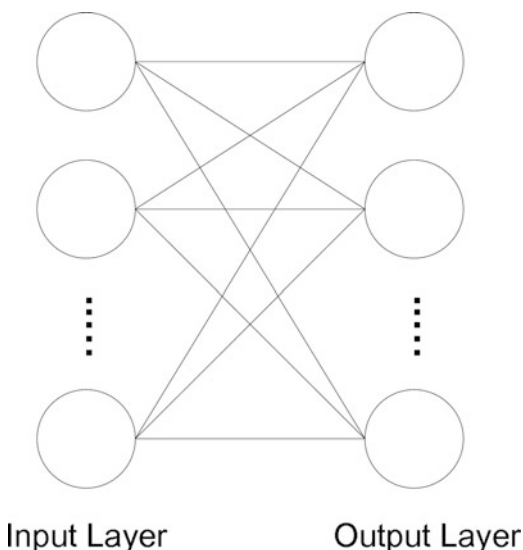
expressed as a linear combination of the inner product of the input vector and the weight vector. The linear classifier that represents a hyperplane in the d dimensional space is the dichotomy that separates the space into the two areas; this classifier kind is designed for performing the binary classification that is characterized as the linear separability. This classifier kind may be applied to the multiple classification and the regression through the decomposition that are mentioned above. The linear classifier is expanded into the SVM (Support Vector Machine), which is covered in Chap. 8, through the simple neural networks, Perceptron.

4.4.4 Perceptron

. This section is concerned with the Perceptron, which is a typical instance of the linear classifier. It was invented in 1950s as the initial neural networks, which is able to learn the training examples. The given problem is assumed to be a linear separable one, and the optimal hyperplane that separates the training examples without any error by its learning performance. Because the nonlinear separability is more usual than the linear separability, the Perceptron was expanded into the MLP (Multiple Layer Perceptron) and the SVM (Support Vector Machine). This section is intended to describe the Perceptron with respect to its architecture and its learning process.

The architecture of the neural networks, Perceptron, is illustrated in Fig. 4.15. Its architecture consists of the two layers: the input layer and the output layer. The values of the input nodes are notated by $x_1, x_2, \dots$, and ones of the output values are notated by $y_1, y_2, \dots$. The weight between the two layers is notated by w_{ij} , where j is an input node index, and i is an output node index. The reason that the

Fig. 4.15 Architecture of Perceptron



input node index follows the output node index in notating the weight is to optimize the weight in the direction from the output node to the input node.

Let us mention the process of classifying the input vector into the positive class or the negative class by the Perceptron. The input vector is given as a d dimensional vector, $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$, and the number of input nodes becomes d . It is assumed that there is a single output node, $\hat{y}$, in the output layer, in the binary classification. The weights are given as a d dimensional vector, $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]$, and the net input of the output node is computed by the inner product of the input vector and the weight vector with the addition of the bias, $\mathbf{b}$ as shown in Eq. (4.12):

$$netinput = \mathbf{x} \cdot \mathbf{w} + b \quad (4.12)$$

The value of y that is -1 or 1 is set by the threshold function that is presented in Eq. (4.13):

$$sim(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} -1 & \text{if } netinput \leq 0 \\ 1 & \text{otherwise} \end{cases} \quad (4.13)$$

The learning of the Perceptron is to update gradually its weight vectors for minimizing the misclassifications on the training examples. The weight vector is initialized with its random values around zero, and the training examples are classified by Eq. (4.14):

$$\hat{y} = F(\mathbf{w} \cdot \mathbf{x} + b) \quad (4.14)$$

where $F(\cdot)$ is a threshold function in Eq. (4.13). When a training example is classified correctly, the weights are not updated, and otherwise, the weights are updated by Eq. (4.15):

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(y - \hat{y})\mathbf{x}_i \quad (4.15)$$

When all of the training examples are classified correctly, the update of the weights is terminated. The learning process by updating the weight vector is to search for the hyperplane that separates the training examples by their labels.

The Perceptron is applicable to the regression task as well as the classification one. The regression is the process of estimating an output value as a continuous value by the input vector. The estimated output, $\hat{y}$, and the desired output, y , are assumed as continuous values. Equation (4.15) is applied continually for updating weights. The Perceptron that is applied to the regression is trained to minimize the difference between the desired output and the estimated one.

4.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We mentioned the classification and the regressions that are the tasks to which we apply the supervised machine learning algorithms. We mentioned some simple classifiers such as a rectangle, a hyperplane, and a matching algorithm, and assumed the given problem as a binary classification. We studied the hyperplane that separates the group of data items into the positive class group and the negative class group. In this chapter, we studied the simple learning algorithms for warming up for studying the subsequent chapters.

A given problem is assumed as a binary classification in discussing simple machine learning algorithms in this chapter. We consider the clustering as the task that is different from the classification. The AHC (Agglomerative Hierarchical Clustering) algorithm, the divisive clustering algorithm, and the online linear clustering algorithm are mentioned as the simple clustering algorithms. This chapter is intended to understand the fundamental machine learning algorithms by mentioning the only simple classification problem. We will cover clustering algorithms that belong to the unsupervised learning algorithm from Chap. 9 to 12.

Let us mention the underfitting as the issue in learning the training examples. The underfitting refers to the phenomena where the training examples are classified or regressed very poorly by the insufficient learning. The underfitting is caused by updating weights insufficiently as the learning process of neural networks or trying to solve the nonlinear problem by an only linear classifier. The underfitting is not applicable to the KNN algorithm, which is covered in Chap. 5, and the Naive Bayes, which is covered in Sect. 6.3. The solution to the underfitting is more additional update of weights in training the neural networks, or addition of more complexity in training other machine learning algorithms.

Let us mention the overfitting as another issue in training the machine learning algorithms. It refers to the phenomena where good classifications or regressions happen on the training examples, but poor ones happen on novice ones. If machine learning algorithms are trained with an insufficient number of training examples, overfitting happens frequently. Designing the neural networks with their very complicated architecture, learning algorithm and updating weights in too many iterations in using the neural networks are causes of overfitting. The solutions to the problem are to add more training examples, to downsize the complexity, and to observe the validation error using the validation set that is separated from the training examples.

The principle of Occam's Razor is to select simplest one among approaches with their same or similar performance. In this principle, the complexity is considered in comparing approaches with each other, as well as the performance. There are some cases of training machine learning algorithms based on the training error and the complexity. When training neural networks, both terms, the training error term and the complexity term, are reflected in the equation of updating weights. Training machine learning algorithms with too high complexity is the cause of the overfitting.

This part is concerned with the supervised machine learning algorithms as the main part. In this part, we mention the instance based machine learning algorithms where the label of a novice example is decided by referring those of training examples, as a popular kind of machine learning algorithm. The probability based machine learning algorithms belong to another popular kind where the posterior probability of example given a category is computed as the categorical score. In this part, the decision tree, which is characterized as the very symbolic machine learning algorithms, and the support vector machine, which is characterized as the kernel based learning, are included. This part is intended to describe the four main kinds of supervised machine learning algorithms as the approaches to the classification and the regression.

This part consists of the four chapters. In Chap. 5, we describe the instance based machine learning algorithms such as the KNN and its variants. In Chap. 6, we mention the three probabilistic machine learning algorithms: Bayes Classifier, Naive Bayes, and Bayesian Networks. Chapter 7 covers the decision tree and the random forests. In Chap. 8, we describe the support vector machine as the kernel based learning.

References

1. T. Master, *Neural, Novel and Hybrid Algorithms for Time Series Prediction* (Wiley, Hoboken, 1995)
2. K. Kim, Financial time series forecasting using support vector machines. *Neurocomputing* **55**(1–2), 307–319 (2003)

Part II

Supervised Learning

Chapter 5

Instance Based Learning



5.1 Introduction

The instance based learning is defined as the type of supervised learning algorithms that solves a problem depending on individual training examples. In advance, we prepare the training examples that are labeled with one of the predefined categories. When an example that we try to classify is given, its label is decided by voting ones of some training examples. Because training examples are not learned in advance, this type of supervised learning is called lazy learning. This section is intended to study some concepts that are necessary for understanding the instance based learning.

Let us consider the naive version of instance based learning. A training set that consists of previous examples is initially given. In the naive version, we make the classification or the regression toward the particular item by looking it up from the training set. When the novice example is different from all training examples, it is rejected instead of providing its answer. In reality, it is not feasible to use the naive version for the classification or the regression.

The similarity metric between two items is the essential operation in the instance based learning. There is no guarantee that a novice example is the same at least one among the training examples, and what is different from all of the training examples is rejected in the naive version. The inner product of two examples that are given as numerical vectors indicates the similarity between them. The cosine similarity is used for normalizing the similarity between zero and one. The variants from the cosine similarity are available as normalized similarity metrics.

Once a similarity matrix is defined, it is possible to select similar examples as the nearest neighbors. It is assumed that the training examples are initially labeled with their own categories. When a novice example is given as the input, its similarities with the labeled training examples are computed. The training examples are ranked by their similarities, and most similar labeled ones are selected as nearest neighbors. They become the important references for deciding the label of a novice example.

The goal of this chapter is to understand the instance based learning, including the KNN algorithm. We need to understand some Naive machine learning algorithms, before studying the KNN algorithm. Afterward, we will understand the KNN with respect to its classification process. We will also understand some variants that are derived from the KNN algorithm as more advanced ones. We will make further discussions on the KNN algorithm for deriving more advanced instance based learning algorithms.

5.2 Primitive Instance Based Learning

This section is concerned with the naive versions of the instance based learning. We mention the simple look-up from the training examples and application of symbolic rules for classifying novice items in Sects. 5.2.1 and 5.2.2. In Sect. 5.2.3, we present the similarity metrics of two examples that are necessary for implementing the instance based learning algorithms. In Sect. 5.2.4, we describe the One Nearest Neighbor as the initial machine learning algorithm with its own adaptability. This section is intended to present the simple versions of instance based learning algorithms.

5.2.1 Look-Up Example

The look-up example is defined as the process of finding the matching one to the novice example from the training examples, in order to classify it. It is assumed that a set of labeled training examples is given as the table that is called look-up table. When an unlabeled novice example is given, one matching with it is retrieved from the look-up table. The label of the matching one from the look-up table becomes that of the novice example. This chapter is intended to describe and demonstrate the look-up example that is the most primitive machine learning algorithm.

The look-up table that consists of the training examples is illustrated in Fig. 5.1. A given problem is assumed as a binary classification, so each example is labeled with the positive class or the negative class. The look-up table is viewed as a set of the labeled training examples, $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$, $\mathbf{x} \in \mathfrak{R}^d$, $y \in \{+, -\}$ where $\mathbf{x}_i$ is a training example that is represented as a d dimensional numerical vector, and y_i is the positive class or the negative class. When the problem is given as a multiple classification, y_i is a discrete value, and when the problem is given as a regression, y_i is given as a continuous value. When studying the machine learning algorithms in the subsequent chapters, the look-up table is called the training set, $\mathbf{Tr}$.

The process of classifying an item by the look-up table is illustrated in Fig. 5.2. It is assumed that the training examples are given as a hash table where the key becomes an input vector and its associative becomes its category. A novice example

Fig. 5.1 Look-up table

Input Vectors	Label
Input Vector 1	+
Input Vector 2	-
.....	
Input Vector N	+

```
Hashtable lookupTable; //Lookup Table given as a Hash Table
Category classifyExampleByLoopupTable(Item example){
    Category cc = lookupTable.findValue(example);
    return cc;
}
```

Fig. 5.2 Classification process in pseudo code

is given as an argument, and the category is found with it as a key from the hash table. The novice example should match exactly with one among the training examples for classifying it. This algorithm has no adaptability; if matching item is not available in the training set, it is rejected.

The look-up table that consists of the training examples is presented in Fig. 5.3. Each item is given as a one dimensional vector and labeled with the positive class or the negative class. The item [10] is given as the input, and its exactly matching one is retrieved from the look-up table. The training example, [10], matches with the novice item and labeled with the positive class, so the novice item is classified into the positive class. If the exactly matching item is not available, it is not classified.

Let us make some remarks on the look-up example as a naive machine learning algorithm. The look-up table is the collection of previous examples for deciding the label or the output value to a novice example that matches with one of them. The lack of the adaptability is the reason of not regarding it as a machine learning algorithm. The naive algorithm is the primitive style of the classification algorithm or the regression algorithm, which are similar as the rule based system. Even if it is not practical, studying the look-up table is the important reference for doing the machine learning algorithms.

5.2.2 Rule Based Approach

This section is concerned with the rule based approach as the classification method. The rule based approach is applied for classifying data items or estimating a continuous output value or output values, before doing the machine learning algorithms. The rules each of which consists of the conditional part and the causal

Fig. 5.3 Example of look-up table

3	4	+
10	5	+
3	5	+
8	5	+
10	1	+
3	10	+
10	5	+
5	5	+
6	3	+
4	2	+
-3	-4	-
-4	-8	-
-9	-3	-
-7	-6	-
-7	-10	-
-5	-8	-
-5	-10	-
-2	-6	-
-7	-1	-
-9	-3	-

part in the if-then form are defined manually. The rule based approach results in its good accuracy but its poor flexibility; only examples that are applicable to the rules are able to be classified. This section is intended to describe the rule based approaches to the classification and the regression, briefly, as the alternative kind to the machine learning algorithms.

The symbolic rules for classifying items are illustrated in Fig. 5.4. Each classification rule consists of the two parts, the conditional part, if $A_i = v_{ij}$, and the causal part, then C_r . The conditional part, if $A_i = v_{ij}$, means “if attribute A_i is equals to the value, v_{ij} ,” and the causal part, then C_r means “the item is classified into the category C_r .” The classification rule may be expanded into a composite rule, if $(A_i = v_{ij}) \wedge (A_m = v_{mj})$, then C_r , where more than one condition is included in the classification rule. The two rules, if $A_i = v_{ij}$, then C_r and if $A_i = v_{ij}$, then C_s , with $C_r \neq C_s$, are called contradiction rules.

Fig. 5.4 Classification rules

if $A_1 = v_1$, then C_1
 if $A_2 = v_2$, then C_2

 if $A_k = v_k$, then C_k

Fig. 5.5 Classification process in pseudo code

```

classifyByRuleList(List ruleList, Item example){
    for each value in example{
        for each rule in ruleList{
            if (rule.isMatch(value))
                return rule.getCategory();
        }
    }
    return 'unclassified';
}

```

Fig. 5.6 Example of rule based approach

if $x_1 \geq 0$, then positive
 if $x_2 \geq 0$, then positive
 if $x_1 < 0$, then negative
 if $x_2 < 0$, then negative

The process of classifying an item by symbolic rules as the pseudo code in Fig. 5.5. A list of symbolic rules and a particular item are given as the arguments. The system searches for a symbolic rule that matches with the given item. If a matching rule is available, the category is taken from it and returned as the output, and otherwise, the message, “unclassified,” is returned. The higher priority is put to the previous rule among matching rules, when more than one rule is matching.

The simple classification rules are presented in Fig. 5.6. It is assumed that an item is given as a two dimensional vector, $[x_1, x_2]$. The item, $[2, 3]$, is classified into the positive class by the classification rule, $x_1 \geq 0$. $[-3, -2]$ is classified into the negative class by the classification rule, $x_1 < 0$. The item $[-12]$ is classified into the positive class by the classification rule, $x_{21} \geq 0$, before applying the classification rule, $x_1 < 0$.

Let us make some remarks on the rule based approach for the classification or the regression. In defining symbolic rules, much prior knowledge is required about the domain. The rule based approach was adopted in implementing the expert system in 1990s. Because the flexibility and the adaptiveness are lack in the rule based system, it is replaced by the machine learning algorithm. The symbolic rules may be used as

the secondary part of the classification system or the regression system for providing bias toward the correct classification or estimation.

5.2.3 Example Similarity

This section is concerned with the similarity between two numerical vectors that are necessary for executing the instance based learning. A novice example is not always matching exactly with any training example. We need to retrieve approximately matching item from the training examples for deciding the label of novice item. It is necessary to define a similarity metric between a novice item and a training example for taking the most similar one from the training set. This section is intended to describe two similarity metrics: Euclidean distance and cosine similarity.

Let us describe briefly the process of encoding a raw data item into a numerical vector. Feature candidates are extracted and some of them are selected as actual features. To each feature, its own value is assigned. A numerical vector where the selected features are attributes is generated. The similarity between raw data items is assumed to one between numerical vectors that are encoded from them.

Let us mention the process of computing the Euclidean distance between two vectors, as a similarity metric. It is assumed the two d dimensional vectors are initially given as $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$ and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$. The Euclidean distance between two vectors is expressed in the general form as Eq. (5.1)

$$dis(\mathbf{x}_1, \mathbf{x}_2) = \left(\sum_{i=1}^d (x_{1i} - x_{2i})^k \right)^{\frac{1}{k}} \quad (5.1)$$

In Eq. (5.1), k is usually set as two, so the Euclidean distance is expressed as Eq. (5.2)

$$dis(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^d (x_{1i} - x_{2i})^2} \quad (5.2)$$

The similarity between two vectors is expressed as the reverse of the Euclidean distance into Eq. (5.3)

$$sim(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{dis(\mathbf{x}_1, \mathbf{x}_2)} \quad (5.3)$$

Let us mention another similarity metric between two examples, called cosine similarity. The inner product between two vectors is normalized under the assumption that the inner product indicates the similarity between two vectors. The cosine similarity between two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, is expressed into Eq. (5.4)

$$\text{sim}(\mathbf{x}_1, \mathbf{x}_2) = \frac{\|\mathbf{x}_1 \cdot \mathbf{x}_2\|}{\|\mathbf{x}_1\| \cdot \|\mathbf{x}_2\|} \quad (5.4)$$

Because $\|\mathbf{x}_1\| \cdot \|\mathbf{x}_2\| \geq \|\mathbf{x}_1 \cdot \mathbf{x}_2\|$, which was mentioned in Sect. 2.2.3, the cosine similarity is always given as a normalized value between zero and one. Equation (5.5) may be mentioned as a cosine similarity variant.

$$\text{sim}(\mathbf{x}_1, \mathbf{x}_2) = \frac{2 \cdot \|\mathbf{x}_1 \cdot \mathbf{x}_2\|}{\|\mathbf{x}_1\| + \|\mathbf{x}_2\|} \quad (5.5)$$

Let us make some remarks on the process of computing the similarity between two items. We need to encode raw data items into numerical vectors, in order to compute the similarity. In encoding so, the information loss may happen; the similarity between the encoded data items is not equal to one between the raw data items. The normalized Euclidean distance is reversed by subtracting it from 1.0. Multiple similarity metrics are computed, and they are averaged into the final similarity in the advanced scheme of computing the similarity.

5.2.4 One Nearest Neighbor

This section is concerned with the 1-NN (One Nearest Neighbor) as the simplest version of the KNN that is entirely covered in Sect. 5.3. We previously mentioned the pure Naive approach where only a matching example is retrieved to the novice one, and it lacks the adaptiveness for classifying items. The 1-NN where a training example that is most similar as the novice one retrieved is the initial classification algorithm with the adaptiveness. The similarity metric, which is mentioned in Sect. 5.2.3, is used for finding the most similar example. This section is intended to describe the 1-NN as the preparation for studying the KNN.

We need to gather examples that are labeled manually as the training examples. The categories are predefined as a list or a hierarchical structure, and the hard classification or the soft classification is considered in the given application area. It is assumed that the given problem is the hard classification, and examples that are labeled with one among the predefined categories are collected. In the case of the soft classification, examples that are labeled with more than one category may be collected. Each example is labeled with one of the categories in the lowest level in the case of the hierarchical classification.

The 1-NN does not learn the training examples in advance, after preparing them. When a novice example is given as the classification target, it starts to learn them. It is assumed that the training examples and the novice one are given as numerical vectors, and the similarities of the novice example with the training ones are computed by Eq. (5.3) or (5.4). The training example with its maximum similarity is selected, and the novice example is classified into the label of the selected. The

ability to process the novice example when its exactly matching one is not available is the reason that the 1-NN is more flexible than the look-up example.

The 1-NN is expanded into the KNN where more than one training example is selected as nearest neighbors. The similarities of the novice example with the training examples are computed. The k most similar training examples are selected as the nearest neighbors. The label of the novice example is decided by voting ones of the nearest neighbors. The KNN will be studied in detail in Sect. 5.3.

Let us make some remarks on the 1-NN that is covered in this section. The 1-NN is viewed as the initial machine learning algorithm with its own adaptability. The scheme of computing a similarity between data items is very important part of this algorithm. The 1-NN where the most similar example is taken may be expanded into the KNN where more than one nearest neighbor is taken. In Sect. 5.3, we will study the KNN including its variant in detail.

5.3 Classification Process

This section is concerned with the process of classifying data items by the KNN. In Sect. 5.3.1, we define the mathematical notations that are involved in explaining the KNNs. In Sects. 5.3.2 and 5.3.3, we explain the two main steps of classifying items by the KNN, and selection and voting of nearest neighbors. In Sect. 5.3.4, we mention the discriminations among attributes in computing the similarities of notice item with the training examples for selecting nearest neighbors. In this section, we focus on the standard version of the KNN.

5.3.1 Notations

This section is concerned with the notations that are involved in studying the machine learning algorithms. The notations that are mentioned in this section will be used in the subsequent chapters, continually. An example or a data item is expressed as d dimensional vector, and a collection of these examples is given as set of vectors. The similarity between two data items is the essential value of executing the machine learning algorithms and is given as a scalar value. This section is intended to mention the mathematical notations for describing more clearly the machine learning algorithms.

Each training example or each novice example is given as a d dimensional vector. The vector, $\mathbf{x}_i$, is expressed by $\mathbf{x}_i = [x_{i1} \ x_{i2} \ \dots \ x_{id}]$, where d elements exist. $\mathbf{x}$, which is given in bold, indicates a vector, and x , which is given in italic, indicates a value. $\mathbf{x}_i$, which has its subscript, indicates a training example, and $\mathbf{x}$ without its superscript indicates a novice example, in modeling machine learning algorithms under the assumption of a set of training examples and a novice example. A particular matrix is notated by an upper character in bold, $\mathbf{X}$.

The set of examples that is prepared for training the machine learning algorithms consists of numerical vectors. For simplicity, the set of examples is divided into the two subsets: the training set for executing the learning process, and the test set for evaluating the performance. The training set is notated by $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and the test set is notated by $Te = \{\mathbf{x}_{N+1}, \mathbf{x}_{N+2}, \dots, \mathbf{x}_P\}$. The validation set is used for tuning the parameters during the learning process, derived from the training set as its subset, and notated by $Va = \{\mathbf{x}_{v1}, \mathbf{x}_{v2}, \dots, \mathbf{x}_{vk}\}$. Among the training examples, $N - k$ examples used for the learning process and k examples are for tuning parameters.

We mentioned the similarity metrics between data items that are given as numerical vectors, in Sect. 5.2.3. The cosine similarity and the inverse Euclidean distance are typical similarity metrics between items. The similarity between two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, is notated by $sim(\mathbf{x}_1, \mathbf{x}_2)$. Here, we adopt the cosine similarity as the similarity metric and assume the similarity between two vectors to be given as a normalized value between zero and one, as shown in Eq. (5.6),

$$0 \leq sim(\mathbf{x}_1, \mathbf{x}_2) \leq 1 \quad (5.6)$$

The similarity threshold, ρ , is set between zero and one because the similarity is a normalized value.

The training examples that are selected as most similar ones as a novice example are called its nearest neighbors. They play the role of references for deciding the label of the novice example, $\mathbf{x}$. The ordered set of nearest neighbors is notated by $Near(k, \mathbf{x}) = \langle (\mathbf{x}_1^N, y_1^N), (\mathbf{x}_2^N, y_2^N), \dots, (\mathbf{x}_k^N, y_k^N) \rangle$. The first element in the ordered set is the nearest neighbor, and the last one is the k th nearest one; the discriminations among them should be put for deciding the label in a KNN variant. Nearest neighbors may be selected differently, depending on the similarity metric.

5.3.2 Nearest Neighbors

This section is concerned with the process of retrieving nearest neighbors in using the KNN algorithm. It is assumed that the problem is given as a binary classification, and all of the training examples are labeled with the positive class or the negative class. The similarities of a novice example with the training examples are computed, and the k most similar training examples are selected as its nearest neighbors. The label of the novice example is decided by referring the labels of the nearest neighbors. This section is intended to explain the process of selecting nearest neighbors from the training examples.

The similarity metric that is used for executing the KNN algorithm is illustrated in Fig. 5.7. $\mathbf{x}$ and $\mathbf{y}$ are given as d dimensional vectors. The Euclidean distance and the cosine similarity are computed by equations that are presented in Fig. 5.7. We need to reverse and normalize the Euclidean distance, but the cosine similarity is initially given as a normalized value. The cosine similarity that considers the feature

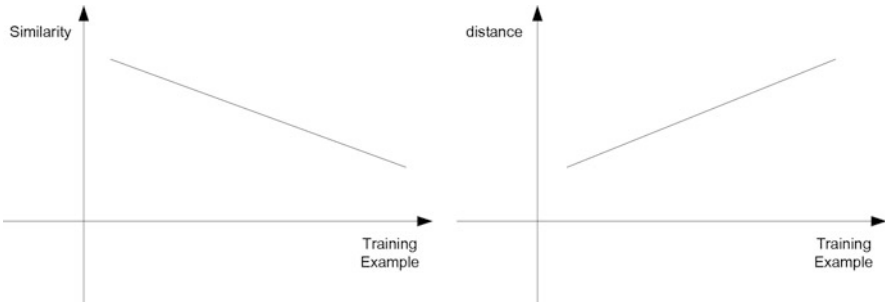
Fig. 5.7 Similarity computation

$$x = [x_1 \ x_2 \ \dots \ x_d]$$

$$y = [y_1 \ y_2 \ \dots \ y_d]$$

$$\text{Euclidean Distance} = \sqrt{\sum_i^d (x_i - y_i)^2}$$

$$\text{Cosine Similarity} = \frac{2 \sum_i^d x_i y_i}{\sum_i^d x_i^2 + \sum_i^d y_i^2} = \frac{2x \cdot y}{\|x\| + \|y\|}$$

**Fig. 5.8** Ranking of training examples

similarity as well as the feature value similarity was proposed recently in order to solve the fragility to sparse vectors.

The training examples are ranked by the ascending order of their distances or the descending order of their similarities, after computing the similarities or the distances, as shown in Fig. 5.8. When the cosine similarity is adopted as the similarity metric, the training examples are sorted by their descending order. When the Euclidean distance is adopted, they are sorted by their ascending order. Some with the maximum similarities or the minimum distances are selected as the nearest neighbors. The label of the novice item is decided by referring ones of the nearest neighbors.

The process of selecting the nearest neighbors among the training examples is illustrated in Fig. 5.9. The training examples are ranked by their similarities or distances. The k most similar ones or k least distant ones are taken as the nearest neighbors. The label of the novice example is decided by voting ones of the nearest neighbors. The continuous output value of a novice example is decided by averaging ones of the nearest neighbors.

Let us make some remarks on the nearest neighbors that are selected for deciding the label of a novice example. The numerical vectors that represent the examples whose elements are normalized are assumed. It is required to define the similarity

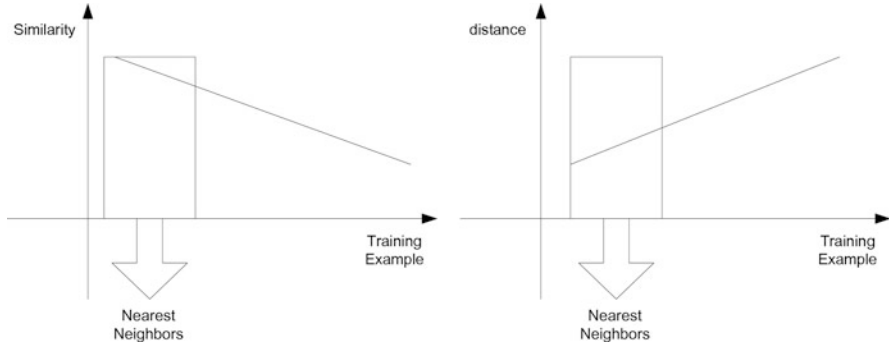


Fig. 5.9 Selection of nearest neighbors

metric between items that are given as numerical vectors for selecting the nearest neighbors. We may consider the discriminations among the nearest neighbors, depending on their similarities with a novice example in voting their labels. We may also consider a variable number of nearest neighbors, depending on the distribution over them around a novice example.

5.3.3 Voting

This section is concerned with the process of deciding the label of a novice example in the KNN algorithm. A novice example is given as the input, and its similarities with the training examples are computed. Most similar training examples are selected as the nearest neighbors, and the label of the novice example is decided by referring ones of them. The process of deciding the label of the novice example by collecting the labels of the nearest neighbors is called voting. This section is intended to describe the schemes of voting labels of the nearest neighbors.

The voting is the scheme of deciding the label of the novice example by referring ones of the nearest neighbor while classifying it by the KNN algorithm. The nearest neighbors of the novice example, $Near(k) = \langle (\mathbf{x}_1^N, y_1^N), (\mathbf{x}_2^N, y_2^N), \dots, (\mathbf{x}_k^N, y_k^N) \rangle$, are selected by the process that was described in Sect. 5.3.2. The set of the nearest neighbors is divided into the two sets: the positive set, $Near_+(k) = \langle (\mathbf{x}_{1+}^N, y_{1+}^N), (\mathbf{x}_{2+}^N, y_{2+}^N), \dots, (\mathbf{x}_{k+}^N, y_{k+}^N) \rangle$, and the negative set, $Near_-(k) = \langle (\mathbf{x}_{1-}^N, y_{1-}^N), (\mathbf{x}_{2-}^N, y_{2-}^N), \dots, (\mathbf{x}_{k-}^N, y_{k-}^N) \rangle$, assuming the given problem as a binary classification. The label of a novice example is decided by the majority of the labels of the nearest neighbors as expressed in Eq. (5.7)

$$c_{\max} = \underset{+, -}{\operatorname{argmax}}(k_+, k_-) \quad (5.7)$$

where k_+ is the number of nearest neighbors that belong to the positive class and k_- is the number of nearest neighbors that belong to the negative class. The odd number of nearest neighbors is usually selected for avoiding the same portions of the two categories.

Let us mention another scheme of voting the labels of the nearest neighbors for deciding the label of the novice example, called weighted voting. The K nearest neighbors are selected from the training examples by the similarity or the distance of the novice example and influence differently on decoding the label. The weights, $w_1 > w_2 > \dots > w_k$, are assigned to the k nearest neighbors, which are sorted by their similarities with the novice example, and the voting influences on the classes are computed by summing the weights, as Eqs. (5.8) and (5.9),

$$V_+ = \sum_{i \in \text{Near}_+(k)} w_i \quad (5.8)$$

$$V_- = \sum_{i \in \text{Near}_-(k)} w_i \quad (5.9)$$

The label of the novice example is decided by the maximum voting influence, as expressed in Eq. (5.10),

$$c_{\max} = \underset{i}{\operatorname{argmax}} V_i \quad (5.10)$$

The simple voting, which is explain above, is viewed case of assigning constant weights to the nearest neighbors.

The weights may be assigned to the nearest neighbors, exponentially. In the above weighting scheme, the weights are assigned to them, arbitrary, keeping the rule, $w_1 > w_2 > \dots > w_k$. A normalized value, $0 < r < 1$, is set, and the relation among the weights is expressed as Eq. (5.11)

$$w_i = wr^{i-1} \quad (5.11)$$

Let us consider the two extreme cases in setting the value, r : the KNN in r that is close to 1.0, and the 1 Nearest Neighbor in r that is closely to 0.0. The number of the nearest neighbors is usually set as a large number in using this voting scheme.

We explained the process of voting the labels of the nearest neighbors and consider the case of using the KNN for a regression task. The nearest neighbors are labeled with their continuous output values, instead of discrete ones. The set of the nearest neighbors, $\text{Near}(k) = \langle (\mathbf{x}_1^N, y_1^N), (\mathbf{x}_2^N, y_2^N), \dots, (\mathbf{x}_k^N, y_k^N) \rangle$, where $y_i \in \mathcal{R}$ is retrieved and the output value of the novice example is estimated by averaging the output values of the nearest neighbors in the simple voting, as shown in Eq. (5.12),

$$\hat{y} = \frac{1}{k} \sum_{i=1}^k y_i \quad (5.12)$$

In the weighted voting, the linear combination of the output values of the nearest neighbors is expressed as Eq. (5.13) is the output of the novice item.

$$\hat{y} = \sum_{i=1}^k w_i y_i \quad (5.13)$$

It is possible to apply the KNN algorithm to the regression as well as the classification.

5.3.4 Attribute Discriminations

This section is concerned with the case of discriminating the attributes by their influence. In the previous section, we already studied the KNN algorithm where the similarities of the novice item with the training examples are computed without the discrimination among attributes. This section focuses on the scheme of assigning a different weight to each attribute, depending on the correlation between its values and the output values. The similarities of the novice example with the training examples are computed, considering the different weights that are assigned to the attributes, while using the KNN algorithm for the classification and the regression. This section is intended to mention the three cases of discriminating the attributes.

Let us consider the case of the discretely valued attributes in discriminating them. All possible values are enumerated for each attribute. The distribution over the categories is observed for each value. Higher weights are assigned to the attributes with their unbalanced distribution over categories. The process is applied to the construction of the decision tree from the training examples and is studied in Chap. 7.

Let us discriminate the attributes with their continuous values. Because the given problem is assumed to be a classification task, it is assumed that the output value is discrete, accordingly. As the codification, a numerical value is assigned to each category, arbitrary, and the correlation coefficient between an attribute and the output value is computed. The weights are assigned to the attributes, proportionally to the absolute value of their correlation coefficients with the output values. In other words, higher weights are set to attributes whose correlation coefficients close to -1 or 1 .

Let us consider the discrimination among the attributes in using the KNN algorithm for the regression task. It is assumed that attributes values and the output value are continuous. The correlation coefficient between attribute values and the output values is computed for each attribute. The weights are assigned to the

attributes, proportionally to the absolute values of their correlation coefficients. Refer to [1], for finding the attribute discrimination process in detail.

Let us make some remarks on the discrimination among the attributes in computing the similarity between two items. It is assumed that the attribute values are given as normalized values between zero and one. In the initial version of the KNN algorithm, it is assumed that the attributes have their identical influences on classifying items. The misclassification may be caused by putting higher weights to the less correlated attributes and lower weights to the more correlated attributes, implicitly. We need to discriminate among the training examples as well as the attributes for improving the classification performance of the KNN classification algorithm.

5.4 Variants

This section is concerned with the variants that are derived from the KNN algorithm that is covered in Sect. 5.3. In Sect. 5.4.1, we mention the KNN version where the number of nearest neighbors is set dynamically. In Sect. 5.4.2, we mention both the Radius Nearest Neighbor and its advanced version. In Sect. 5.4.4, we consider both the direct nearest neighbors and indirect ones as the hierarchical nearest neighbors. In Sect. 5.4.4, we point out the hub examples that should be removed while using the KNN algorithm.

5.4.1 *Dynamic Nearest Neighbor*

This section is concerned with the KNN variant where the number of nearest neighbors is changed automatically. The KNN algorithm that was entirely studied in Sect. 5.3 is the initial version where the number of nearest neighbors is initially set and fixed subsequently. In this version, the two external parameters, the initial number of nearest neighbors and the similarity threshold, are used and more nearest neighbors are added voting their labels, depending on the similarities among training examples. Utilizing additional similar training examples for classifying items more reliably is intended for this version. This section is intended to describe the motivation and the classification process of this KNN variant.

The necessity of setting the number of nearest neighbors dynamically is illustrated in Fig. 5.10. In using the KNN algorithm, it is assumed that the number of nearest neighbors is fixed to three. In the left part of Fig. 5.10, the third nearest neighbor is not needed for voting the labels, whereas in the right part of Fig. 5.10, one more nearest neighbor is needed. From Fig. 5.10, it is necessary to set differently the number of nearest neighbors, depending on the distribution over the nearest neighbors. In this section, we mention the process of classifying items under the dynamic number of nearest neighbors.

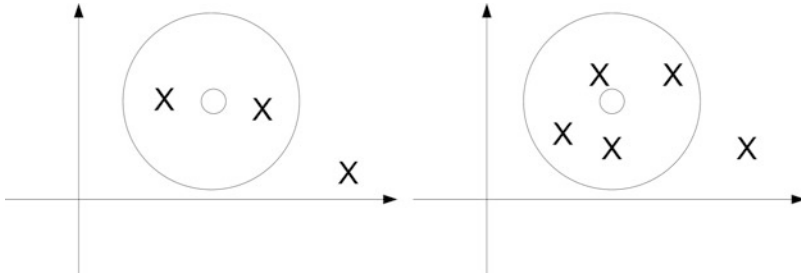


Fig. 5.10 Dynamic number of nearest neighbors

```

classifyByDynamicKNN(List trainingExampleList, Item example, real similarityThreshold){
    return classifyByKNN(trainingExampleList, example, similarityThreshold, 1);
}

classifyByKNN(List trainingExampleList, Item example, real similarityThreshold
              int nearestNeighborNumber){
    if(noMoreNearestNeighbor(trainingExampleList, example, nearestNeighborNumber)
        return voteLabel(trainingExampleList, nearestNeighborNumber);
    return classifyByKNN(trainingExampleList, similarityThreshold, nearestNeighborNumber + 2);
}

noMoreNearestNeighbor(List trainingExampleList, Item example, real similarityThreshold
                      int nearestNeighborNumber){
    kthNearestNeighbor = trainingExampleList.getLastNearestNeighbor(example, nearestNeighborNumber);
    nextNearestNeighbor = trainingExampleList.getLastNearestNeighbor(example, nearestNeighborNumber+2);
    if(kthNearestNeighbor.getSimilarity(nextNearestNeighbor) <= similarityThreshold)
        return true;
    return false;
}

```

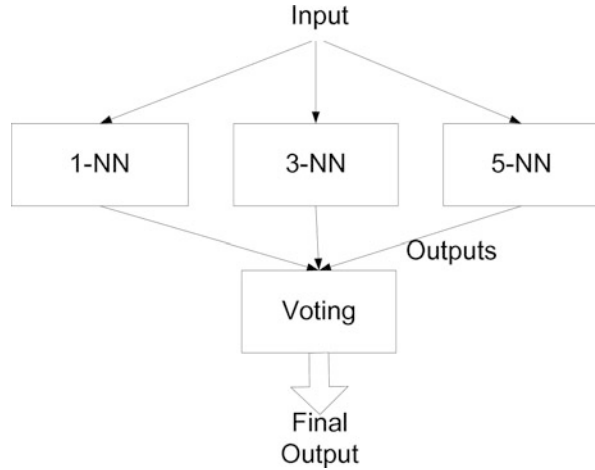
Fig. 5.11 Classification process in pseudo code

The process of classifying items is illustrated as a pseudo code in Fig. 5.11. The similarity threshold is given as an additional external parameter in this variant. When two more training examples have their higher similarities as the novice item, they are added to the nearest neighbors. The reason of selecting the additional nearest neighbors is to avoid even number of nearest neighbors. The KNN variant that is presented in Fig. 5.11 is the version that is modified slightly from the initial one.

The combination of multiple KNN algorithms for more reliability in classifying data items is illustrated in Fig. 5.12. We prepare the three KNN algorithms: 1-NN, 3-NN, and 5-NN. A data item is classified by the three KNN algorithms, and the final output is decided by voting ones of the three KNNs. When using the multiple KNN algorithms, the dual voting happens; one is to vote labels of nearest neighbors within each KNN, and the other is to do the labels of the three KNNs. The scheme of combining the three KNNs is called voting and studied in detail in Chap. 13.

Let us make some remarks on the KNN variant that allows the variable number of nearest neighbors during the execution. In this version, we consider the similarities and the distances of the nearest neighbors in selecting them for voting their labels. One more external parameter, the similarity threshold, is added taking additional nearest neighbors, in the algorithm that is presented in Fig. 5.11. Multiple KNN algorithms that are discriminated by the number of nearest neighbors are combined

Fig. 5.12 Multiple K nearest neighbors



for classifying data items with each other. The combination may be implemented as parallel algorithms where multiple KNN algorithms classify an item at the same time, independently.

5.4.2 Concentric Nearest Neighbor

This section is concerned with the CNN (Concentric Nearest Neighbor) as a variant of the KNN algorithm. The RNN (Radius Nearest Neighbor) is mentioned as a classification algorithm where the nearest neighbors are selected with the similarity more than the threshold, or within the radius. The CNN is viewed as a combination of multiple RNNs as a concentric form, and higher weights are assigned to nearest neighbors in the inner part. The CNN, which is mentioned in this section, is intended to solve the problem where no nearest neighbor is taken in the RNN. This section is intended to study the CNN as another KNN variant.

The process of classifying an item by the RNN (Radius Nearest Neighbors) is illustrated in Fig. 5.13. For each training example, its similarity with the novice item is computed like the case of the KNN algorithm. The training examples with their similarities that are more than the given threshold are selected as the nearest neighbor, and the label of the novice item is decided by voting their labels. The difference from the KNN algorithm is to select the training examples by the similarity threshold or the distance threshold, is called radius. The limit of this variant is the possibility of a very sparse number of nearest neighbors, depending on the area.

The multiple radiuses are set by using this classification algorithm, rather than a single radius, as shown in Fig. 5.14. Above, we pointed out the possibility of taking no nearest neighbor in a particular radius. The multiple radiuses are used like the

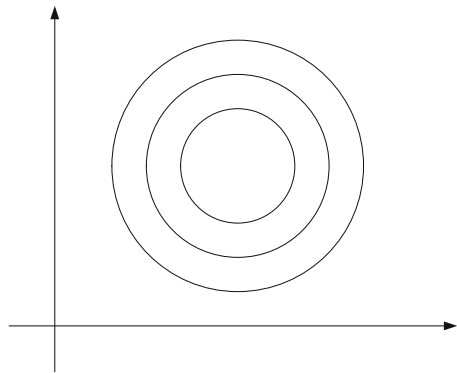

```

classifyByRNN (List trainingExampleList, Item example, real radius){
    for each trainingExample in trainingExampleList{
        distance = example.getDistance(trainingExample);
        if(distance <= radius)
            nearestNeighborList.add(trainingExample);
    }
    categoryName = nearestNeighborList.voting();
    return categoryName;
}

```

Fig. 5.13 Radius Nearest Neighbor

Fig. 5.14 Multiple radius
nearest neighbor



concentric circles as presented in Fig. 5.14, for avoiding the situation. The weights are assigned proportionally from the outside to the inner part. In this version, the nearest neighbors are qualified with the multiple levels.

The four schemes of weighting the nearest neighbors in the concentric form are illustrated in Fig. 5.15. In the first schemes, the constant weights are assigned to all of the neighbors, as presented in the top and left of Fig. 5.15. In the second scheme that is presented in the top right of Fig. 5.15, the weights are assigned, corresponding to each radius. In the third scheme, the weights are assigned linearly inverse proportional to the distance, as shown in the bottom and left of Fig. 5.15. In the last scheme, the weights are assigned exponentially inverse proportional to the distance as presented in the bottom-right of Fig. 5.15.

Let us make some remarks on the KNN variant that is covered in this section. The RNN where the training examples with more similarities than the threshold are selected as the nearest neighbors becomes the basis for deriving another kind of the KNN variant. The RNN is expanded into the variant by setting the multiple similarity thresholds. We consider various schemes of assigning weights to the nearest neighbors. Automatically, increasing or decreasing the similarity threshold depends on the sparseness of the nearest neighbor, as the hybrid of the KNN and the RNN.

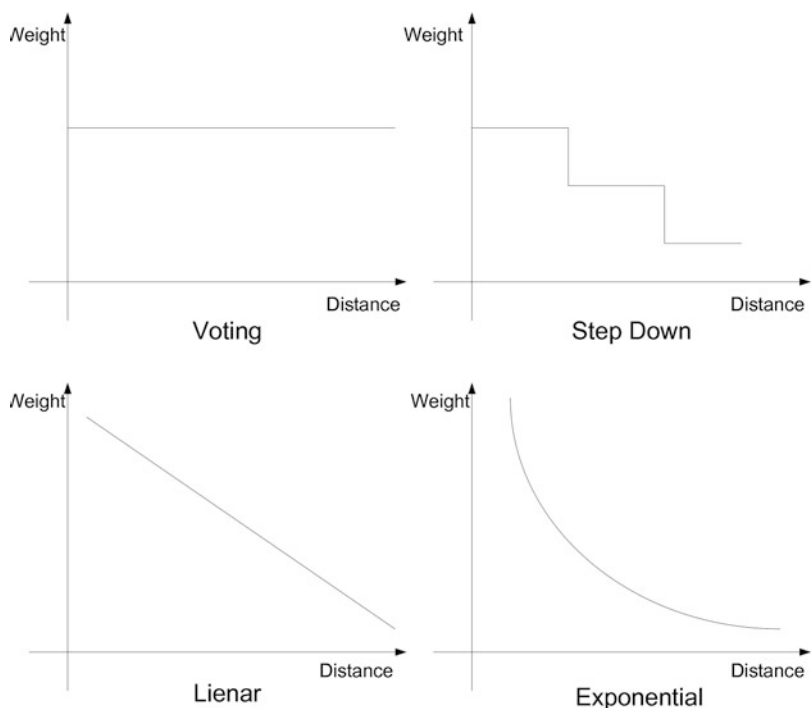


Fig. 5.15 Weighting schemes

5.4.3 Hierarchical Nearest Neighbor

This section is concerned with the KNN variant that considers the indirect neighbors. Until now, we have considered the only direct neighbors near the novice example. In this variant, the idea is to use the neighbors of a selected neighbor, called indirect neighbors, for deciding the label of a novice item. The neighbors are expanded hierarchically from the novice example. This section is intended to describe the KNN variant that builds neighbors hierarchically.

The direct nearest neighbor and the indirect ones are visualized in the two dimensional space, as shown in Fig. 5.16. This version is assumed as a RNN, and the vectors within the radius from the novice item are the direct nearest neighbors. We may consider another radius from a nearest neighbor, and the vectors within the radius are the indirect nearest neighbors. The indirect nearest neighbors are ones from a direct neighbor and expanded with multiple levels, hierarchically. The neighbors that are advanced by multiple levels become far from the novice item.

The process of classifying a novice item is illustrated as a pseudo code in Fig. 5.17. The direct nearest neighbors to the given novice item are extracted by computing its similarities with the training examples. For each direct nearest neighbor, its nearest neighbors are taken as the indirect nearest neighbors to the

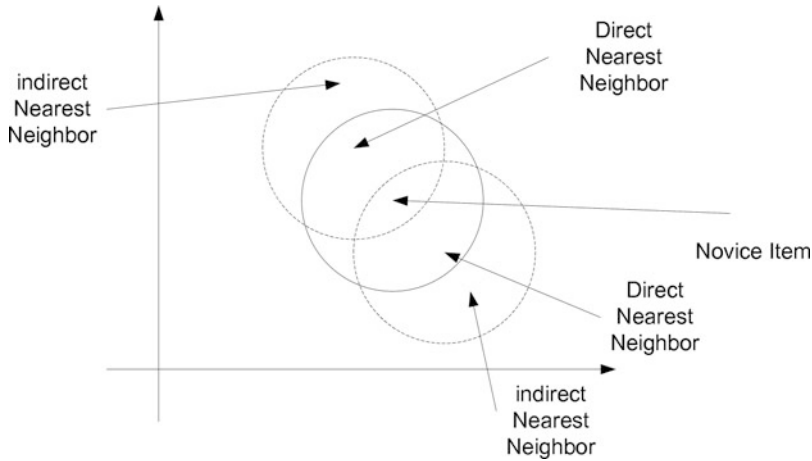


Fig. 5.16 Indirect nearest neighbor

```

classifyByIndirectKNN (List trainingExampleList, Item example, real similarityThreshold){
    directNearestNeighborList = example.getNearestNeighborList(trainingExampleList);
    Label classifiedLabel = nearestNeighborList.vote(categoryList);
    for each directNearestNeighbor in directNearestNeighborList{
        indirectNearestNeighborList = directNearestNeighbor.getNearestNeighborList(trainingExampleList);
        classifiedLabel.update(indirectNearestNeighborList.vote(categoryList));
    }
    return classifiedLabel.getLabelName();
}

```

Fig. 5.17 Classification process in pseudo code

novice item. The labels of the direct nearest neighbors and the indirect ones are voted with the higher weights that are assigned to the direct ones and the lower weights that are assigned to the indirect ones. The schemes of weighting the nearest neighbors are presented in Sect. 5.4.2.

The direct neighbors and the indirect ones are presented as a hierarchical structure in Fig. 5.18. The novice item that we try to classify is given as a root node in the tree. The direct nearest neighbors from the novice item are given as the nodes in the second level. The nearest neighbors from each direct neighbor are given as their child nodes. The view of the direct neighbors and the indirect neighbors into a hierarchical structure is the reason of calling this KNN variant hierarchical KNN algorithm.

Let us make some remarks on the KNN variant that is called hierarchical nearest neighbor. The direct neighbors are ones from a novice item, and the indirect neighbors are ones from each direct neighbor. When deciding the label of the novice item, the indirect neighbors are considered as well as the direct neighbors. The novice item, its direct neighbors, and the indirect neighbors are viewed as a tree as shown in Fig. 5.18. The scheme of weighting the direct neighbors and the indirect ones is mentioned in Sect. 5.4.2.

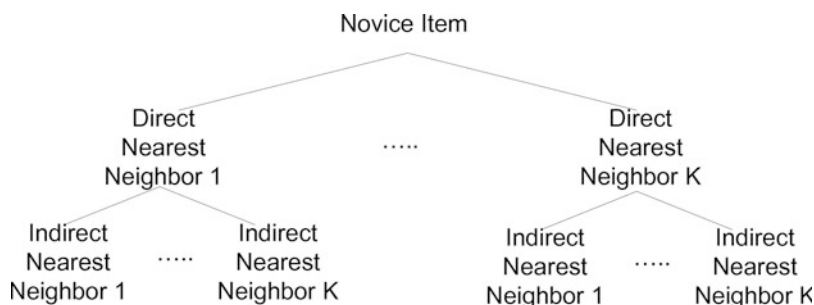


Fig. 5.18 Hierarchical representation of nearest neighbors

5.4.4 Hub Examples

The hub example is defined as one that has many neighbors within a distance. It becomes the cause of high probability of selecting examples around it, and the decision of the label of the novice example is skewed by ones around it. The hub degree of the training examples should be computed, and ones with their high hub degrees should be removed in advance. The weights should be assigned inversely proportionally to the hub degree to each nearest neighbor in voting their labels. This section is intended to describe the hub examples by using the KNN algorithm for the classification, in order to improve the classification performance.

The four types of the hub examples that influence on the classification performance by the KNN algorithm are presented in Fig. 5.19. The inner hub in the top left of Fig. 5.19 is the training example with many neighbors inside its own class. The boundary hub that is presented in the top left of Fig. 5.19 is one with many neighbors in the boundary between the two classes. The opposite hub that is presented in the bottom-left in Fig. 5.19 is one with many neighbors in the opposite class. The outer hub that is presented in the bottom-right of Fig. 5.19 is one with its neighbors in the outlier area that is far from the predefined categories.

Let us explore the influence of the four types of hub examples on the classification performance as presented in Fig. 5.20. The inner hub examples in the top left in Fig. 5.20 are useless in using the KNN algorithm. The boundary hub examples that are shown in the top right of Fig. 5.20 rise up the ambiguity for classifying novice examples. The opposite hub examples that are shown in the bottom-left of Fig. 5.20 become the causes of misclassifying items that should belong to the opposite class into the own class. The outlier hub examples that are shown in the bottom-right of Fig. 5.20 play the role of noises in the training set.

The process of removing the hub examples is illustrated as a pseudo code in Fig. 5.21. The radius is given as the parameter, and the training examples with its similar examples within the radius are regarded as hub examples. For each training example, if it has its more nearest neighbors than the threshold, it is removed as a hub example. The list of the training examples except the hub examples is

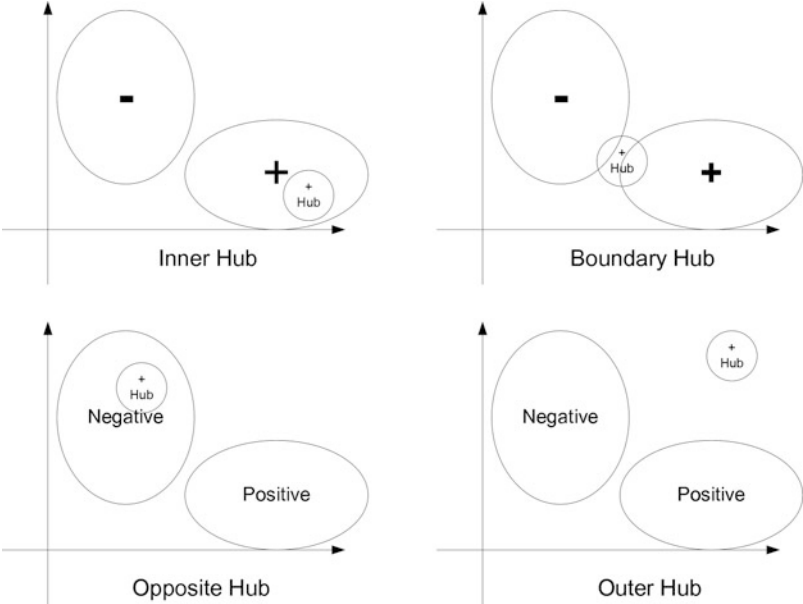


Fig. 5.19 Types of hub examples

returned. By removing the hub examples, it is expected to improve the classification performance by using the KNN algorithm.

Let us make some remarks on the hub examples that are mentioned as the issue in applying the KNN algorithm to the classification tasks. The hub example is the training example around which many training examples exist. It is asserted that the hub examples should be removed for improving the classification performance. We consider various schemes of detecting the hub examples by quantifying the distances and the number of nearest neighbors for each training example. The KNN versions that detect and remove the hub examples before classifying a novice item will be covered in the next study.

5.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We mentioned the Naive approach as the primitive machine learning algorithm that retrieves the most matching example and decides the label of the novice item by the label of the matching one. We expanded the Naive approach into the KNN algorithm and described it in detail with respect to its classification process. We mentioned the four variants that are derived from the KNN algorithm as its improved version.

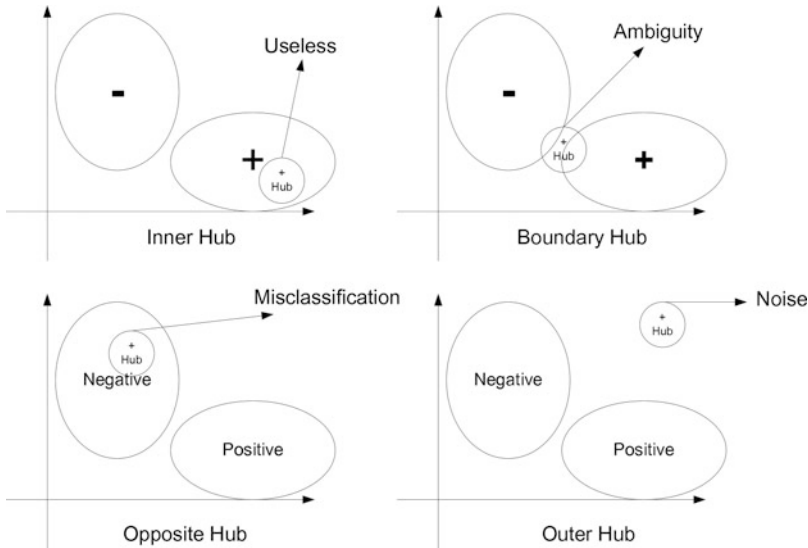


Fig. 5.20 Influence of hub examples on classification

```

removeHubExamples (List trainingExampleList, real radius, int maxNearestNumber){
    for each example in trainingExampleList{
        nearestNumber = example.getNearestNumber(radius);
        if(nearestNumber > maxNearestNumber)
            trainingExampleList.remove(example);
    }
    return trainingExampleList();
}

```

Fig. 5.21 Removal process of hub examples

We studied, in this chapter, the Naive approach as the prerequisite for the KNN algorithm, its initial version, and its variants.

The issue in using the KNN algorithm is how to decide the number of nearest neighbors. The labels of nearest neighbors are voted with their identical portions in the initial version of the KNN algorithm. The classification is biased strongly toward the nearest neighbors in the small number of nearest neighbors. In the large number of nearest neighbors, there is the risk of misclassifying item by the majority of relative distant neighbors. We may consider the discriminated portion among the nearest neighbors reversely proportionally to their distances.

There are various schemes of computing a similarity between items in using the KNN algorithm as a classification tool. The cosine similarity is the typical similarity metric between two numerical vectors and given as a normalized value. The inverse Euclidean distance is also the similarity metric between them. Many variants are derived from the cosine similarity. The similarities among attributes may be considered in computing the similarity between two items [2].

Let us consider various schemes in selecting the nearest neighbors. In the initial version of KNN algorithm, the number of nearest neighbors by ranking the training examples by their similarities with the novice item is fixed. Any training examples whose similarities are higher than the threshold are selected in the Radius Nearest Neighbors. The training examples are clustered with the regardless of their labels by their similarities, and ones in the cluster with its highest similarity become nearest neighbors. The nearest neighbors to the novice item are selected as direct ones, and nearest neighbors to each direct one are retrieved as indirect ones.

Let us mention the schemes of voting labels of the nearest neighbors for deciding one of a novice item. In the initial version of the KNN algorithm, the decision is made by the majority of labels of nearest neighbors with the same proportion to them. It may evolve into its more advanced version by assigning different weights to the nearest neighbors. They are graded with several levels, and different weights are assigned to them in voting their labels. We need to exclude the subexamples that are mentioned in Sect. 5.4.4, in voting labels.

References

1. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, Berlin, 2018)
2. T. Jo, Text classification using feature similarity based K nearest neighbor. *AS Medical Sci.* **3**(4), 13–21 (2019)

Chapter 6

Probabilistic Learning



6.1 Introduction

The probabilistic learning is defined as the type of supervised learning that computes the probability of a category given an item, called posterior using the Bayes rule. The probability of an individual example given a category, called a likelihood, is computed directly from the training examples. The posterior is computed by applying the Bayes rule to the likelihood in principle, but we are actually interested not only in its value but also in its comparison with the others. If it is assumed that probabilities of categories, called priors, are identical, the category is decided directly from the likelihoods. This section is intended to understand the basic concepts about the probabilistic learning.

Let us mention the conditional probability, $P(A|B)$, for understanding this kind of machine learning algorithms. $P(A)$ and $P(B)$ are the probabilities that indicate the possibilities of events A and B occurring, respectively. The conditional probability, $P(A|B)$, indicates the possibility of the event, A , occurring, under the assumption that event B occurs. The conditional probability, $P(A|B)$, is expressed as Eq. (6.1),

$$P(A|B) = \frac{P(AB)}{P(B)} \quad (6.1)$$

The conditional probability, $P(B|A)$, is expressed as Eq. (6.2),

$$P(B|A) = \frac{P(AB)}{P(A)} \quad (6.2)$$

The Bayes rule becomes the basis for building the probabilistic learning algorithms. The relation between the two conditional probabilities, $P(A|B)$ and $P(B|A)$, is expressed by the Bayes rule. From the training examples, we extract the conditional probability, $P(\mathbf{x}|C)$, called likelihood. We compute the conditional

probability, $P(C|\mathbf{x})$, called posteriori, by the Bayes rule of $P(\mathbf{x}|C)$. The Bayes rule is explained in detail in Sect. 6.2.2.

The goal of the probability learning is to compute the probability of category given a novice example. The probability is called posteriori and notated by $P(C|\mathbf{x})$. From the training examples, we may compute the probability of an example given the category, $P(\mathbf{x}|C)$, called likelihood. The probability of each category, called priori, is considered, and it is usually assumed that the priori over categories is identical to each other. Therefore, it is assumed that the posteriori, $P(C|\mathbf{x})$, is proportional to the likelihood, $P(\mathbf{x}|C)$.

The goal of this chapter is to understand the probabilistic learning algorithms as the most popular types of machine learning algorithms. We will understand the Bayes classifier as the initial type of probabilistic learning, assuming that each category is given as a Gaussian distribution. We need the Naive Bayes as the popular probabilistic learning type, assuming that attributes are independent of each other. We will also understand the Bayesian learning that consists of the two-step learning: the construction of the Bayesian Networks and the probability estimation from the training examples. We will make the further discussions for deriving more advanced probabilistic learning algorithms and the hybrid one that is mixed with other types of supervised learning algorithms.

6.2 Bayes Classifier

This section is concerned with the Bayes classifier where each category is assumed as a normal distribution. In Sects. 6.2.1 and 6.2.2, we review the probability and the Bayes rule as the basis for formulating the probabilistic classifier. In Sect. 6.2.3, we describe, in detail, the normal distribution that is assumed as the probability distribution over each category. In Sect. 6.2.4, we mention the process of classifying an item by the Bayes classifier. This section is intended to describe the Bayes classifier as the first version of probabilistic learning.

6.2.1 Probabilities

The probability is defined as the degree of possibility with which an event occurs. The event that is covered in the probability is the visible case, which is made by a particular behavior, such as the tail or the head of a coin, in case of tossing it. The random variable is the variable of numerical values that correspond to individual events. The expectation is the average over values of the random variable, and the variance is the variability over values of the random variable. This section is intended to mention events, the random variable, and the expectation for providing the basic concepts of probability.

A set of events is defined before considering the probabilities. There are two events, head and tail, in tossing a coin, and the event set is notated by $E = \{H, T\}$. There are six events in rolling a dice, and the set is notated by $E = \{1, 2, 3, 4, 5, 6\}$. The event set is expressed as the general form, $E = \{E_1, E_2, \dots, E_k\}$. The probability is mentioned as possibility metric with which each event occurs, given as an element of the set, E .

The random variable indicates a numerical value that expresses an event. The random variables are notated as X , Y , and Z , and in tossing a coin, it is expressed as $X = 1$ indicating the head and $X = 0$ indicating the tail. The random variable is expressed from 1 to 6 in rolling a dice. The function of the random variable for generating the probability is called the probability function and notated by $P(X)$. The probability function is characterized by Eqs. (6.3) and (6.4),

$$0 \leq P(X) \leq 1 \quad (6.3)$$

$$\sum_x P(X) = 1 \quad (6.4)$$

The expectation of the probability is the average over products of the probability and the random variable. It is expressed into Eq. (6.5),

$$E[X] = \sum_x X P(X) \quad (6.5)$$

The variance is expressed into Eq. (6.6),

$$V[X] = E[E[X] - X P(X)] \quad (6.6)$$

Equation (6.6) is modified into Eq. (6.7), for computing the variance easily from the expectation,

$$V[X] = E[X^2] - (E[X])^2 \quad (6.7)$$

For taking the process of deriving Eq. (6.7) from Eq. (6.6), refer to the literature [4].

Let us make some remarks on the probability. The area, probability, should be distinguished from its two similar areas, statistics and fuzzy theory. The probability is the area that deals with the possibility degree of which an event happens, whereas the statistics is the area for estimating values that characterize the population, using samples. The probability focuses on the possibility of event and assumes that each event is crisp, whereas the fuzzy theory considers the partial event as the mixture of happening and not-happening. The fuzzy probability combines the probability with the fuzzy theory; it defines the partial event and deals with its probability [2].

6.2.2 Bayes Rule

This section is concerned with the Bayes rule as the basis for deriving the probabilistic learning. The Bayes rule is the mathematical rule for relating the conditional probability, $P(A|B)$, with the other, $P(B|A)$. We mention the two events A and B , the probabilities of them, $P(A)$, and the conditional probabilities, $P(A|B)$ and $P(B|A)$, as the probabilities which are involved in studying the Bayes rule. In the context classification, $P(\mathbf{x}|C)$ is the likelihood that is the probability of the input vector, $\mathbf{x}$, given the category, C , and $P(C|\mathbf{x})$ is the posteriori that is the probability of the category, C , given in the input vector, $\mathbf{x}$. This section is intended to define the Bayes rule, mathematically, and connect it with the classification task.

Let us study the conditional probability briefly, before studying the Bayes rule. Let us assume the two events, A and B , and $P(A)$ and $P(B)$ are, respectively, the probability of event A and the probability of event B . $P(A|B)$ is the probability of event A under the case where event B happens, and $P(B|A)$ is the probability of event B under the case where event A happens. The two probabilities, $P(A|B)$ and $P(B|A)$, are expressed, respectively, into Eqs. (6.8) and (6.9),

$$P(A|B) = \frac{P(AB)}{P(B)} \quad (6.8)$$

$$P(B|A) = \frac{P(AB)}{P(A)} \quad (6.9)$$

When the two events, A and B , are independent of each other, the conditional probabilities are expressed into Eqs. (6.10) and (6.11),

$$P(A|B) = P(A) \quad (6.10)$$

$$P(B|A) = P(B) \quad (6.11)$$

Equation (6.12) is derived from Eqs. (6.10) and (6.11),

$$P(AB) = P(A)P(B) \quad (6.12)$$

Let us present the Bayes rule that shows the relation between the two conditional probabilities, $P(A|B)$ and $P(B|A)$. The probability, $P(AB)$, is expressed into Eqs. (6.13) and (6.14),

$$P(AB) = P(A|B)P(B) \quad (6.13)$$

$$P(AB) = P(B|A)P(A) \quad (6.14)$$

The conditional probability, $P(A|B)$, is expressed into Eq. (6.15),

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} \quad (6.15)$$

Equation (6.15), which is called Bayes rule, represents the relation between the two conditional probabilities, $P(A|B)$ and $P(B|A)$. It is used for computing $P(A|B)$ using $P(B|A)$.

The goal of the probability learning is to compute the probability of category given a novice example. The probability is called posteriori and notated by $P(C|\mathbf{x})$. From the training examples, we may compute the probability of an example given the category, $P(\mathbf{x}|C)$, called likelihood. The probability of each category, called priori, is considered, and it is usually assumed that the priori over categories is identical to each other. Therefore, it is assumed that the posteriori, $P(C|\mathbf{x})$, is proportional to the likelihood, $P(\mathbf{x}|C)$.

The goal of this chapter is to understand the probabilistic learning algorithms as the most popular types of machine learning algorithms. We will understand the Bayes classifier as the initial type of probabilistic learning, assuming that each category is given as a Gaussian distribution. We need the Naive Bayes as the popular probabilistic learning type, assuming that attributes are independent of each other. We will also understand the Bayesian learning that consists of the two-step learning: the construction of the Bayesian Networks and the probability estimation from the training examples. We will make the further discussions for deriving more advanced probabilistic learning algorithms and the hybrid one that is mixed with other types of supervised learning algorithms.

6.2.3 Gaussian Distribution

This section is concerned with the Gaussian distribution that is assumed to be identically labeled training examples. The Bayes classifier is called parametric classification algorithm where the predefined categories are assumed to be probability distributions. It is assumed that the distribution over the identically labeled training examples is the Gaussian distribution; each category corresponds to its own Gaussian distribution. It is possible to define each category as an alternative distribution such as uniform distribution or fuzzy distribution, but this case is set out of this study. This section is intended to describe the Gaussian distribution based on its parameters.

Let us mention the Gaussian distribution over the training examples that are given as numerical vectors. It is assumed that the classification task is given as a binary classification, so the two sets of training examples are given as the set of training examples that are labeled with the positive class, $C_+ = \{\mathbf{x}_{+1}, \mathbf{x}_{+2}, \dots, \mathbf{x}_{+m}\}$ and the set of ones that are labeled with the negative class, $C_- = \{\mathbf{x}_{-1}, \mathbf{x}_{-2}, \dots, \mathbf{x}_{-m}\}$. The positive class and the negative class are defined as the Gaussian distributions whose mean vectors are $\boldsymbol{\mu}_+ = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_{+i}$ and $\boldsymbol{\mu}_- = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_{-i}$. The $d \times d$ covariance matrix is set in both classes as a diagonal matrix as follows:

$$\begin{bmatrix} k & 0 & \dots & 0 \\ 0 & k & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & k \end{bmatrix}$$

The Bayes classifier tends to depend only on mean vectors for classifying items, ignoring the covariance matrix.

Let us mention another style of the Gaussian distribution over numerical vectors. In the previous case, the covariance matrix is assumed to be one where the diagonal values are identical constants and the others are zero values. As a slightly more advanced Gaussian distribution, different variances are expressed as a diagonal covariance matrix as follows:

$$\begin{bmatrix} k_1 & 0 & \dots & 0 \\ 0 & k_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & k_d \end{bmatrix}$$

where $k_1 \neq k_2 \neq \dots \neq k_d$. The factors are also assumed to be independent ones, in this case. These variances are used as different weights for computing a similarity between mean vectors.

Let us mention the general type of Gaussian distribution where its covariance matrix is given as a non-diagonal matrix. If the features are assumed to be independent of each other, the covariance matrix is given as a diagonal matrix like the above cases. The probability of a vector which represents an item, which is given as a vector, $\mathbf{x}$, is estimated as equation (6.16),

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \quad (6.16)$$

where $\boldsymbol{\mu}$ is the mean vector, and $\mathbf{\Sigma}$ is the covariance matrix. Computing the determinant of the covariance matrix, $|\mathbf{\Sigma}|$, and the inverse matrix, $\mathbf{\Sigma}^{-1}$, takes very high complexity. In defining the Gaussian distribution for each category, it is usually assumed that the features are independent of each other.

Let us make some remarks on the Gaussian distribution, which is assumed as the probability distribution for each category in the Bayes classifier. The distribution over scalars is characterized by its mean and its variance. The Gaussian distribution over vectors is done by its mean vector and its covariance matrix. When it is assumed that features are independent of each other, the covariance matrix is given as a diagonal matrix. When applying the Gaussian distribution to each category as its distribution in using the Bayes classifier, it is assumed that features are independent of each other for simplicity.

6.2.4 Classification

This section is concerned with the Bayes classifier as a parametric classification algorithm where distributions over the training examples are assumed. Each category is assumed as a Gaussian distribution that is mentioned in Sect. 6.2.3. It is characterized by a mean vector and a covariance matrix. A novice item is classified by its distance from the category mean vectors, assuming the covariance matrices as diagonal ones in all categories. This section is intended to describe the process of classifying a novice item by assuming a distribution for each category.

The first step of applying the Bayes classifier to the classification task is to define the distribution for each category. It is assumed that the problem is given as a binary classification, and the training examples are labeled with the positive class or the negative class. The training set is notated by $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and the training set is divided into the positive set, the set of the training examples that are labeled with the positive class, $Tr_+ = \{\mathbf{x}_1^+, \mathbf{x}_2^+, \dots, \mathbf{x}_{\frac{N}{2}}^+\}$, and the negative set, the set of training examples that are labeled with the negative class, $Tr_- = \{\mathbf{x}_1^-, \mathbf{x}_2^-, \dots, \mathbf{x}_{\frac{N}{2}}^-\}$, under the assumption of the balanced distribution over the two categories. For the simplicity, the covariance matrix is assumed as a constant diagonal matrix, and the mean vectors of the positive class and the negative class are computed by Eq. (6.17)

$$\mu_+ = \frac{2}{N} \sum_{i=1}^{\frac{N}{2}} \mathbf{x}_i^+, \quad \mu_- = \frac{2}{N} \sum_{i=1}^{\frac{N}{2}} \mathbf{x}_i^- \quad (6.17)$$

The covariance matrix and the mean vector characterize the Gaussian distribution of the positive class or the negative class.

Let us mention the process of computing likelihoods in the Bayes classifier. In Sect. 6.2.2, we mentioned the posteriori that indicates directly the categorical score and is proportional to the likelihood. When the novice vector, $\mathbf{x}$, is given as the input, the likelihood of the category C_i , $P(\mathbf{x}|C_i)$, is computed under the assumption of the category, C_i , given as the Gaussian distribution. The inverse Euclidean distances or the cosine similarities with the category mean vectors are proportional to the likelihoods under the assumption of the covariance matrices that are set identically to all categories. The posteriori is mapped with the cosine similarity with each mean vector in the Bayes classifier.

Let us mention the process of classifying a novice item by the Bayes classifier. The labeled examples are given as the training examples, and the mean vectors are computed category by category. The similarities with the mean vectors or the distances from them of the novice item are computed. The label of the novice item is decided by the category whose mean vector is most similar or least distant. Because the category likelihoods are proportional to the similarities with the category vectors, the direct calculation of them, $P(\mathbf{x}|C_1)$, $P(\mathbf{x}|C_2)$, $\dots$, $P(\mathbf{x}|C_m)$, is omitted.

Let us make some remarks on the Bayes classifier that is described in this section. It is assumed that each category is given as its own Gaussian distribution in this kind of classification algorithm. The Gaussian distribution over each category is characterized with its mean vector and its covariance matrix. Each item is classified based on its similarities with the category mean vectors, under the assumption that the covariance matrices are constant diagonal ones in all categories. When using the covariance matrices that are different to each category in their general form, this algorithm becomes very complicated.

6.3 Naive Bayes

This section is concerned with the Naive Bayes as the most popular probabilistic classification algorithm. In Sect. 6.3.1, we explain the process of classifying data items under the optimized parameters by the Naive Bayes. In Sect. 6.3.2, we describe the process of computing the parameters of Naive Bayes, using the training examples as the learning process. In Sects. 6.3.3 and 6.3.4, we mention the variants of Naive Bayes and its application to the text classification, respectively. This section is intended to describe the Naive Bayes with respect to its classification, its learning, its variants, and its application.

6.3.1 Classification

This section is concerned with the process of classifying items using the Naive Bayes. In this section, it is assumed that the parameter of the Naive Bayes is optimized by the training examples. There are three steps in classifying items in the Naive Bayes: the probability estimation of individual attribute values, the likelihood estimation, and the category decision. The learning process and the variants of the Naive Bayes are covered in the subsequent sections. This section focuses on the classification process of the Naive Bayes under the assumption of its optimized parameters.

The process of computing the parameters of the Naive Bayes from the training examples, which is called learning process, is illustrated in Fig. 6.1. It is assumed that the given problem is a binary classification, and each vector is a binary vector where each element is one or zero. The likelihood of the individual attribute value given the category is rate of the number of training examples whose attribute is the corresponding value and is labeled with the category to the total number of the training examples that are labeled with the category. For example, $P(a_i = 0|+)$ is the likelihood of given the positive class and is expressed with Eq. (6.18),

$$P(a_i = 0|+) = \frac{\#((a_i = 0) \wedge (+))}{\#(+)} \quad (6.18)$$

$$P(a_i = 0 | +) = \frac{\# \text{Sample Examples with } a_i = 0 \text{ in Positive Class}}{\# \text{Sample Examples in Positive Class}}$$

Sample Examples		
Positive Class	Negative Class	
[1, 1, 1, 0, 0, 1]	[0, 0, 0, 1, 1, 1]	$P(a_1 = 0 +) = 0, P(a_1 = 0 -) = 1$
[1, 1, 1, 0, 1, 0]	[0, 0, 0, 0, 1, 1]	$P(a_1 = 1 +) = 1, P(a_1 = 1 -) = 0$
[1, 1, 1, 0, 0, 0]	[0, 0, 0, 1, 0, 1]	$P(a_2 = 0 +) = 0, P(a_2 = 0 -) = 0.667$
[1, 1, 0, 0, 0, 0]	[0, 0, 1, 1, 1, 1]	$P(a_2 = 1 +) = 1, P(a_2 = 1 -) = 0.333$
[1, 1, 0, 1, 0, 0]	[0, 1, 0, 1, 1, 1]	$P(a_3 = 0 +) = 0.333, P(a_3 = 0 -) = 0.833$
[1, 1, 1, 1, 0, 0]	[0, 1, 0, 1, 0, 1]	$P(a_3 = 1 +) = 0.667, P(a_3 = 1 -) = 0.167$
		$P(a_4 = 0 +) = 0.667, P(a_4 = 0 -) = 0.167$
		$P(a_4 = 1 +) = 0.333, P(a_4 = 1 -) = 0.833$
		$P(a_5 = 0 +) = 0.833, P(a_5 = 0 -) = 0.667$
		$P(a_5 = 1 +) = 0.167, P(a_5 = 1 -) = 0.333$
		$P(a_6 = 0 +) = 0.833, P(a_6 = 0 -) = 0$
		$P(a_6 = 1 +) = 0.167, P(a_6 = 1 -) = 1$

Fig. 6.1 Parameters of Naive Bayes

$$\begin{aligned}
 P([1, 1, 0, 0, 0, 1] | +) &= 1 * 1 * 0.333 * 0.667 * 0.833 * 0.167 \\
 P([1, 1, 0, 0, 0, 1] | -) &= 0 * 0 * 0.667 * 0.333 * 0.167 * 0.833 \\
 P([1, 0, 0, 1, 1, 1] | +) &= 1 * 0 * 0.333 * 0.333 * 0.167 * 0.167 \\
 P([1, 0, 0, 1, 1, 1] | -) &= 0 * 1 * 0.667 * 0.667 * 0.833 * 0.833
 \end{aligned}$$

Fig. 6.2 Likelihood computations

where $\#((a_i = 0) \wedge (+))$ is the number of the training examples whose attribute, a_i , is zero and that are labeled with the positive class, and $\#(+)$ is the number of the training examples that are labeled with the positive class. The number of parameters in the Naive Bayes is given as $\# \text{categories} \times \# \text{attribute} \times \# \text{values}$, assuming that each attribute has the same number of values.

The process of computing the category likelihoods to the novice examples is illustrated in Fig. 6.2. The two novice examples, [1 1 0 0 0 1] and [1 1 0 1 1 1], are given as the six dimensional binary vectors and are classified, respectively, into the positive class and the negative class. In classifying the example, [1 1 0 0 0 1], the non-zero likelihood is directed to the positive class, and the zero likelihood is done to the negative class. In classifying the example, [1 1 0 1 1 1], the zero likelihoods are favored to both the classes, so it is rejected in this task. If the likelihood of an attribute value, at least, is zero, the likelihood of the entire vector becomes zero.

The smoothing operation of the Naive Bayes for avoiding the above problem is illustrated in Fig. 6.3. The likelihood of the vector is computed by the product of likelihoods of individual elements under the assumption of independence among attributes. As an alternative way, we consider computing the likelihood of the vector to each category by averaging attribute likelihoods. Even if both items are classified into the same class, the zero likelihood to the negative class is avoided with more probability. Averaging over the individual attribute likelihoods is safer than the product of them.

$$\begin{aligned}
P([1, 1, 0, 0, 0, 1]|+) &= (1 + 1 + 0.333 + 0.667 + 0.833 + 0.167)/6 = 0.667 \\
P([1, 1, 0, 0, 0, 1]|-) &= (0 + 0 + 0.667 + 0.333 + 0.167 + 0.833)/6 = 0.333 \\
P([1, 0, 0, 1, 1, 1]|+) &= (1 + 0 + 0.333 + 0.333 + 0.167 + 0.167)/6 = 0.333 \\
P([1, 0, 0, 1, 1, 1]|-) &= (0 + 1 + 0.667 + 0.667 + 0.833 + 0.833)/6 = 0.667
\end{aligned}$$

Fig. 6.3 Fuzzy classification

Let us make some remarks on the process of classifying items by the Naive Bayes. The assumption of the independence among attributes is inherent in the classification algorithm. Even if the attributes are actually dependent on each other in the classification tasks, the performance keeps sound. It is very fragile in the product of likelihoods of individual attribute values for computing the likelihood of a vector. Equation (6.18) is modified into Eq. (6.19),

$$P(a_i = 0|+) = \frac{K + \#((a_i = 0) \wedge (+))}{K + \#(+)} \quad (6.19)$$

where K is a constant, in order to avoid the zero in the smoothing version.

6.3.2 Learning

This section is concerned with the learning process of the Naive Bayes from the training examples. It is assumed that the problem is given as a binary classification and the training examples are labeled with the positive class or the negative class. The attribute values are given as binary values for the simplicity in computing the parameters as the additional assumption. The likelihoods of individual attribute values to both classes are given as parameters. This section is intended to describe the process of computing the likelihoods from the training examples as the learning process.

If an attribute is continuous, it is necessary to discretize it for training the Naive Bayes. The assumption that is inherent in using the Naive Bayes as a classifier is that all of the attributes are discrete. The discretization is the process of dividing an entire value range of the attribute into a finite number of value intervals; each value interval is called bin. The number of value intervals becomes the issue in discretizing a continuous attribute. Assuming that all of the attribute values are given as binary values, we will explain the learning process of the Naive Bayes.

The likelihoods of individual attribute values to categories are computed as the learning process of the Naive Bayes. It is assumed that the given problem is a binary classification and individual attribute values are given as binary values, as mentioned above. The number of the training examples that are labeled with the positive class and the number of ones that are labeled with the negative class are notated, respectively, by N_+ and N_- , and the number of training examples, where

$a_i = v_i$, which are labeled with the positive class is notated by $N_+(a_i = v_i)$. The likelihood of the attribute value, to the positive class, $P(a_i = v_i|+)$, is computed by Eq. (6.20),

$$P(a_i = v_i|+) = \frac{N_+(a_i = v_i)}{N_+} \quad (6.20)$$

And the likelihood to the negative class is computed by Eq. (6.21),

$$P(a_i = v_i|-) = \frac{N_-(a_i = v_i)}{N_-} \quad (6.21)$$

The number of parameters is $2 \times d \times 2$ in encoding data items into d dimensional binary vectors for the binary classification.

The likelihood of the vector, $\mathbf{x}$, to the category, C , $P(\mathbf{x}|C)$, is computed by products of likelihoods of individual attribute values to the category, under the assumption of the independence among attributes. The likelihood, $P(\mathbf{x}|C)$, where $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$, is expressed as Eq. (6.22),

$$P(\mathbf{x}|C) = \sum_{i=1}^d P(x_i|C) \quad (6.22)$$

The fact that a likelihood of an individual attribute value is given as zero, causes the zero likelihood. We need to modify Eqs. (6.20) and (6.21) into Eqs. (6.23) and (6.24),

$$P(a_i = v_i|+) = \frac{K + N_+(a_i = v_i)}{K + N_+} \quad (6.23)$$

$$P(a_i = v_i|-) = \frac{K + N_-(a_i = v_i)}{K + N_-} \quad (6.24)$$

where K is a constant; this is called the smoothing operation. The zero likelihood should be avoided by using Eqs. (6.23) and (6.24), instead of Eqs. (6.20) and (6.21), for computing likelihoods of individual attribute values to the category.

Let us consider the granularity of discretizing continuous attribute values for using the Naive Bayes. The granularity is the size of each value interval, called bin. A big granularity is the case where the value interval is big and the number of discrete values is small in an attribute, and a small granularity is the opposite case. In the big granularity, a small number of parameters are computed, but the classification is less reliable, whereas in the small granularity, a large number of parameters are computed, but the classification is more reliable. In the very small granularity, the risks are more possibility of zero likelihoods and the overfitting where the performance is good at the training examples, but poor to the novice examples.

6.3.3 Variants

This section is concerned with the variants that are derived from the Naive Bayes, which is described in Sect. 6.3.2. The classification process and the learning process of the Naive Bayes are covered in the previous sections. The variants are derived from the Naive Bayes by discriminating the attributes and the training examples. In viewing the set of training examples as an attribute–item matrix, the attributes that are given as the columns and the training examples that are given as the rows are discriminated. This section is intended to mention the three variants of the Naive Bayes.

Because the logarithmic is characterized as the proportionality and the smoothing, it is used, instead of the direct value. Let us remind Eqs. (6.23) and (6.24) for computing the likelihoods, as mentioned in Sect. 6.3.2. The equations are modified as the logarithmic scales into Eqs. (6.25) and (6.26),

$$P(a_i = v_i|+) = \frac{\log(K + N_+(a_i = v_i))}{\log(K + N_+)} \quad (6.25)$$

$$P(a_i = v_i|-) = \frac{\log(K + N_-(a_i = v_i))}{\log(K + N_-)} \quad (6.26)$$

The variant is usually applicable to the very large number of training examples. It is intended to rescale the likelihoods, rather than improving the performance.

Let us mention another variant of the Naive Bayes where the attributes are discriminated. Each attribute influences differently on classifying data items; we need to discriminate the attributes by assigning their different weights. The weights that are assigned to d attributes are notated by $w_1, w_2, \dots, w_d$, where $\sum_{i=1}^d = 1.0$. The likelihoods are computed by the adjusted equations, Eqs. (6.27) and (6.28),

$$P(a_i = v_i|+) = w_i \frac{K + N_+(a_i = v_i)}{K + N_+} \quad (6.27)$$

$$P(a_i = v_i|-) = w_i \frac{K + N_-(a_i = v_i)}{K + N_-} \quad (6.28)$$

The attribute weights are decided based on the correlation between the attribute values and the output values.

Because the individual training examples have their own different quality, we need to consider the discriminations among them. We assume the weights, $w_1, w_2, \dots, w_N$, which correspond to the training examples, based on their qualities. Equations (6.23) and (6.24) are Modified, respectively, into Eqs. (6.29) and (6.30),

$$P(a_i = v_i | +) = \frac{K + \sum_{j \in N_+(a_i=v_i)} w_j}{K + \sum_{j \in N_+} w_j} \quad (6.29)$$

$$P(a_i = v_i | -) = \frac{K + \sum_{j \in N_-(a_i=v_i)} w_j}{K + \sum_{j \in N_-} w_j} \quad (6.30)$$

It is required to define the policy for judging the qualities of the training examples for using this variant of the Naive Bayes. The weights of the training examples are tuned using the validation set that is separated from the training set.

Let us consider the case of combining the Naive Bayes with the KNN algorithm that was studied in Chap. 5. This combination is intended to consider only nearest training examples for computing the likelihoods of novice example to the categories. In classifying the novice item, the nearest neighbors are retrieved using the KNN algorithm, and the likelihoods are computed by the Naive Bayes. The learning process is executed differently to each novice example, depending on its nearest neighbors, instead of all training examples, is called local learning. In the local learning, it is expected to improve the classification performance, but the learning process should be executed whenever an individual novice item is given as the payment.

6.3.4 Application to Text Classification

This section is concerned with the process of applying the Naive Bayes to the text classification. In the previous work, the case of applying so is mentioned [1]. A text is encoded into a binary vector whose attributes are words, and each attribute value is given as zero that indicates the presence of the word in the text and one that indicates its absence. Its learning process is to compute the likelihoods of individual values to the predefined categories. This section is intended to describe the process of applying the Naive Bayes to the text classification.

The raw texts are encoded into binary vectors whose attributes are the selected words. The words are extracted from a corpus by indexing it. Some words are selected as features among them by their weights. For each feature, zero that indicates its absence in the text or one that indicates its presence is assigned. The process of encoding texts into numerical vectors was covered in Chap. 3.

Let us mention the learning process in applying the Naive Bayes to the text classification. For the simplicity, it is assumed that each text is encoded into a d dimensional binary vector, and each text is classified into the positive class or the negative class, and the likelihoods of each attribute value to the positive class and the negative class are computed. The attribute and the word are notated, respectively, by a_i and t_i , and the likelihoods in the presence of the word, t_i , are computed by Eqs. (6.31) and (6.32),

$$P(a_i = v_i | +) = \frac{|Tr_+ \cap Tr_i|}{|Tr_+|} \quad (6.31)$$

$$P(a_i = v_i | -) = \frac{|Tr_- \cap Tr_i|}{|Tr_-|} \quad (6.32)$$

where Tr_+ is the set of the sample texts that are labeled with the positive class, Tr_- is the set of sample texts that are labeled with the negative class, and Tr_i is the set of the samples texts that include the word, t_i . In the case of the absence of the word, t_i , the likelihoods are computed by Eqs. (6.33) and (6.34),

$$P(a_i = v_i | +) = \frac{|Tr_+ - Tr_i|}{|Tr_+|} \quad (6.33)$$

$$P(a_i = v_i | -) = \frac{|Tr_- - Tr_i|}{|Tr_-|} \quad (6.34)$$

The number of likelihoods of individual attribute values in the d dimensional binary vector, as parameters of the Naive Bayes, is $2 \cdot d$.

A novel text is classified after training the Naive Bayes by the above process. The learning of the Naive Bayes is to compute the above likelihoods from the training examples that are given as binary vectors. A novice vector is given as d dimensional binary vector, $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$, and the likelihoods of corresponds to the values are retrieved as $P(a_i = x_i | +)$ and $P(a_i = x_i | -)$ for each element. The likelihoods of the vector to the categories are computed by Eqs. (6.35) and (6.36),

$$P(\mathbf{x} | +) = \prod_{i=1}^d P(a_i = x_i | +) \quad (6.35)$$

$$P(\mathbf{x} | -) = \prod_{i=1}^d P(a_i = x_i | -) \quad (6.36)$$

The risk of multiplying them is the possibility of generating zero probability easily, so the multiplication may be replaced by averaging, as shown in Eqs. (6.37) and (6.38),

$$P(\mathbf{x} | +) = \frac{1}{d} \sum_{i=1}^d P(a_i = x_i | +) \quad (6.37)$$

$$P(\mathbf{x} | -) = \frac{1}{d} \sum_{i=1}^d P(a_i = x_i | -) \quad (6.38)$$

The Naive Bayes model is applied to the text classification in this section, and let us make remarks on it. In encoding texts into numerical vectors, the words are given as the features and the binary values are given as the attribute values. The given problem is assumed to a binary classification where each text is classified into the positive class or the negative class. The text classification belongs to the multiple classifications, rather than the binary classification, actually, but it may be decomposed into binary classification tasks as many as categories; the fact becomes the reason of assuming the task as a binary classification. When the frequencies are used as feature values, instead of the binary values, the initial version is recommended to be replaced by the smoothing version.

6.4 Bayesian Learning

This section is concerned with the more advanced probabilistic learning that is called Bayesian Learning. In Sect. 6.4.1, we present and interpret graphically the Bayesian Networks. In Sect. 6.4.2, we express the causal relations that are given as edges in the Bayesian Networks as equations. In Sect. 6.4.3, we describe the process of learning training examples for constructing the Bayesian Networks. In Sect. 6.4.4, we compare the three probabilistic learning algorithms that are covered in this chapter with each other.

6.4.1 Bayesian Networks

This section is concerned with the graphical representation of the Bayesian Networks. The attributes are assumed as independent ones in studying the Naive Byes, but the dependencies among them exist in reality. The Bayesian Network is a directed graph where its nodes are the attributes and its edges are the causal relations. In an edge between two nodes, the start node indicates the conditional attribute and the end node indicates the causal one. This section is intended to present and interpret graphically the Bayesian Networks.

Let us explore the graph types, which is presented in Fig. 6.4, before discussing the Bayesian Networks. The graph in the top left of Fig. 6.4 belongs to the unweighted and undirected graph where neither weight nor direction is assigned to each edge. The graph in the top right belongs to the unweighted and direct graph, where each edge is given as an arrow, instead of a line; for example, the edge between node A and node B indicates that node A is the source and node B is the destination. The graph in the bottom-left is a weighted and undirected graph where a weight is assigned to each edge and it is given as a line. The graph in the bottom-right is a weighted and directed graph.

The causal relation between two nodes is illustrated in Fig. 6.5, as the simplest Bayesian Networks. Two attributes, A and B, are given as nodes, and a single edge is

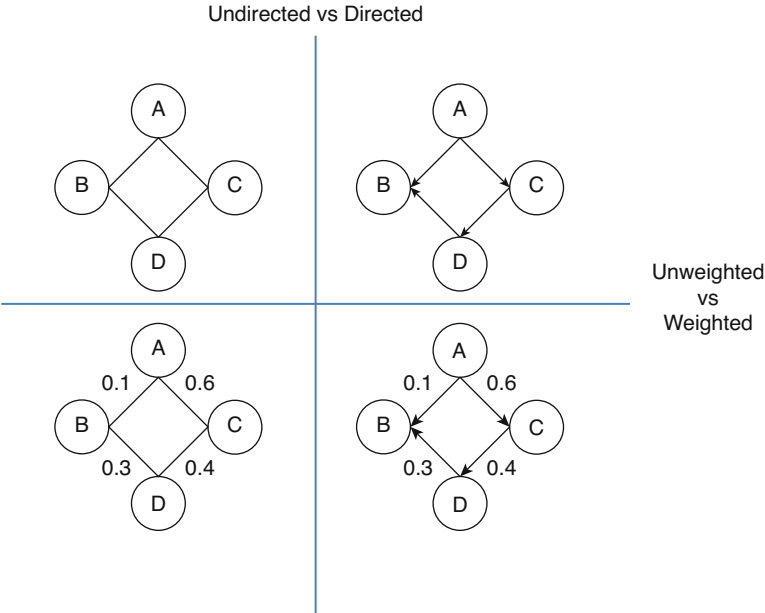
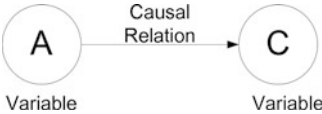


Fig. 6.4 Graph types

Fig. 6.5 Nodes and edges in Bayesian Networks



given as an arrow. In the graphical representation, node A is given as the conditional part, and node B is given as the causal part. The causal relation of the two attributes is defined by a table from this representation, and it will be described in detail in Sect. 6.4.2. When two attributes are independent of each other, there is no causal relation.

A little more complicated Bayesian Networks are illustrated in Fig. 6.6, as one more example. The five attributes, A, B, C, D, and E, are given as nodes. The attribute C is influenced by the attribute A, the attribute D is influenced by the attributes, A and B, and the attribute E is influenced by attribute E. The attributes, C and D, are influenced by the attribute A but are independent of each other; this case between the attributes, C and D, is called conditional independence [3]. The attributes, A and E, are independent ones, in interpreting the Bayesian Networks, but they are related with each other, indirectly, through the attribute, D.

Let us make some remarks on the Bayesian Networks that are presented as a graph. The causal relation as a unidirectional relation between two attributes is defined in the Bayesian Networks. The Bayesian Networks that are presented graphically and conceptually belong to the directed graph. It is assumed that two nodes that are the grand parent and the grandchild are independent ones. The nodes

Fig. 6.6 Interpretation of Bayesian Networks

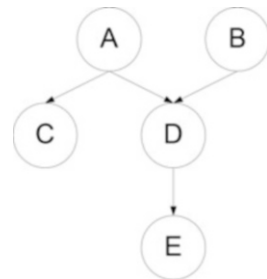
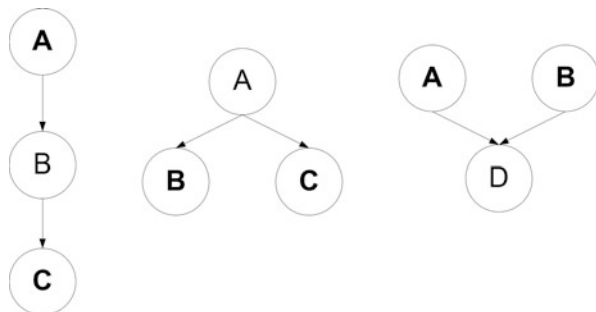


Fig. 6.7 Independent relation of two variables



that share their same parent and called siblings are conditionally independent of each other.

6.4.2 Causal Relation

This section is concerned with the causal relation among the variables. In Sect. 6.4.1, the causal relation between two variables is represented as an arrow in the Bayesian Networks. In this section, we try to express the causal relation as a table of two variables. In the table, the row corresponds to the variable in the causal part, and the column corresponds to the variable in the conditional part, and each cell indicates the conditional probability. This section is intended to express the causal relation that is viewed as the arrow in the Bayesian Networks as a tabular form.

The cases of independent variables in the Bayesian Networks are illustrated in Fig. 6.7. The variables, A and C, which are shown in the bold, are independent as the relation between the grand parent and the grandchild, as shown in the left part of Fig. 6.7. The variables, B and C, in the middle part of Fig. 6.7, are independent ones as the siblings in their child positions. The variables, A and B, in the right part of Fig. 6.7, are independent ones as the siblings in their parent positions. The case of the independent of the variables, B and C, in the middle part of Fig. 6.7, is called conditional independence.

The table is illustrated in Fig. 6.8, for presenting the causal relation in detail. In the causal relation, the variable, A, is the conditional part, and the variable, C,

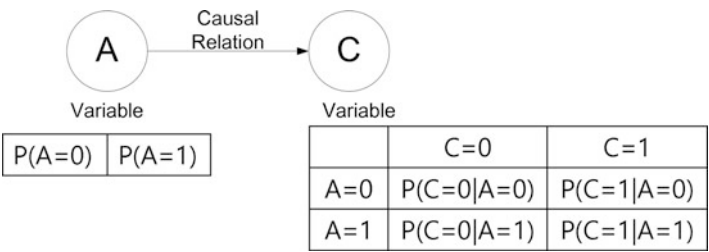


Fig. 6.8 Causal relation of two variables

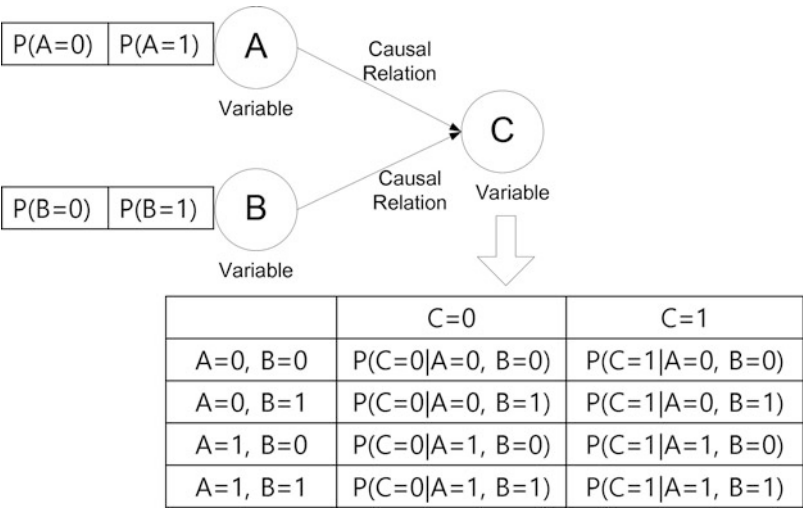


Fig. 6.9 Causal relation of three variables

is the causal part, assuming each variable accommodates a binary value. In the variable, A, the prior probabilities of the values are defined, and in the variable, C, the conditional probabilities are defined as a table. If the number of discrete values in the variable, A, is n , and the number of discrete values in the variable, C, is m , the table size becomes $n \times m$. It is required to discretize continuous values for defining the table, and the probabilities in it are decided through the learning.

The causal relation of the three variables is illustrated in Fig. 6.9. The variables, A and B, are given in the conditional part, and the variable, C, is given in the causal part; it is assumed that each variable has a binary value, zero or one. The primary probabilities of values are given in the variables, A and B, and the combination of values of both variables, A and B, is given for the conditional probabilities in the variable C. The variables, A and B, are independent of each other, and the variable C is dependent on the variables, A and B. We consider the probability table, in other types of relations of the three variables.

Let us make some remarks on the tabular form of the conditional probabilities; it represents the causal relation. The prior probabilities of the values are given in the variables that start the causal relation. The variation without its connection with other variables is independent of them. The variables that are given as siblings and share the same parent variable are independent of others; this case is called conditional independent variables. Each variable is assumed to have its discrete values in constructing the tabular form of the conditional probabilities in the causal relation; it is required to discretize the continuous value for doing it.

6.4.3 Learning Process

This section is concerned with the learning process of the Bayesian Networks. The causal relationships among the variables are represented graphically into the Bayesian Networks, and each causal relation is interpreted into a probability table. The Bayesian Networks learn the training examples with the two steps: the construction of the Bayesian Networks and the definition of the conditional probability table in each causal relation. The data items are classified in the Bayesian Learning, considering the causal relationships among the attributes. This section is intended to describe the learning process and the classification process of the Bayesian Learning.

The learning process of the Bayesian Networks for building the classification capacity and the regression capacity consists of the two steps. The first step is to construct the Bayesian Networks, and the second step is to construct the probability tables in each variable. All possible pairs are generated from the attribute set, and the support, which is the rate of the frequency of same values of two attributes to the total number of the training examples, and the confidence, which is the rate of the frequency of same value of the two attributes to the number of the same value of the conditional attribute, are computed. If both the support and the confidence are more than the thresholds, the two attributes are decided to have the causal relation. The Bayesian Networks are drawn by deciding the causal relation among the attributes from the training examples.

After building the Bayesian Networks, the probability table is defined for each node. By defining the causal relation to each pair of attributes, the Bayesian Networks are constructed. The prior probability is computed for each value in each initial attribute, which is isolated one or initially starting one by Eq. (6.39),

$$P(a_i = v_i) = \frac{freq(a_i = v_i)}{N} \quad (6.39)$$

The conditional probability, $P(a_i = v_i | a_{j1} = v_{j1}, \dots, a_{jk} = v_{jk})$, for each combination of values of the source variables and the variable, in the variable, a_i , is expressed by equation (6.40) which involves the source variables, $a_{j1}, a_{j2}, \dots, a_{jk}$,

$$P(a_i = v_i | a_{j1} = v_{j1}, \dots, a_{jk} = v_{jk}) = \frac{freq(a_i = v_i, a_{j1} = v_{j1}, \dots, a_{jk} = v_{jk})}{freq(a_{j1} = v_{j1}, \dots, a_{jk} = v_{jk})} \quad (6.40)$$

The probability table is defined for each variable in the Bayesian Networks by the training examples.

Let us mention the process of classifying a data item by the Bayesian Networks. In the Naive Bayes, the likelihoods in the input vector are computed by the product of ones of individual values, under the assumption of the independence among the attributes. The Bayesian Networks is constructed, and the conditional probabilities are defined in each variable, as the learning process from the training examples. If there is the causal relation between the two attributes, a_i and a_j , the probability of the two attributes is expressed in Eqs. (6.41) and (6.42),

$$P(a_i = v_i, a_j = v_j) \neq P(a_i = v_i)P(a_j = v_j) \quad (6.41)$$

$$\begin{aligned} P(a_i = v_i, a_j = v_j) &= P(a_i = v_i | a_j = v_j)P(a_j = v_j) \\ &= P(a_j = v_j | a_i = v_i)P(a_i = v_i) \end{aligned} \quad (6.42)$$

The conditional probability, $P(a_i = v_i | a_j = v_j)$ or $P(a_j = v_j | a_i = v_i)$, is obtained from the table that is defined inside the Bayesian Networks.

Let us point out some issues in training the Bayesian Networks by the training examples. There are two steps in the learning process: constructing the Bayesian Networks and defining the probability table in each node. The indirect causal relation between two variables is treated as the independent relation. The Bayesian Networks are mapped into the Naive Bayes, by removing the attributes that are correlated strongly with each other. The Bayesian Networks may be expanded into the Markov Networks, considering the correlation relation between attributes, rather than the causal relation [3].

6.4.4 Comparisons

Table 6.1 illustrates the comparison of the three classifiers: the Bayes classifier, the Naive Bayes, and the Bayesian Networks. In this chapter, we studied the three classification algorithms as the probabilistic learning algorithms. They are compared with the respect to their differences as presented in Table 6.1. We may consider the probabilistic algorithms with mixture of the three models, by the comparison. This section is intended to state the differences of the three kinds of the probabilistic algorithm for each agenda in Table 6.1.

As shown in the second row in Table 6.1, the likelihoods of atomic entities are explored in the three kinds of the probabilistic learning algorithm. In the

Table 6.1 Bayes classifier vs. Naive Bayes vs. Bayesian Networks

	Bayes classifier	Naive Bayes	Bayesian Networks
Likelihood	$P(\mathbf{x} C)$	$\prod_{i=1}^{i=d} P(x_i C)$	$P(x_1, \dots, x_d C)$
Class distribution	Gaussian distribution	Not assumed	Not assumed
Attribute relation	Compound	Independence	Causality
Complexity	Low	Medium	High

Bayes classifier, the input vector is set as a likelihood entity. In the Naive Bayes, an individual attribute value becomes a likelihood entity that is assumed as the independent of other attribute values. A single attribute value or more than one attribute value is a likelihood entity in the Bayesian Learning, following the Bayesian Networks. The process of classifying a novice item by computing its likelihoods to the categories is common in the kinds of classifiers.

In the third row of Table 6.1, the difference of the three kinds of the probabilistic learning algorithms with respect to their likelihoods is presented. In the Bayes classifier, the likelihood of the given example, itself, is based on the probability of the vector in the Gaussian distribution. In the Naive Bayes, the product of the likelihoods of individual values to a category is used as the likelihood of the example. In the Bayesian Learning, the likelihood of an item is decided by the relations among the attributes in the Bayesian Networks. Because each item is classified into the category that corresponds to the maximum likelihood, the three kinds of the probabilistic algorithm that are involved in the comparison belong to the maximum likelihood learning.

Let us consider the process of estimating the likelihoods in the three kinds of the probabilistic learning. In the Bayes classifier, the parameters of the Gaussian distribution, the mean vectors and the covariance matrix, are estimated for each category. The learning process of the Naive Bayes is to estimate the likelihoods of individual attribute values to each category. The learning process of the Bayesian Learning is to define the causal relations among the attributes and construct the conditional probability table in each relation. In this kind of the learning algorithms, the items are classified by their category likelihoods.

Let us make some remarks on the three types of the probabilistic learning algorithms: the Bayes classifier, the Naive Bayes, and the Bayesian Learning. In the simplified Bayes Classifier which omits the covariance matrix, assuming the Gaussian distribution, considers the distance from the mean vectors or the similarity as them. In the Naive Bayes, the independences among the attributes are assumed in computing the likelihoods of the input vector to the predefined categories. The learning process of the Bayesian Learning is to construct the Bayesian Networks and define the probability table in each variable.

6.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. The Bayes classifier where each category is assumed as a Gaussian distribution is the first type of the probabilistic learning. The Naive Bayes is the probabilistic learning where the likelihood of a novice item is computed by product of likelihoods of its individual attribute values under the assumption of independences of attributes. The advanced probabilistic learning is the Bayesian Learning that builds the Bayesian Networks and defines the conditional probability table for each variable. In this chapter, we studied the process of classifying the novice items by the likelihoods to the categories in the three kinds of probabilistic learning.

The discretization of each attribute value is required for using the Naive Bayes or the Bayesian Learning. Because it is possible to compute the similarity with the mean vector without discretizing values, the discretization is not required in using the Bayes classifier. Because it is necessary to compute likelihoods of individual attribute values to each category, it is necessary to discretize continuous attribute values in using the Naive Bayes. Because conditional probability table should be defined in each node in the Bayesian Networks, it is also necessary to discretize the continuous attribute values in using the Bayesian Learning. The issue in using the Naive Bayes and the Bayesian Learning is how to decide the interval size in discretizing attribute values.

Let us consider the probabilistic learning for the regression task as well as the classification task. In the classification task, the conditional probabilities of the categories, $c_1, c_2, \dots, c_k$, given the novice example, $\mathbf{x}$, $P(c_1|\mathbf{x})$, $P(c_2|\mathbf{x})$, $\dots$, $P(c_k|\mathbf{x})$, are computed. In the regression task, the conditional probability of estimated output value given the novice example, $P(\hat{y}|\mathbf{x})$, is computed. The output value is decided by the maximal likelihood, $\hat{y} = \operatorname{argmax}_y P(\mathbf{x}|y)$, because of $P(y|\mathbf{x}) \propto P(\mathbf{x}|y)$. The regression is solved by mapping the regression into the classification by discretizing the output value.

The fuzzy concepts may be introduced in using the Bayes classifier as a classification algorithm. The fuzzy distribution such as the triangle distribution and the trapezoid distribution is defined for each category. The parameters of the Gaussian distribution, the mean vector and the covariance matrix, are treated as a fuzzy vector or a fuzzy matrix. After defining the Gaussian distribution for each category, category membership values are assigned to each training example as it label. We may derive various variants from the Bayes classifier by planting the fuzzy concepts on the algorithm.

Let us consider using the Naive Bayes for implementing the semi-supervised learning. It refers to the learning type where the unlabeled examples are used as well as the labeled examples for training the machine learning algorithms for improving their classification or regression performances. The categories are assigned to the unlabeled examples based on their similarities with the labeled ones. We may consider modifying the Naive Bayes into the unsupervised version, before doing

it into its semi-supervised version. We present the case of using the Naive Bayes by combining it with the EM algorithm in Chap. 11.

References

1. T. Mitchell, *Machine Learning* (McGraw Hill, New York, 1997)
2. J.J. Buckley, *Fuzzy Probabilities* (Springer, Berlin, 2003)
3. D. Koller, N. Friedman, *Probabilistic Graphical Models: Principles and Techniques* (The MIT Press, Cambridge, 2009)
4. H. Pishro-Nik, *Introduction to Probability, Statistics, and Random Process* (Kappa Research, Sunderland, 2014)

Chapter 7

Decision Tree



7.1 Introduction

The decision tree is the tree for classifying items following branches from its root node. Its terminal nodes are given as one of the predefined categories, and the others are given as attributes in the decision tree. Each branch from a node corresponds to a value or a value interval of the corresponding attribute. Learning of decision tree is the process of building itself from the training examples. This section is intended to describe the structure, the construction, and the pruning of the decision tree for understanding the classification process and the learning process.

Let us view the structure of the decision which is used for classifying an item. Its nodes indicate attributes or categories, and its branches mean attribute values or attribute value intervals, in a decision tree. Among the nodes, each terminal node indicates a category and each non-terminal node indicates an attribute. The branches from the root node to a terminal node become the path for classifying an item. The decision tree should be constructed using the training examples before classifying an item following its branches.

The decision tree is constructed as the learning process from the training examples. It is assumed that the problem is given as a binary classification, and the terminal node indicates the positive class or the negative class. One among the attributes is selected as the root node based on the information theory. The branches from an attribute are defined to child nodes or terminal nodes. The process of constructing the decision tree will be explained in detail in Sect. 7.3.

The pruning after constructing the decision tree is to cut off some attributes and branches. When a big decision tree that classifies training examples almost completely correctly is constructed, it cannot avoid the overfitting where the training examples are classified very well, but novice ones are classified very poorly. We need to prepare some examples that are separated from the training examples, called validation set, and observe the performance of classifying them. As the downsizing process, some attributes and branches are cut off from the decision tree, in order to maximize the classification performance of the validation set. The construction

and the pruning of the decision tree are for optimizing the trade-off between the complexity and the performance.

The goal of this chapter is to understand the decision tree as a symbolic learning algorithm. We understand the decision tree structure and the process of classifying data items. We need to understand the process of constructing the decision tree from the training examples as its learning process, together with its pruning process. We will also understand some variants that are derived from the decision tree with respect to their differences from the standard version. We will make the further discussions for deriving more advanced decision trees and hybrid learning algorithms that are mixed with other types.

7.2 Classification Process

This section is concerned with the process of classifying items by the decision tree. In Sect. 7.2.1, we explain the decision tree structure and present the toy example in Sect. 7.2.2. In Sect. 7.2.3, we mention the application of the decision tree to the text classification task. In Sect. 7.2.4, we describe the process of extracting symbolic rules from the decision tree, in order to prove that the decision tree is characterized as a symbolic learning algorithm. This section is intended to explain the process of classifying items by the decision tree, and the next one is intended to explain the process of constructing the decision tree from the training examples.

7.2.1 Basic Structure

This section is concerned with the basic structure of the decision tree which we used for classifying an item. Each node in the decision tree indicates the attribute of a numerical vector or categories. Each edge indicates a value or a value interval of attribute which corresponds to the source node. Each item is classified following edges from the root node to a terminal node. This section is intended to describe the nodes and edges of the decision tree for understanding its basic structure.

The attributes of the training examples and the nodes of the decision tree are illustrated in Fig. 7.1. Each data item is represented as a d dimensional numerical vector with the d attributes, $A_1, A_2, \dots, A_d$, and the given problem is assumed to be a binary classification where each item is classified into the positive class, $+$, or the negative class, $-$. The non-terminal nodes including the root node indicate attributes. The terminal nodes are given as either of the two categories. In each node, an attribute is determined by the information gain which reflects the distribution over the two categories.

Edges in the decision tree in both discrete attribute values and continuous ones are illustrated in Fig. 7.2. As shown in the left side of Fig. 7.2, if an attribute value is discrete, each edge corresponds to an individual value of the attribute, A_{ij} , viewed

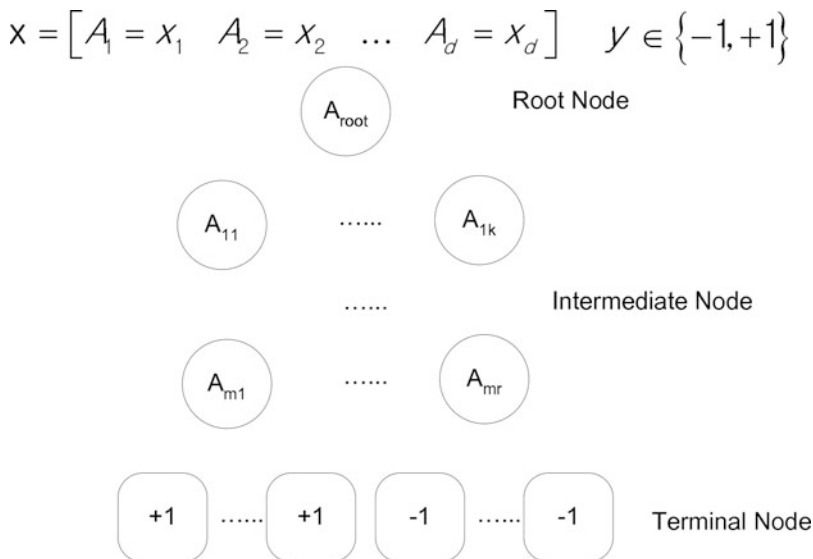


Fig. 7.1 Nodes of decision tree

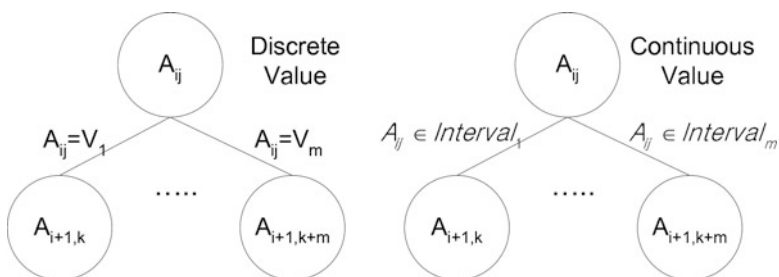


Fig. 7.2 Edges of decision tree

in the parent node. As shown in the right side of Fig. 7.2, if an attribute value is continuous, each edge corresponds to a value interval of the attribute, A_{ij} . When too many discrete values are given in the attribute, it is necessary to set each edge as a value interval. The child nodes that are connected with the edges from the parent node are categories or other attributes.

The process of classifying item by the decision tree is presented as the pseudo code in Fig. 7.3. The root node is extracted from the decision tree by calling the method, `getRootNode`, and the method, `classifyByNode`, is called with the argument, `rootNode`, in calling the initial function, `classifyByDecisionTree`. If the current node is terminal node, its label is returned as the output, and otherwise, the branch is selected for deciding its child node, and the method, `classifyByNode`, is called recursively with the argument, `childNode`. The function, `selectBranch`, returns

```

classifyByDecisionTree(Item example){
    rootNode = decisionTree.getRootNode();
    classifyByNode(rootNode, example);
}

classifyByNode(Node currentNode, Item example){
    if(isTerminal(currentNode)){
        categoryName = currentNode.getCategoryName();
        return categoryName;
    }
    branch = selectBranch(currentNode, example);
    childNode = (currentNode, branch);
    return classifyByNode(childNode, example);
}

selectBranch(Node currentNode, Item example){
    attributeValue = example.getAttributeValue(currentNode);
    branchList = currentNode.getBranchList();
    for each branch in branchList{
        if(branch.isMatch(attributeValue))
            return branch
    }
}

```

Fig. 7.3 Classification process in pseudo code

the branch that corresponds to the attribute value. The item is classified by calling the method, `classifyByNode`, recursively from the root node, following the branches.

Let us make some remarks on the basic structure of the decision tree which is mentioned above. The group of nodes in the node is divided into the two groups: the terminal nodes that indicate the predefined categories and the non-terminal nodes that indicate the attributes. The branches in the decision tree indicate the values or value intervals of the attribute corresponding to the parent node. An item is classified by exploring from the root node to the terminal node, following the branches. Selecting a branch is to search for the matching one to the corresponding value or such value interval.

7.2.2 Toy Examples

This section is concerned with the toy examples that are presented for understanding the classification process in the decision tree. It is assumed that the given problem is a binary classification where the positive class and the negative class are available as the predefined categories. We mention the three toy examples: the binary

Fig. 7.4 Binary classification with single variable

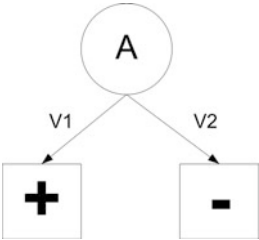
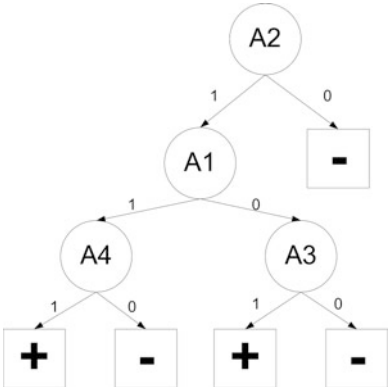


Fig. 7.5 Binary classification with multiple variables



classification with a single variable, one with multiple variables, and the multiple classification with a single variable. We present the three simple decision trees that correspond to the three toy examples. This section is intended to mention the three simple cases of the decision tree on toy examples for understanding the execution of the decision tree, easily.

The simplest decision tree for the binary classification by a single attribute is illustrated in Fig. 7.4. In Fig. 7.4, a root node is given as the attribute, and the two terminal nodes are given as the two categories, the positive class and the negative class. In Fig. 7.4, if a data item has its value, $v1$ in its attribute A , the data item is classified into the positive class, following the left branch, and if the attribute A , is the value $v2$, the item is classified into the negative class, following the right branch. The two edges in the decision tree that is shown in Fig. 7.4 correspond, respectively, to values, $v1$ and $v2$. The data item, with $A \neq v1$ and $A \neq v2$, is rejected in this case.

The binary classification by the decision tree with multiple attributes is illustrated in Fig. 7.5. In this task, it is assumed that the four attributes, A_1 , A_2 , A_3 , and A_4 , are given, and the binary value, 0 or 1, is assigned to each attribute. Classifying the item proceeds in the direction, A_2 , A_1 , A_3 , or A_4 . The item with the attribute values, $A_1 = 1$, $A_2 = 1$, $A_3 = 0$, and $A_4 = 1$, is classified as the positive class, by the classification path, $A_2 \rightarrow A_1 \rightarrow A_3$, for example. If $A_2 = 0$, the item is classified as the negative class, absolutely, in the examples, which is shown in Fig. 7.5.

Fig. 7.6 Multiple classification with single variable

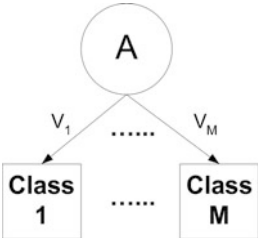
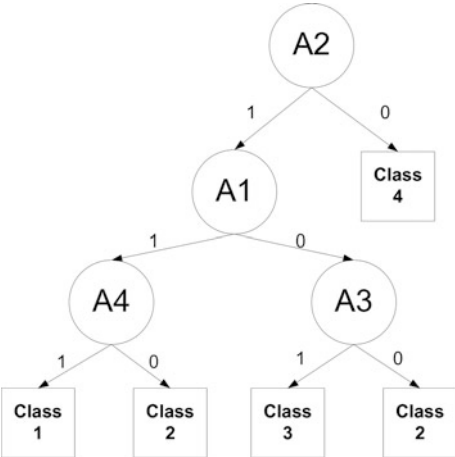


Fig. 7.7 Multiple classification with multiple variables



The multiple classification based on a single attribute by the decision tree is illustrated in Fig. 7.6. It is assumed that the task is to classify each item to one of M categories. The root node is given as the attribute, each branch from the root node corresponds to a value or a value interval in the attribute, A , and the child nodes correspond to the M categories. The branch from the root node which corresponds to the attribute value or the attribute value interval is selected for reaching a terminal node. In the soft classification, the overlap between categories of attribute values may be allowed.

The case of classifying each item with its multiple attributes to one of more than two categories is shown in Fig. 7.7. Each item is represented with the four attribute values, A_1, A_2, A_3, A_4 , each attribute value is given as a binary value, 0 or 1, and the four categories, class 1, class 2, class 3, and class4, are predefined. Each item is classified by the direction, $A_2 \rightarrow A_1 \rightarrow A_3$ or A_4 . For example, the vector, $[1\ 1\ 0\ 1]$, is classified as class 1, by $A_2 = 1 \rightarrow A_1 = 1 \rightarrow A_4 = 1 \rightarrow$ class 1. The item with $A_2 = 0$ is classified as class 4, absolutely.

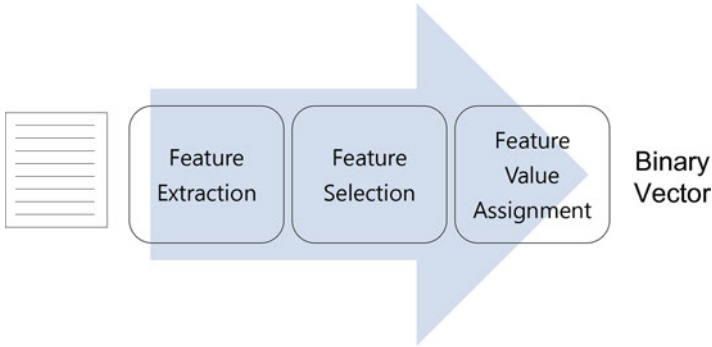


Fig. 7.8 Text representation

7.2.3 Text Classification

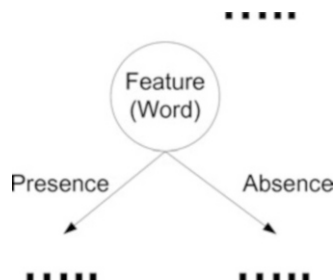
This section is concerned with the application of the decision tree to the text classification task. It is defined as the process of assigning one or some among the predefined categories to each item. A text that is written in a natural language is encoded into a numerical vector, and the decision tree is applied to the task. The attributes of a numerical vector that represents a text are words, so the non-terminal nodes indicate words and the terminal nodes indicate the predefined categories. This section is intended to describe the process of applying the decision tree to the text classification.

The process of encoding a text into a numerical vector is illustrated in Fig. 7.8. Words are generated by indexing a text collection as feature candidates in the first step. Some among the feature candidates are selected as features by a criterion such as their coverage in the collection. For each feature, a value that indicates its relationship with the text is assigned. In applying the decision tree to the text classification, it is assigned that a feature value is given as a binary value, 0 as the absence in the text and 1 as the presence.

A particular non-terminal node and its branches in the decision tree for the text categorization are shown in Fig. 7.9. The features of numerical vectors which represent texts are given as words. The non-terminal nodes are given as words, and their branches are given as the binary values, zero and one. The right branch is the case where the corresponding word appears in the text, and the left branch is one where the word does not appear. The terminal nodes stand for the categories or topics that are predefined in advance.

The process of classifying a text by the decision tree is illustrated in Fig. 7.10. It is assumed that the decision tree is constructed by sample texts, and a novice text is given as the input and encoded into a binary numerical vector whose features are words. If a feature value that corresponds to the current node is one, the right branch is selected and the given function is called recursively, and otherwise, the left branch is selected and it is called recursively. If the current node is a terminal

Fig. 7.9 Application of decision tree to text classification



```

classifyTextByDecisionTree(Item textRepresentation){
    Node = decisionTree.getRootNode();
    classifyByNode(rootNode, textRepresentation);
}

classifyTextByNode(Node currentNode, Item example){
    if(isTerminal(currentNode)){
        label = currentNode.getLabel();
        return label;
    }
    feature = currentNode.getFeature();
    if(textRepresentation.getFeature() == 1)
        return classifyByNode(rightChildNode, example);
    return classifyByNode(leftChildNode, example);
}

```

Fig. 7.10 Classification process in pseudo code

node, the category is returned as its label. The process of classifying items by the decision tree is implemented by the recursive call of the function.

Let us make some remarks on applying the decision tree to the text classification. In this application area, some words in a text collection are used as features for representing a text as a numerical vector. It is assumed that all feature values are given as binary values; a text is encoded into a binary vector. Only two possible values in each attribute become the reason of only two branches from each non-terminal node. If the feature value is given as a continuous value such as TF-IDF (Term Frequency–Inverse Document Frequency) weight, it should be discretized.

7.2.4 Rule Extraction

This section is concerned with the process of extracting symbolic rules from the decision tree. In the previous sections, we studied the process of classifying a data

Fig. 7.11 Rule extraction in single variable binary classification

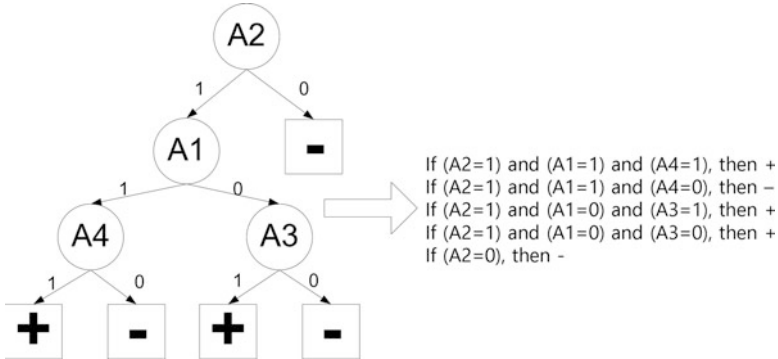
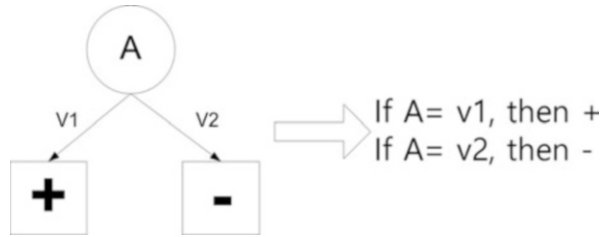


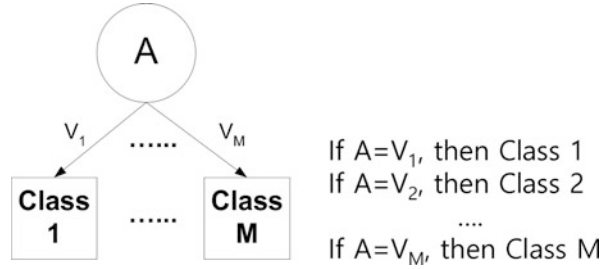
Fig. 7.12 Rule extraction in multiple variable binary classification

item by the decision tree, following the path. The symbolic rules are extracted from the decision tree, after classifying a data item, as the evidence why it is classified so. The merit of the decision tree is the possibility of providing the symbolic reasons of classifying data items. This section is intended to describe the process of extracting symbolic rules as the classification evidence.

The process of extracting symbolic rules from the decision tree for one variable binary classification is illustrated in Fig. 7.11. In the decision tree, a single attribute and the two values in it are assumed as shown in Fig. 7.11. The rule, “if $A = v_1$, then positive class,” is extracted, following the left branch, and the rule, “if $A = v_2$, then positive class,” is extracted, following the right branch. In classifying the item as the positive class, the rule, if $A = v_1$, then +, is proposed as the evidence. The decision tree that is presented in Fig. 7.11 is simplest.

The symbolic rules and the decision tree in the case of the binary classification with its multiple variables are illustrated in Fig. 7.12. The four attributes, A_1 , A_2 , A_3 , and A_4 , are given, and the positive class and the negative class are given as the predefined categories in this problem. The path from the root node to the terminal node is defined as a symbolic rule; in the if-then rule, the path indicates the conditional part, and the terminal node that is reached by the path indicates the causal part. For example, the leftmost path from A_2 to A_4 indicates the conditional part, $A_2=1$, $A_2=1$, and $A_4=1$, and the leftmost terminate indicates the causal part, “then +”; the symbolic rule is expressed as “if $A_2=1$ and $A_1=1$ and $A_4=1$, then +.” The vector $[1 \ 1 \ * \ 1]$ is classified as the positive class by the symbolic rule.

Fig. 7.13 Rule extraction in single variable multiple classification



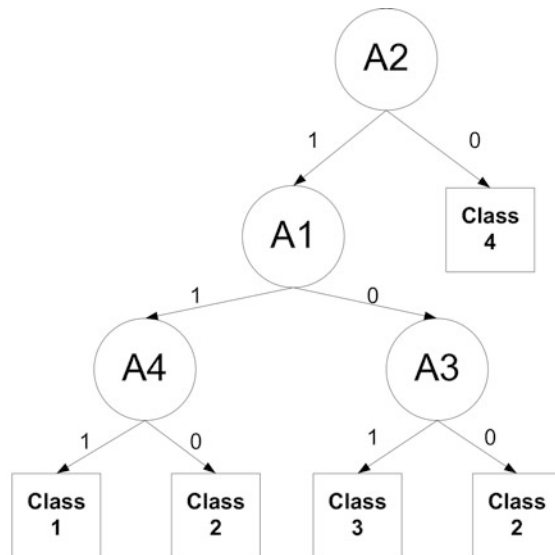
The symbolic rules that are extracted from the decision tree for the multiple classification with a single attribute are illustrated in Fig. 7.13. A given problem is specified as the process of classifying each item to one of M categories with an attribute. Class 1, class 2, ..., class M are the causal parts, and each class corresponds to an edge. For example, class 1 is connected with the attribute as its parent node, through the edge which is labeled with v_1 ; the symbolic rule is extracted as if $A = v_1$, then Class 1. The one to one matching between the attribute value and the category exists in the decision tree.

The symbolic rules from the more complicated decision tree for the multiple classification with the multiple attributes are illustrated in Fig. 7.14. Each item is classified to one of the four categories with its four attributes. The branches in the path from the root node to the terminal node belong to the conditional part and the terminal node that is reached by the path is the causal part. For example, following the leftmost path $A_2 \rightarrow A_1 \rightarrow A_4 \rightarrow$ Class 1, the rule, if $(A_2=1)$ and $(A_1=1)$ and $(A_4=1)$, then Class 1, is extracted. The decision tree that is presented in Fig. 7.14 is applicable to only hard classification where each item is classified to only one category.

7.3 Learning Process

This section is concerned with the process of constructing the decision tree from the training examples as its learning process. In Sect. 7.3.1, we mention the preprocessing including the attribute value discretization as the requirement for building the decision tree. In Sects. 7.3.2 and 7.3.3, we explain the process of selecting attributes as the root node and the interior nodes, respectively. In Sect. 7.3.4, we describe the pruning that cuts off branches of the decision tree for improving its generalization performance. This section is intended to describe the process of constructing the decision tree using the training examples.

Fig. 7.14 Rule extraction in multiple variable multiple classification



If (A2=1) and (A1=1) and (A4=1), then Class 1
 If (A2=1) and (A1=1) and (A4=0), then Class 2
 If (A2=1) and (A1=0) and (A3=1), then Class 3
 If (A2=1) and (A1=0) and (A3=0), then Class 2
 If (A2=0), then Class 4

7.3.1 Preprocessing

This section is concerned with the preprocessing of data items for applying the decision tree to the classification task. The discretization of attribute values is required for building the decision tree; a continuous attribute value causes an infinite number of branches. The optimal number of value intervals is necessary for its reliability and its efficiency; many value intervals cause the good reliability but the poor efficiency, and a few value intervals cause the good efficiency but the poor reliability. The demerit of decision tree is the discretization that results in the information loss, but the merit is the possibility of extracting symbolic rules for tracing the classification. This section is intended to describe the preprocessing of data items as the preparation for using the decision tree for the classification task.

We need to remove the noises from the numerical vectors for constructing the decision tree as the learning process. The decision tree is very fragile to the noises of input data, compared with other supervised learning algorithms. The attribute values of the training examples and ones with non-sensible values should be removed. The values that violate against the constraints or the integrity are regarded as noises; for example, the continuous value is given to the attribute whose value should be

discrete and the value that is scaled output of the range belongs to noises. The unreliable values may be corrected by estimating them rather than removing them.

It is required to discretize continuous attribute values for using the decision tree for the classification task. The discretization refers to the process of mapping an entire continuous value range into a finite number of values. In the entire range, the maximum and the minimum are selected, and the several value intervals between them, called bins, are defined. The number of bins in the attribute value range and each bin size become the issues for discretizing the continuous value range. The process of matching each string to a numerical value is called codification.

Let us consider missing values among attribute values in the training examples in constructing the decision tree. The issue is mentioned in Sect. 3.2.3 as one of encoding relational data into numerical vectors. Because the training examples are treated attribute by attribute in the decision tree, it is possible to use the numerical vector with some missing values for constructing the decision tree. If an attribute has any missing value, it is excluded in constructing a node and its branches. The decision tree is constructed based on the columns under the assumption that the training examples are composed of a table where its rows are examples and its columns are attributes.

Let us consider putting the privacy protection on some attributes. In other words, some attributes are public and the others are private. The attributes that are protected as private ones are not used for building the decision tree. If such attributes are essential for classifying items, it influences on the classification performance. We need to do the research for improving the classification performance, preserving the privacy.

7.3.2 *Root Node*

This section is concerned with the process of constructing the root node and its branches of the decision tree. It is assumed that the problem is given as a binary classification and the training examples are labeled with the positive class or the negative class. The distribution over categories on attribute values is observed for each attribute, and the attribute with the most uneven distribution is selected as the root node. The information gain is used for measuring the attributes quantitatively. This section is intended to explain the process of selecting an attribute as the root node of the decision tree.

Let us mention the assumption before explaining the process of selecting an attribute as the root node. Each data item is represented into a d dimensional numerical vector, and each element is given as a binary value, 0 or 1. Each non-terminal node that indicates an attribute has two branches that indicates 0 or 1. The given problem is assumed to be a binary classification, and each training example is labeled with the positive class and the negative class. The task in constructing the decision tree is assumed to classify each binary vector as the positive class or the negative class.

Let us mention the process of computing the criteria for selecting the root node among the attributes. It is assumed that each attribute value is given as 0 or 1, and the given problem is a binary classification. When the positive class or the negative class are completely balanced with each other, the entropy is 1. For each attribute, the two entropies are considered; one is $-p_{a_i=0}(p_{+(a_i=0)}\log_2 p_{+(a_i=0)})$ where $p_{a_i=0}$ is the portion of the training examples where $a_i = 0$ and $p_{+(a_i=0)}$ is the portion of the training examples which are labeled with the positive class where $a_i = 0$, and the other is $-p_{a_i=1}(p_{+(a_i=1)}\log_2 p_{+(a_i=1)})$, where $p_{a_i=1}$ is the portion of the training examples where $a_i = 1$, $p_{+(a_i=1)}$ is the portion of the training examples which are labeled with the positive class where $a_i = 1$. For each attribute, the information gain is computed by Eq. (7.1),

$$IG(a_i) = 1.0 - p_{a_i=0}(p_{+(a_i=0)}\log_2 p_{+(a_i=0)}) - p_{a_i=1}(p_{+(a_i=1)}\log_2 p_{+(a_i=1)}) \quad (7.1)$$

and the attribute with its maximum information gain is selected as the root node. Since the relations between portions of the positive class and the negative class in each attribute value are expressed into Eqs. (7.2) and (7.3),

$$p_{-(a_i=0)} = 1.0 - p_{+(a_i=0)} \quad (7.2)$$

$$p_{-(a_i=1)} = 1.0 - p_{+(a_i=1)} \quad (7.3)$$

the symmetry distribution of entropy in 0.5 as the center is expressed into Eqs. (7.4) and (7.5),

$$-p_{-(a_i=0)}\log_2 p_{-(a_i=0)} = -(1.0 - p_{+(a_i=0)})\log(1.0 - p_{+(a_i=0)}) \quad (7.4)$$

$$-p_{-(a_i=1)}\log_2 p_{-(a_i=1)} = -(1.0 - p_{+(a_i=1)})\log(1.0 - p_{+(a_i=1)}) \quad (7.5)$$

and the entropies to the negative class are computed by Eqs. (7.6) and (7.7)

$$-p_{a_i=0}(p_{-(a_i=0)}\log_2 p_{-(a_i=0)}) = -p_{a_i=0}(p_{+(a_i=0)}\log_2 p_{+(a_i=0)}) \quad (7.6)$$

$$-p_{a_i=1}(p_{-(a_i=1)}\log_2 p_{-(a_i=1)}) = -p_{a_i=1}(p_{+(a_i=1)}\log_2 p_{+(a_i=1)}) \quad (7.7)$$

Let us consider the process of constructing the decision tree for the multiple classification where each item is classified to one of more than two categories. It is assumed that all attribute values are discretized or discrete initially; the number of values in each attribute is finite. The dominance of a particular category over the others is the criteria for selecting one among the attributes as the root node; the dominance is computed for each value in each attribute, and the dominance values are averted as the dominance of the given attribute. The attribute with the maximum dominance of the category is selected as the root node, and we need to define the quantitative measure of the dominance. The entropy and the information gain are applied by decomposing the multiple classification into binary classifications.

Let us make some remarks on the definition of the root node and its branches in constructing the decision tree. The first step of constructing the decision tree from the training examples is to decide one among the attributes as the root node. It is assumed that all attribute values are discrete, and it is required to discretize continuous values. The unbalanced distribution over the categories in the attribute value, corresponding to a branch, is the criteria for selecting the root node. If almost all of the training examples with the given attribute value belong to a particular category, the child node that is connected to the branch becomes a terminal node.

7.3.3 Interior Nodes

This section is concerned with the process of generating the interior nodes and their branches. In Sect. 7.3.2, we described the process of generating the root node and its branches, in constructing the decision tree. A non-terminal node is decided among the attributes except the one that corresponds to the root node, by the process that is described in Sect. 7.3.2, recursively. Until almost of the training examples are classified correctly, the generation of an interior node is iterated. This section is intended to explain the process of constructing the interior node in the second, the third, and the n th level.

Let us consider the intermediate node in the second level after deciding the root node. The set of training examples that correspond to the edge from the root node, $a_r = v$, is notated by $Tr_{a_r=v}$ and is divided into the positive set, $Tr_{(a_r=v)+}$, and the negative set, $Tr_{(a_r=v)-}$, assuming the given problem as a binary classification. The information gains of the attributes except one which corresponds to the root node, are computed by the process that is described in Sect. 7.3.2. The attribute with the maximum information gain is selected as the intermediate node in this level. The node that is built by the process is the child node of the root node which is connected down by the edge that corresponds to a particular value of the attribute which correspond to the root node.

In the third level, let us try to generate the intermediate node by selecting an attribute from the node in the second level, which is notated by a_p . It is assumed that the node, a_p , is connected by the edge, $a_r = v_1$, to the root node, a_r , and the attribute, a_c , is decided as the node in the third level in the edge, $a_p = v_2$, from the node, a_p . The subset of the training set, Tr , which consists of the training examples with two attributes, $a_r = v_1$ and $a_p = v_2$, is notated by $Tr_{a_r=v_1 \wedge a_p=v_2}$, and the information gain is computed for each attribute, except a_r and a_p .

Let us mention the process of selecting an attribute in the k level, as the general case. The edges from the root node to the parent node in the $k - 1$ level, a_{k-1} , are expressed as Eq. (7.8),

$$\bigwedge_{i=1}^{k-1} (a_i = v_i) \quad (7.8)$$

The edge from the parent node is assumed as $a_k = v_k$, and the subset of the training set is notated as $Tr_{\wedge_{i=1}^{k-1}(a_i=v_i)}$. The information gain of the attributes, except the ones that are selected as non-terminal nodes, is computed, and the attribute with its maximal information is selected by the process that is described in Sect. 7.3.2. When almost all of the examples in the set, $Tr_{\wedge_{i=1}^{k-1}(a_i=v_i)}$ belong to the positive class or the negative class, the terminal node which indicates either of the two classes becomes the child node.

Let us make some remarks on the process of constructing the interior nodes and their branches in the decision tree. The process of learning the training examples is implemented recursively. The entire set of the training examples and the entire set of attributes are initially prepared, and when almost of the training examples belong to a category, it is terminated by labeling the terminal node as the category. The training set and the attributes are updated into the subset satisfying the conjunction form, $\wedge_{i=1}^{k-1}(a_i = v_i)$ and the attributes, except $a_1, a_2, \dots, a_k$, subsequently. It is necessary to allow misclassification on the training examples in constructing the decision tree, in order to prevent the overgrowth of the decision tree which causes the overfitting.

7.3.4 Pruning

This section is concerned with the process of cutting the decision tree down for improving its generalization performance. Until now, we have studied the process of constructing the decision tree from the training examples. Some examples are taken from the training set as the validation ones, and the branches are cut off based on the misclassification on the validation examples. As an alternative way, branching from an interior node is terminated, allowing some misclassifications. This section is intended to describe the two ways of optimizing the decision tree size, together with the validation set.

The cross validation is introduced for pruning the decision tree more reliably, instead of using the separated validation set. The limits of using a single validation set are the reduction of number of the training examples as the cause of the overfitting and the strong biased utilization of the training examples. The training set is partitioned into n subsets, and $n - 1$ subsets are used as the training set and one subset is used as the validation set. The iteration of using so from the first subset to the last one, partition by partition, is called cross validation. It may be used for tuning the parameters of any machine learning algorithm, as well as pruning the decision tree.

Let us mention the first scheme of constructing the decision tree, avoiding the overfitting. In this scheme, whenever the decision tree grows, we observe the classification performance, using the validation set. When the misclassification is increased in the validation set or through the cross validation, the category node is connected to its parent node directly, instead of adding the interior node. The

decision tree grows until the misclassification rate is increased. This scheme is characterized as the prudent growth of the decision tree, rather than pruning.

We may consider constructing the complete decision tree and downsizing it by removing branches. We mentioned the process of constructing the branches, observing the classification performance on the validation set, or through the cross validation. As the branches are removed gradually, the classification error on the validation set is reduced and increased; the pruning is stopped in the turning point. The decision tree is constructed from the training examples, allowing the slight classification error on the training set, as an alternative way of the scheme.

Let us make some remarks on pruning the decision tree into its optimal size. The pruning is defined as the process of cutting off some branches after constructing the complete decision tree, in the context of the decision tree. If the branches are constructed very prudently by the cross validation, allowing some training error, the pruning is not needed. The complete decision tree classifies almost of the training examples correctly but is very poor to the novice examples by the overfitting. In pruning the decision tree, we consider the trade-off between the complexity as the decision tree size and the performance as the classification error.

7.4 Variants

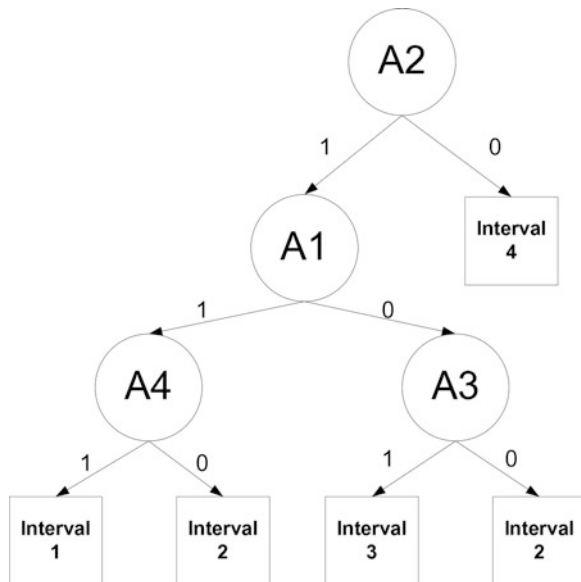
This section is concerned with the variants of decision tree and similar algorithms with the decision tree. In Sect. 7.4.1, we mention the version of decision tree which is used for the regression tasks. In Sect. 7.4.2, we study the decision list where decision rules are given as a list. In Sect. 7.4.3, we describe the random forest that is given as multiple decision trees for the classification tasks. In Sect. 7.4.4, we consider the decision graph where siblings are connected with each other.

7.4.1 Regression Version

This section is concerned with the modification of the decision tree for applying it to the regression tasks. The difference of the regression from the classification is that its output value is continuous. The regression may be mapped into the classification by discretizing the output value. The decision of the number of intervals is the important issue in discretizing it. This section is intended to mention the scheme of applying the decision tree to the regression.

In using the decision tree for the regression task, we need to discretize the continuous output value. For first, we decide the maximum as the upper bound, y_{max} , and the minimum as the lower bound, y_{min} , of the output value. The number of intervals is b , and the intervals of the output value are given as Eq. (7.9),

Fig. 7.15 Decision tree structure in regression



$$\begin{aligned}
 & \left[y_{\min}, y_{\min} + \frac{y_{\max} - y_{\min}}{b} \right] \\
 & \left[y_{\min} + \frac{y_{\max} - y_{\min}}{b}, y_{\min} + 2\frac{y_{\max} - y_{\min}}{b} \right] \\
 & \dots \\
 & \left[y_{\max} - \frac{y_{\max} - y_{\min}}{b}, y_{\max} \right]
 \end{aligned} \tag{7.9}$$

Each interval is expressed as Eq. (7.10) in the general form,

$$\left[y_{\min} + (i - 1)\frac{y_{\max} - y_{\min}}{b}, y_{\min} + i\frac{y_{\max} - y_{\min}}{b} \right] \tag{7.10}$$

How to decide the number of intervals becomes the issue in applying the decision tree to the regression task.

The structure of the decision tree for doing the regression task is illustrated in Fig. 7.15. The continuous output value is discretized into a finite number of intervals. The categories that are given as terminal nodes are replaced by the intervals. The regression is mapped into a classification task in using the decision tree. The linear regression models or the multiple layer perceptrons are used for predicting a continuous value, instead of the decision tree.

The process of applying the decision tree to the multivariate regression is illustrated in Fig. 7.16. The task refers to the regression that predicts more than

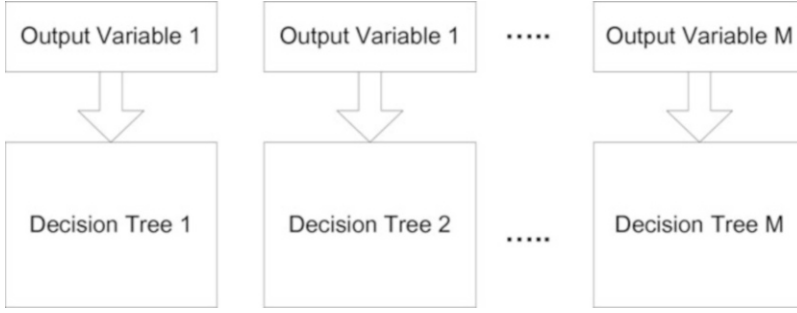


Fig. 7.16 Multivariate regression version

one output variable. The multivariate regression may be decomposed into independent univariate regressions as many as output variables. The decision tree is applied by the process that is mentioned above for each univariate regression task, independently. It is possible to decompose the multivariate regression into binary classification, through several steps.

Let us make some remarks on the decision tree that is applied to the regression task. The decision of the number of intervals of the output value is an important issue in applying the decision tree to the regression problem. The value interval is provided, rather than a particular continuous value, in this case. The regression task is mapped into a classification task, and it may be decomposed further into binary classification tasks. The independent decision trees as many as the output variables are constructed in the case of the multivariate regression task.

7.4.2 Decision List

The decision list is defined as a disjunction form of symbolic rules for the positive class. The given problem is assumed to be a multiple classification where the interested class is the positive class. The symbolic rules are defined as a disjunctive form that connects symbolic roles by “or,” from the training examples, for each category. If one rule is applicable to the positive class, at least, in the disjunctive form, the item is classified to the corresponding class. This section is intended to study the decision list with respect to its difference from the decision tree and its learning process.

The training examples are mentioned before discussing the learning and the classification of the decision list. It is assumed that the categories are given as a finite set, which is shown in Eq. (7.11),

$$C = \{c_1, c_2, \dots, c_{|C|}\} \quad (7.11)$$


```

extractDecisionRuleList(List trainingExampleList, List attributeList, List categoryList){
  do until trainingExampleList.isSparse(){
    for each category in categoryList{
      identicalLabeledExampleList = trainingExampleList.getIdentialLabeledExampleList(category);
      decisionRule = new DecisionRule(category);
      for each attribute in attributeList{
        valueList = attribute.getValueList();
        value = valueList.getMaximumConfidence(category);
        decisionRule.addCondition(attribute, value);
        attribute.setValueList(valueList.remove(value));
      }
      coveredExampleList = trainingExampleList.applyDecisionRule(decisionRule);
      trainingExampleList.remove(coveredExampleList);
    }
  }
}

```

Fig. 7.17 Extraction of decision rules

and the classification type is the hard classification where each item is classified to only one category in the set. Each training example is given as a d dimensional vector, $\mathbf{x}_i = [x_{i1} \ x_{i2} \ \dots \ x_{id}]$, and the training set is notated by $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. It is assumed that an individual value is given as a discrete value, $x_{ij} = v_j \in V = \{v_1, v_2, \dots, v_m\}$, where i is the training example index, and j is the attribute index. Each item in the disjunctive form is notated by $a_j = v_j$.

The process of extracting the decision rules from the training examples is illustrated as a pseudo code in Fig. 7.17. The three sets are given as the input: the training example set, the predefined category set, and the attribute set. The value with the maximum coverage for the category is found among the attribute values for each attribute and each category, and such attribute value is set by the decision rule, which is shown in Eq. (7.12),

$$\text{if } (a_i = v_i) \text{ then } c_j \quad (7.12)$$

The rule is applied to some among the training examples, and the training set is updated by removing the ones to which the rule is applied. When the number of the remaining training examples becomes sparse, the extraction of the decision rules is terminated.

The process of classifying an item by the decision list is illustrated as a pseudo code in Fig. 7.18. It is assumed that each decision rule consists of an attribute value as the conditional part and a category as the causal part. When the matching rule is available, the category is extracted. The extracted one is assigned to the novice item. If one decision rule matches at least, the category may be assigned to the novice item; the decision rule is given as a conjunctive form for each category.

We need to compare the decision tree that is studied in Sect. 7.3 entirely and the decision list that is covered in this section with each other. The root node that is the attribute is available as only one in the decision tree. The conjunction form of the decision rules is given category by category independently in the decision

```

classifyByDecisionList(List decisionRuleList, Item example){
    for each decisionRule in decisionRuleList{
        if(decisionRule.isApplicable(example))
            return decisionRule.getCategory();
    }
}

```

Fig. 7.18 Classification process in pseudo code



Fig. 7.19 Partition of training examples

list. In the decision tree, an item is exclusively classified by following the path from the root node to the terminal node, which is given as only a category, and the overlapping classification is more possible in the decision list by matching more than one conjunctive form. The classification rules are expressed as the symbolic ones that are viewed as if-then form in both.

7.4.3 *Random Forest*

This section is concerned with the random forest that is expanded from the decision tree. We studied the decision tree as a single tree that is constructed from the training examples. The idea of the random forest is to partition the training set into subsets and construct the different decision trees, subset by subset. In the random forest, each item is classified by voting the outputs of the trees. This section is intended to describe the random forest with respect to its differences from the decision tree.

The partition of the training set into subsets is illustrated in Fig. 7.19. The decision tree is constructed from each subset. The random forest is the committee of the decision trees that are constructed from the different subset. The assumption that each subset is a random partition of the training set becomes the reason of calling the decision tree committee random forest. The schemes of partitioning the training set into subsets will be studied in detail in Chap. 13.

The process of constructing the decision tree from each subset is illustrated in Fig. 7.20. The training set is partitioned into subsets at random, as mentioned above. The decision tree is constructed independently from each subset by the process that is described entirely in Sect. 7.3. The K decision trees are given as the committee

Fig. 7.20 Construction of decision tree

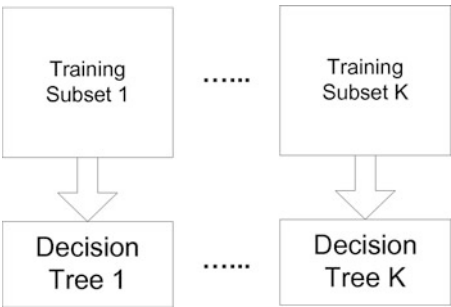
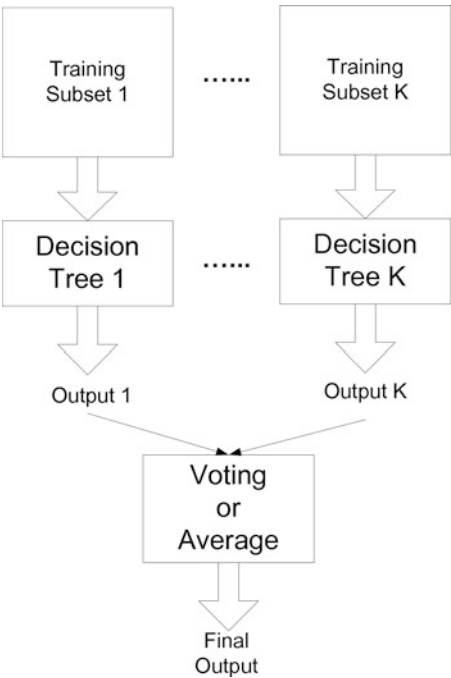


Fig. 7.21 Classification process



after constructing each of them from its own subset. The K decision trees are involved in classifying a novice item, after learning the training examples.

The learning process and the classification process of the random forest are presented in Fig. 7.21. The training set is partitioned into subsets, and the decision tree is constructed from each subset. The novice item is classified by each decision tree in the random forest. The final output of the novice item is decided by voting or averaging the outputs of the decision trees. The scheme of combining the multiple decision trees as machine learning algorithms is called voting, and it will be studied in detail in Chap. 13.

Let us make some remarks on the random forest into which the decision tree is expanded. The random forest is viewed as a combination of multiple decision trees.

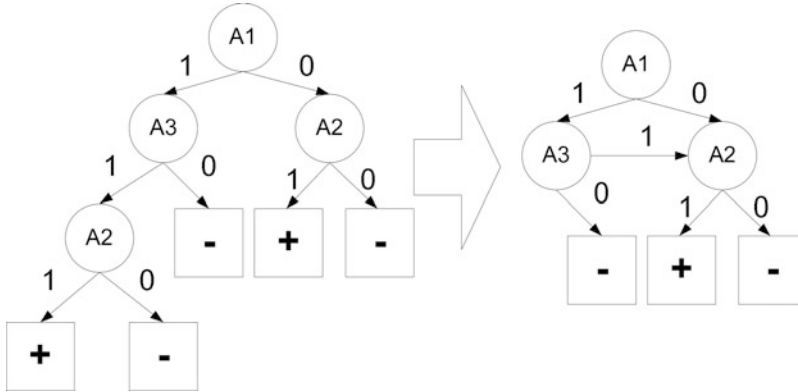


Fig. 7.22 Branch to sibling

Because, in the training examples, it is assumed that each row and each column correspond, respectively, to an individual training example and an attribute, the partition of the training set into several subsets is viewed as the horizontal partition. The random forest may be constructed by partitioning the attribute set into subsets, which is viewed as the vertical partition. The scheme of partitioning the training set and the attribute set is the important issue for building the random forest.

7.4.4 Decision Graph

This section is concerned with the decision graph that is a decision tree variant. We studied the decision list that generates the classification rules as lists in Sect. 7.4.2. The decision graph is the variant that allows the connection between siblings. That is, the different point of the decision graph from the decision tree. This section is intended to describe the decision graph as a decision tree variant.

Mentioning the decision graph is intended to simplify the decision tree structure by connecting the siblings with each other, as shown in Fig. 7.22. The attribute A2 is checked in the cases of A1=1 and A3=1, as shown in the left of Fig. 7.22. Attributes A2 and A3 are siblings that share their parent node A1. The decision tree is simplified by making the branch from A2 to A3, as shown in the right of Fig. 7.22. Because the siblings are connected with each other, the right of Fig. 7.22 belongs to the graph.

The classification process by the decision graph is illustrated in Fig. 7.23. It is implemented by the recursive call of the function, `classifyByNode`. The algorithm is terminated from the function, `classifyByNode`, by returning the current label, in case of the terminal node. Otherwise, the corresponding edge is selected, and the function, `classifyByNode`, with the connected node is called recursively. The

```

classifyByDecisionGraph(Item example){
    startNode = decisionGraph.getStartNode();
    classifyByNode(startNode, example);
}

classifyByNode(Node currentNode, Item example){
    if(isLabel(currentNode)){
        categoryName = currentNode.getCategoryName();
        return categoryName;
    }
    edge = selectEdge(currentNode, example);
    nextNode = (currentNode, edge);
    return classifyByNode(nextNode, example);
}

selectEdge(Node currentNode, Item example){
    attributeValue = example.getAttributeValue(currentNode);
    outgoingEdgeList = currentNode.getOutgoingEdgeList();
    for each outgoingEdge in edgeList{
        if(outgoingEdge.isMatch(attributeValue))
            return outgoingEdge;
    }
}

```

Fig. 7.23 Classification process in pseudo code

classification process is same as that of the decision tree except the connection between siblings.

The combination of the decision tree with the Bayesian Networks is illustrated for simplifying the architecture in Fig. 7.24. When $A1 = 1$ and $A3 = 1$, we need to check the value of attribute $A3$. The causal relation from $A3$ to $A4$ is provided, and the strong correlation between the two attributes is discovered from the probability table. It is converted into the decision tree with its simpler architecture as shown in the right of Fig. 7.24, by omitting attribute $A4$, which is connected from attribute $A3$. What is illustrated in Fig. 7.24 is the case of using the Bayesian Networks for simplifying the decision tree.

Let us make some remarks the decision graph which is mentioned as a variant of the decision tree. We consider the decision graph as the simplified model by allowing siblings to connect with each other. It is very complicated to implement the decision graph for allowing flexibility, which is presented in Fig. 7.22, by visiting nodes in same level. It is easy to access to the parent node and its child node in implementing a tree; it requires visiting the parent node for visiting a sibling node. The left part in Fig. 7.22 is adopted for the easier implementation, in spite of its more complicated architecture.

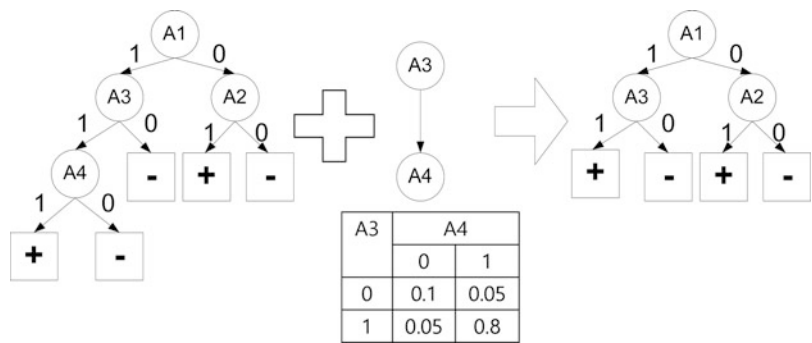


Fig. 7.24 Bayesian networks + decision tree

7.5 Summary and Further Discussions

Let us summarize what is studied in this chapter, entirely. We studied the classification process of the decision tree, by demonstrating some examples. The learning in the decision tree is the process of constructing a decision tree, itself, from the training examples. We studied some variants that are derived from the decision tree, such as the decision list and the decision graph. This chapter covers the architecture, the classification process, the learning process, and the variants of the decision tree.

Depending on which of machine learning algorithms is used, it may require the discretization of attribute values. It requires the discretization for using the Naive Bayes or the Bayesian Learning, which are covered in Chap. 6. Because we define finite number of branches from a node, we need to discretize attribute values in using the decision tree. Without the discretization, we face with the situation where we define the infinite number of branches in building the decision tree. However, we need not discretization in using the KNN algorithm that is covered in Chap. 5 or the SVM (Support Vector Machine) that is covered in Chap. 8.

Let us consider the fuzzy decision tree by introducing the fuzzy concepts to the decision tree which is studied in this chapter. The fuzzy concepts are used for expressing the uncertainty and the value approximations [1]. The branches from a parent node to its child nodes correspond to its corresponding attribute values. A fuzzy decision tree is constructed by expressing branches as fuzzy values. Defining fuzzy distributions to the branches is the additional step for building the fuzzy decision tree.

Let us mention the possibility of extracting the symbolic rules from the decision tree as its merit. This fact becomes the reason of preferring the decision tree to other machine learning algorithms in spite of its less performance. Users require the evidences as well as the accurate answers in case of deciding whether a company is bankrupt or not, using a machine learning algorithm. The capability of presenting the evidence as a symbolic form is an important merit of machine learning algorithms enough to cover their less performance. The decision tree may

be preferred in a field to the neural networks with their higher robustness, tolerance, and performance.

Let us consider building the decision tree from the training examples by the evolutionary computation. Here, we consider the training error and the tree complexity as the basis for evaluating the decision tree fitness. The four types of evolutionary computation are mentioned as genetic algorithm, genetic programming, evolutionary strategy, and evolutionary programming, and in building the decision tree, the second type may be adopted. We may select multiple decision trees by the evolutionary computation, and they are combined with each other as the random forests. We may apply the genetic algorithm or the evolutionary strategy for selecting non-terminal nodes.

Reference

1. T.J. Ross, *Fuzzy Logic with Engineering Applications* (McGraw-Hill, New York, 1995)

Chapter 8

Support Vector Machine



8.1 Introduction

The SVM (Support Vector Machine) is defined as the dual hyperplane classifier with the maximal margin in the mapped space that is called feature space. The SVM that is derived from a single linear classifier, which is called Perceptron, is intended for solving nonlinear classification problems. Another space with the linear separability is defined differently from the original space, and each item is assumed to be mapped into one in another space. The similarity between two items in the feature space is computed by defining the kernel function. This section is intended to describe the basic concepts that are necessary for understanding the learning process of the SVM.

Because the SVM is derived from the linear classifier, we need to mention it before discussing the SVM. The linear classifier is the classification algorithm that sets an optimal hyperplane as the boundary between classes. The linear classifier is modeled as a linear equation, as shown in Eq. (8.1),

$$y = \sum_{i=1}^d w_i x_i + b \quad (8.1)$$

where $x_1, \dots, x_d$ are the input data, $w_1, \dots, w_d$ are the weights, and b is the bias. The weights, $w_1, \dots, w_d$ are optimized through the learning from the training examples. The Perceptron, which is the early neural networks, sets a random single hyperplane and optimizes the weights to minimize the error on the training examples.

The dual space is assumed in using the SVM for a classification task. The initial space of the training examples is called the original space, and another space that is mapped from the original space is called the feature space. The SVM is intended to convert the space with its nonlinear separability into one with the linear separability. The learning process of the SVM is to construct the parallel dual hyperplanes with their maximal margin in the feature space. Mapping explicitly individual training

examples is avoided by using the kernel function of the two original vectors as inner product of ones in the feature space.

The kernel function is viewed as the inner product of two vectors in the feature space. When using the inner product of two vectors in the original space as a kernel function, the SVM is called primitive SVM in viewing mapping no training example in another space. The similarity of two vectors in the feature space is computed without mapping individual examples by defining the function of two vectors, called kernel function. It is the similarity metric or the inner product of two vectors in the feature space. It will be explained in detail and characterized mathematically, in Sect. 8.2.2.

The goal of this chapter is to understand the SVM as the most popular machine learning algorithms. We will understand the process of classifying items by learning SVM. We need to understand the process of learning the training examples by deriving the optimization problem. We will also understand some variants as the modified SVM versions. We make the further discussions about what is studied in this chapter and for deriving the more advanced versions and the hybrid versions that are mixed with other types of machine learning algorithms.

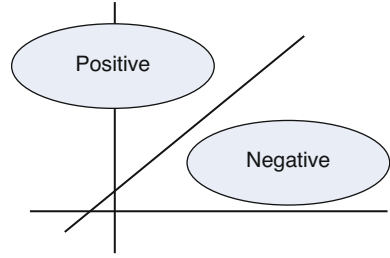
8.2 Classification Process

This section is concerned with the process of classifying items by the SVM. We mention the linear classifier that sets a linear classification boundary as a hyperplane in Sect. 8.2.1 and mention the mathematical definition and characterization of the kernel functions in Sect. 8.2.2. In Sect. 8.2.3, we mention the SVM as the linear classifier that sets dual hyperplanes with the maximal margin between them, and model it as a linear combination of Lagrange multipliers and inner products of the novice example and a training example. In Sect. 8.2.4, we mention the process of classifying them as the generalization. This section is intended to explain the process of classifying items by the SVM.

8.2.1 *Linear Classifier*

The linear classifier is defined as the classification algorithm that sets a hyperplane as the classification boundary. The linear separability is mentioned as the case where all training examples are separated by their categories by a hyperplane in their own space. The linear classifier is applicable completely to the linearly separable problem, and a hyperplane equation is defined as the classification rule. The coefficients of the hyperplane equation are initialized at random and updated in order to classify all training examples correctly as the learning process. This section is intended to describe the linear separability, the hyperplane equation, and the learning process for understanding the linear classifier.

Fig. 8.1 Linearly separable classification



The linear separability of the two dimensional space is visualized in Fig. 8.1. The given problem is assumed as a binary classification where each item is classified into the positive class or the negative class. Each training example is represented as a two dimensional vector, and a single hyperplane is set as the classification boundary. All training examples are classified correctly using the optimal hyperplane in the case of linear separability. The overlapping between the two classes or the quadratic boundary, such as a hyperbolic, is regarded as a nonlinear separability.

The linear boundary is expressed as a hyperplane equation with the assumption of linear separability that is shown in Fig. 8.1. The input vector is assumed to be a d dimensional numerical vector, $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$. The hyperplane that is given as the classification boundary is expressed by Eq. (8.2):

$$w_1x_1 + w_2x_2 + \dots + w_dx_d + b = b + \sum_{i=1}^d w_ix_i \quad (8.2)$$

where w_i is a coefficient or weight and b is the bias. The binary classification is expressed as Eq. (8.3), using Eq. (8.2):

$$y = \text{sign}(b + \sum_{i=1}^d w_ix_i) \quad (8.3)$$

and the negative or the positive is the output value. In the regression, it is expressed as Eq. (8.4):

$$y = b + \sum_{i=1}^d w_ix_i \quad (8.4)$$

Let us mention the learning process of the simple linear classifier where the hyperplane equation is given as a classifier under the assumption of the linear separability. The weights, $w_1, w_2, \dots, w_d$, are initialized in the above hyperplane equation, at random. For each training example, if it is classified incorrectly, the weights are adjusted. Until all of the training examples are classified correctly, classifying each training example and adjusting the weights are iterated. In the case

of nonlinear separability, it is impossible to classify all of the training examples, correctly.

Let us make some remarks on the linear classifier that is described in this section. The linear separability is required for classifying all of the training examples by the hyperplane, correctly. Its equation is given as a weighted linear combination of elements of numerical values in the vector. The learning process is the iteration of updating the weights of the hyperplane equation until their convergence. We need to consider the nonlinear separability that is usual than the linear separability in the real world.

8.2.2 Kernel Functions

The kernel function is defined as the similarity metric or the inner product between two vectors in the mapped space. In studying the SVM, we mention the dual vector spaces: the original space and the feature space. Because a scalar value results from applying the inner product on vector in any dimension, we may compute the similarity or the inner product of two vectors in the feature space through the kernel function without mapping individual vectors in the original space into ones in the feature space. Mapping the original space with its nonlinear separability into the feature space with the linear separability is originally the idea of the SVM. This section is intended to describe the kernel function mathematically, together with the inner product of two vectors.

Let us mention the inner product of two numerical vectors as the primitive kernel function. When two vectors are perpendicular to each other in the geometrical view, the inner product of them becomes zero; it is said that two vectors are orthogonal to each other, in this case. When two vectors are directed identically to each other, in the geometrical view, the inner product of them becomes maximal. The inner product indicates the similarity between them. The cosine similarity between two vectors is the trial to normalize the inner product of two vectors between zero and one.

Let us mention the inner product of two vectors in the feature space. The two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$ are vectors in the original space, and $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$ are ones in the feature space. The inner product of two vectors in the feature space, $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$, is expressed into a kernel function as shown in Eq. (8.5):

$$\Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2) = K(\mathbf{x}_1, \mathbf{x}_2) \quad (8.5)$$

The inner product of $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$ is computed directly from $\mathbf{x}_1$ and $\mathbf{x}_2$ without mapping them. The fact that the inner product from any dimension vectors results in a scalar value is the reason for defining the kernel function of two original vectors as the inner product of ones in the feature space.

The mathematical characterization of the kernel function is illustrated in Fig. 8.2. Adding two kernel functions and/or multiplying a kernel function by a constant

Fig. 8.2 Properties of kernel functions

$$\begin{aligned}
 K(x, y) &= K_1(x, z) + K_2(x, z) \\
 K(x, y) &= \alpha_1 K_1(x, y) \\
 K(x, y) &= K_1(x, y) K_2(x, y) \\
 K(x, y) &= f(x) f(y) \\
 K(x, y) &= K_3(\varphi(x), \varphi(y)) \\
 K(x, y) &= xBy
 \end{aligned}$$

value result in a kernel function. The multiplication of two kernel functions or the multiplication of inner product of two vector functions is also a kernel function. The kernel function whose arguments mapped into vectors in the feature space is also a kernel function. The product of a vector, a matrix, and another vector is also a kernel function.

Let us make some remarks on the kernel function that is covered in this section. The kernel function indicates the inner product of two vectors in the feature space. The kernel function is intended to avoid mapping vectors into ones in the feature space, individually. Because the inner product indicates the similarity between two vectors, the kernel function is necessary for getting the similarity between vectors in the feature space. The kernel function on strings as well as numerical vectors, which is called string kernel, exists [2].

8.2.3 Lagrange Multipliers

This section is concerned with the Lagrange multipliers that are coefficients of products of a training example and a novice example. The linear classifier is expressed as a linear combination of products of a weight and a novice example. The weights are optimized by the training examples in the linear classifier, and the products of a weight and a novice example into those of a Lagrange multiplier and an inner product of a training example and a novice example. The number of the Lagrange multipliers is identical to the number of training examples. This section is intended to describe the mapping process from the weight based linear combination into the Lagrange multiplier based one.

Let us review the hyperplane equation that was mentioned in Sect. 8.2.1 before mapping it into another form. The linear classifier is expressed into Eq. (8.3) under the assumption that the task is given as a binary classification. The weights in Eq. (8.3) are expressed into a weight vector, $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]$. The weight vector is optimized by the training set that is notated by $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. The dual hyperplanes with their maximal margin are expressed as Eqs. (8.6) and (8.7),

$$b + \sum_{i=1}^d w_i x_i = 1 \quad (8.6)$$

$$b + \sum_{i=1}^d w_i x_i = -1 \quad (8.7)$$

Let us map Eq. (8.3) that expresses the linear classifier into the equation that depends on the Lagrange multipliers. $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ is the training set that is given for training the SVM, and $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]$ is the weight vector. The weight vector is expressed as Eq. (8.8):

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i \quad (8.8)$$

and Eq. (8.3) is changed into Eq. (8.9):

$$y = \text{sign}(b + \sum_{i=1}^N (\alpha_i \mathbf{x}_i) \mathbf{x}) = \text{sign}(b + \sum_{i=1}^N \alpha_i (\mathbf{x}_i \cdot \mathbf{x})) \quad (8.9)$$

As the learning process, the process of optimizing the weights, $w_1, w_2, \dots, w_d$, is changed into one of optimizing the Lagrange multipliers, $\alpha_1, \alpha_2, \dots, \alpha_N$. The Lagrange multipliers stand for the importance of training examples in Eq. (8.9), and the training examples that correspond to non-zero Lagrange multipliers are called support vectors.

Let us derive the general equation including the kernel function that expresses the SVM. In the feature space, the vectors that are mapped from the original vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, are $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$. The inner product of the two vectors, $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$, is given as a kernel function of the two original vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, as shown in Eq. (8.5). Equation (8.9) about the original vectors is transformed into Eq. (8.10) about the mapped vector using Eq. (8.5):

$$y = \text{sign}(b + \sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_1) \Phi(\mathbf{x})) = \text{sign}(b + \sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x})) \quad (8.10)$$

The Lagrange multiplier, α_i , becomes a coefficient of the kernel function in Eq. (8.10).

Let us make some remarks on the Lagrange multipliers that are mentioned in this section. The linear combination of attributes values and weights is the equation that expresses a linear classifier. The optimization of weights that are coefficients on the attribute values through the training examples becomes the learning process of the linear classifier. The determination of weights by the training examples is the reason

for mapping the linear combination of attribute values and weights into one of inner products each of which is applied to a training example and a novice example and the coefficients called Lagrange multipliers. The inner product is replaced by the kernel function, in the case of the mapped examples.

8.2.4 Generalization

This section is concerned with the process of classifying a novice item by the general SVM version. The general SVM equation is expressed using the kernel function. In this section, it is assumed that the Lagrange multipliers are optimized by the learning process that is described in Sect. 8.3. The kernel function of a training example and a novice example indicates the similarity between them as the basis for classifying a novice example. This section is intended to describe the SVM as the classification algorithm.

The kernel function was mentioned as the inner product of vectors in the feature space. It is assumed that the vectors in the original space are $\mathbf{x}_1$ and $\mathbf{x}_2$, and ones in the feature space are $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$. The inner product of the two vectors in the feature space is the kernel function of the two vectors in the original space as Eq. (8.5). There are three types of kernel functions: the linear kernel function, $K(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$, the polynomial kernel function, $K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^p (\mathbf{x}_1 \cdot \mathbf{x}_2 + c_i)^i$, and the radial kernel function, $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|)$. The linear kernel function is the case of mapping the vectors into ones in another space.

The general form of the SVM is expressed as Eq. (8.10). It is assumed that the vector, $\mathbf{x}$, in the original space is mapped into the vector in another space, $\Phi(\mathbf{x})$, and the dual hyperplanes in the mapped space are expressed as Eqs. (8.11) and (8.12):

$$y = -b + \sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_i) \Phi(\mathbf{x}) = -b + \sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x}) \quad (8.11)$$

$$y = b + \sum_{i=1}^N \alpha_i \Phi(\mathbf{x}_i) \Phi(\mathbf{x}) = b + \sum_{i=1}^N \alpha_i K(\mathbf{x}_i, \mathbf{x}) \quad (8.12)$$

If the radial kernel function is adopted, the dual hyperplanes in the mapped space are expressed in Eqs. (8.13) and (8.14):

$$y = -b + \sum_{i=1}^N \alpha_i \exp(-\|\mathbf{x}_i - \mathbf{x}\|) \quad (8.13)$$

$$y = b + \sum_{i=1}^N \alpha_i \exp(-\|\mathbf{x}_i - \mathbf{x}\|) \quad (8.14)$$

Equations (8.11) and (8.12) express dual hyperplanes with their maximal margin under the assumption of the optimized Lagrange multipliers.

Let us mention the process of classifying an item by the general SVM version. It is assumed that the Lagrange multipliers, which are presented in Eqs. (8.11) and (8.12), are optimized by the learning process, and it will be described in Sect. 8.4. The given problem is assumed to be a binary classification, and a continuous value is computed by Eq. (8.10). If the value is greater than or equal to +1, it is classified into the positive class, and if it is less than or equal to -1, it is classified into the negative class. In Sect. 8.2, we focus on the classification process, whereas in the subsequent sections, we focus on the learning process.

Let us make some remarks on the process of classifying an item by the SVM. The linear equation based on the attribute weights, $w_1, w_2, \dots, w_d$, is converted into one based on the training example weights called, Lagrange multipliers, $\alpha_1, \alpha_2, \dots, \alpha_N$. Each item is classified by Eq. (8.9), which is based on the linear combination of products each of which consists of a Lagrange multiplier and a similarity between a training example and a novice item. The similarity between two vectors in the feature space is expressed with a kernel function. We mentioned the decomposition into binary classification in Sect. 1.2.4, in applying the SVM to the regression and the multiple classification.

8.3 Learning Process

This section is concerned with the process of optimizing the Lagrange multipliers as the learning process. In Sect. 8.3.1, we describe the primal problem that is derived from the SVM equation by differentiating the loss function. In Sect. 8.3.2, we derive the dual problem as another constraints from the primal problem. In Sects. 8.3.3 and 8.3.4, we mention the SMO algorithm and others for optimizing the Lagrange multipliers. This section is intended to describe the process of optimizing the Lagrange multipliers, together with the definition of constraints.

8.3.1 Primal Problem

This section is concerned with the constraint for optimizing the weight vector, $\mathbf{w}$. An important condition of training the SVM is to maximize the margin between the dual hyperplanes. The equations for optimizing both are derived by differentiating the loss function by the weight vector and the bias. The equations are used for deriving the dual problem from the primal problem. This section is intended to describe the primal problem as the condition for optimizing the weight vector.

Let us express the dual hyperplane margin into an equation for maximizing it. The dual hyperplanes that are parallel to each other are $b + \mathbf{w} \cdot \mathbf{x}_1 = 1$ and $b +$

$\mathbf{w} \cdot \mathbf{x}_1 = -1$, and the direction of the weight vector, $\mathbf{w}$, is perpendicular to the dual hyperplanes. The distance between the dual hyperplanes is expressed into Eq. (8.15):

$$\mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 2 \quad (8.15)$$

The both sides of Eq. (8.15) are divided by the norm of the weight vector, $\|\mathbf{w}\|$, as Eq. (8.16):

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = \frac{2}{\|\mathbf{w}\|} \quad (8.16)$$

and the distance between the dual hyperplanes becomes the projection of the difference between two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, on their own hyperplane onto the weight vector, $\mathbf{w}$. The distance between the dual hyperplanes is proportional reversely to the norm of the weight vector from Eq. (8.16); the minimization of the weight vector means the maximization of the margin between the dual hyperplanes.

Let us mention the mathematical conditions for classifying an item using the SVM. The given problem is assumed to be a binary classification, where each item is classified into the positive class or the negative class. For classifying the item, $\mathbf{x}$, $\mathbf{w} \cdot \mathbf{x} + b \geq 1$ is for the positive class, $y = 1$, and $\mathbf{w} \cdot \mathbf{x} + b \leq -1$ is for the negative class, $y = -1$. The general form of the condition for classifying an item correctly is expressed as Eq. (8.17):

$$y(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 \quad (8.17)$$

$-1 < \mathbf{w} \cdot \mathbf{x} + b < 1$ indicates the position between the dual hyperplanes.

The objective function should be defined from the conditions for the correct classifications over the training examples and the maximal margin between dual hyperplanes. Equation (8.17) and the minimization of the norm of the weight vector, $\|\mathbf{w}\|$, are constraints for defining the objective function. It is defined from the two constraints as Eq. (8.18):

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] \quad (8.18)$$

which should be minimized for satisfying the two conditions. The first term, $\frac{1}{2} \|\mathbf{w}\|^2$, should be minimized, and the second term, $\sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$, should be maximized for minimizing Eq. (8.18). The primal problem is expressed as the objective function that is given in Eq. (8.18).

Even if the vectors are mapped into ones in another space, the linear separability is not guaranteed. Some training example may locate in the margin between the parallel dual hyperplanes. The condition for the correct classifications is modified from Eq. (8.17), into Eq. (8.19):

$$y(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi \quad (8.19)$$

where ξ is called slack variable. The condition of minimizing, $\frac{1}{2}\|\mathbf{w}\|^2$, is modified into minimizing Eq. (8.20)

$$\frac{1}{2}\|\mathbf{w}\|^2 + \xi \quad (8.20)$$

The dual hyperplanes are derived, allowing some training examples on the margin.

8.3.2 Dual Problem

This section is concerned with the process of deriving the dual problem from the primal problem. The primal problem where the weight vector is optimized for minimizing the misclassification and maximizing the margin between the hyperplanes was defined in Sect. 8.3.1. We need to map the primal problem into another type of the problem, which is called dual problem. Satisfying the dual problem, the Lagrange multipliers in Eq. (8.10) should be optimized. This section is intended to describe the process of deriving the dual problem.

We need to differentiate the objective function, which is given in Eq. (8.18), for minimizing it. The objective function is differentiated by the bias, in order to search for the optimal bias. The differentiation is expressed by Eq. (8.21):

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0 \rightarrow \sum_{i=1}^N \alpha_i y_i = 0 \quad (8.21)$$

The sum of products each of which is one of the Lagrange multipliers and a target output that is given as -1 or $+1$ should become zero, as shown in Eq. (8.21). Equation (8.21) becomes the condition for finding the optimal bias.

Let us differentiate the objective function by the weight vector for finding the optimal one. The weight vector, the bias, and the Lagrange multipliers are given as the variables of the objective function. Equation (8.22) is derived by differentiating the objective function by the weight vector,

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \quad (8.22)$$

The weight vector is given as the sum of products of the Lagrange multipliers, the desired outputs, and the training examples. We derive the dual problem by substituting the weight vector by the Lagrange multipliers from the primal problem.

The equation of the dual problem is derived by substituting the weight vector by the Lagrange multipliers using Eq. (8.22). Equation (8.23) is derived from the objective function that is expressed in Eq. (8.18), by Eq. (8.21),

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i y_i \mathbf{w}^T \mathbf{x}_i + \sum_{i=1}^N \alpha_i \quad (8.23)$$

The norm of the weight vector is defined as Eq. (8.24), using Eq. (8.22),

$$\mathbf{w}^T \mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{w}^T \mathbf{x}_i = \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (8.24)$$

The constraint of the dual problem is derived by removing the weight vector, as Eq. (8.25),

$$Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad (8.25)$$

which should be minimized. Equation (8.21) is given as another constraint for minimizing the above equation of the dual problem.

Let us consider the dual problem of the SVM with the soft margin. In this case, some training examples are allowed to locate in the margin between the dual hyperplanes. One more constraint, $0 \leq \alpha \leq C$, is added to the above dual problem; all Lagrange multipliers should not exceed the parameter, C . The dual problem in the SVM with the soft marking is given as Eq. (8.26):

$$\text{minimize } Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j, \quad \sum_{i=1}^N \alpha_i = 0, \quad 0 \leq \alpha \leq C \quad (8.26)$$

The limit of value of the Lagrange multipliers is allowed by putting the training examples in the margin.

8.3.3 SMO Algorithm

This section is concerned with the process of optimizing the Lagrange multipliers. The dual problem was defined as the optimization constraints of the Lagrange multipliers, in Sect. 8.3.2. The SMO algorithm is mentioned as the optimization techniques of the Lagrange multipliers, based on the KKT (Karush–Kuhn–Tucker) condition. The constraint, $0 \leq \alpha \leq C$, is added to the dual problem under the

Fig. 8.3 KKT condition

$$\begin{aligned}\alpha_i &= 0 \Leftrightarrow d_i y_i > 1 \\ 0 \leq \alpha_i \leq C &\Leftrightarrow d_i y_i = 1 \\ \alpha_i &= C \Leftrightarrow d_i y_i < 1\end{aligned}$$

do until satisfy KKT condition

```
for(int i = 0; i < N; i++)  $\alpha_i = 0$  //initialize lagrange multipliers
for(int i = 0; i < N; i++) //for each training example
    if violate against KKT condition
        index1 = i;
        index2 = rand(0, N - 1) //except i
        updateLagrangeMultipliers(index1, index2)
        updateBias(index1, index2)
```

Fig. 8.4 Pseudo code of SMO algorithm

assumption of allowing some training examples in the margin between the dual hyperplanes. This section is intended to describe the process of optimizing both the Lagrange multipliers and the bias.

The KKT condition is illustrated as inequations in Fig. 8.3. Zeros are assigned to the Lagrange multipliers whose training examples are inside the boundary. The Lagrange multipliers that correspond to non-zero constants that are less than the capacity, C , indicate the training examples that locate closely to the correct hyperplane boundary. When the training examples locate in the margin or inside the incorrect hyperplane boundary, the Lagrange multiplier is set to the maximum value, capacity. The KKT condition becomes the guideline for assigning the Lagrange multipliers to the training examples.

The process of optimizing the Lagrange multipliers is illustrated as a pseudo code in Fig. 8.4. All of the Lagrange multipliers are initialized as zeros, $\alpha_1 = \alpha_2 = \dots = \alpha_N = 0$. When the KKT condition is violated, the Lagrange multipliers of the current training example and a random one are updated for each training example. Satisfying the KKT condition of the entire training examples becomes the termination condition of the SMO algorithm. The optimized Lagrange multipliers satisfying the KKT condition are decided as the output of the algorithm.

The process of updating the Lagrange multipliers of the two training examples is illustrated as a pseudo code in Fig. 8.5. y_{index1} and y_{index2} are, respectively, the target outputs of the two examples, $\mathbf{x}_{index1}$ and $\mathbf{x}_{index2}$; $\hat{y}_{index1}$ and $\hat{y}_{index2}$ are, respectively, the computed outputs of them; α_{index1} and α_{index2} are the Lagrange multipliers of them; $K(\cdot)$ is the kernel function of them; and E_{index1} and E_{index2} are the error of them, in Fig. 8.5. The lower bound, L , and the upper bound, H , are computed differently in the different desired output and the same desired output

$$\begin{aligned}
& \text{updateLagrangeMultipliers}(\text{index1}, \text{index2}) \\
& \text{if}(y_{\text{index1}} = y_{\text{index2}}) \\
& \quad L = \max(0, \alpha_{\text{index2}} + \alpha_{\text{index1}} - C) \\
& \quad H = \max(0, \alpha_{\text{index2}} + \alpha_{\text{index1}}) \\
& \text{if}(y_{\text{index1}} \neq y_{\text{index2}}) \\
& \quad L = \max(0, \alpha_{\text{index2}} - \alpha_{\text{index1}}) \\
& \quad H = \max(0, C + \alpha_{\text{index2}} - \alpha_{\text{index1}}) \\
& \quad \eta = K(\mathbf{x}_{\text{index1}}, \mathbf{x}_{\text{index1}}) + K(\mathbf{x}_{\text{index2}}, \mathbf{x}_{\text{index2}}) - 2K(\mathbf{x}_{\text{index1}}, \mathbf{x}_{\text{index2}}) \\
& \quad E_{\text{index2}} = \hat{y}_{\text{index2}} - y_{\text{index2}} \\
& \quad E_{\text{index1}} = \hat{y}_{\text{index1}} - y_{\text{index1}} \\
& \quad \alpha_{\text{index2}}^{\text{new}} = \alpha_{\text{index2}} + \frac{y_{\text{index2}}(E_{\text{index1}} - E_{\text{index2}})}{\eta} \quad | \quad L \leq \alpha_{\text{index2}}^{\text{new}} \leq H \\
& \quad \alpha_{\text{index1}}^{\text{new}} = \alpha_{\text{index1}} + (y_{\text{index1}} y_{\text{index2}})(\alpha_{\text{index2}} - \alpha_{\text{index2}}^{\text{new}})
\end{aligned}$$

Fig. 8.5 The process of updating the Lagrange multipliers

$$\begin{aligned}
& \text{updateBias}(\text{index1}, \text{index2}) \\
& b_1 = E_{\text{index1}} + y_{\text{index1}}(\alpha_{\text{index1}}^{\text{new}} - \alpha_{\text{index1}})K(\mathbf{x}_{\text{index1}}, \mathbf{x}_{\text{index1}}) + y_{\text{index2}}(\alpha_{\text{index2}}^{\text{new}} - \alpha_{\text{index2}})K(\mathbf{x}_{\text{index1}}, \mathbf{x}_{\text{index2}}) \\
& b_2 = E_{\text{index2}} + y_{\text{index1}}(\alpha_{\text{index1}}^{\text{new}} - \alpha_{\text{index1}})K(\mathbf{x}_{\text{index1}}, \mathbf{x}_{\text{index2}}) + y_{\text{index2}}(\alpha_{\text{index2}}^{\text{new}} - \alpha_{\text{index2}})K(\mathbf{x}_{\text{index2}}, \mathbf{x}_{\text{index2}}) \\
& b^{\text{new}} = \frac{1}{2}(b_1 + b_2)
\end{aligned}$$

Fig. 8.6 The process of updating the bias

of the two vectors, and the errors, E_{index1} and E_{index2} , are found between the target outputs and the computed outputs. The new Lagrange multipliers, $\alpha_{\text{index1}}^{\text{new}}$ and $\alpha_{\text{index2}}^{\text{new}}$, of the two vectors, $\mathbf{x}_{\text{index1}}$ and $\mathbf{x}_{\text{index2}}$, are computed by equations in the second last line and the last line in Fig. 8.5. The update of the Lagrange multipliers is intended to satisfy the KKT condition.

The process of updating the bias is illustrated in Fig. 8.6. The two indices of two examples are given as the arguments. The summation of products of the desired output, the difference between the new Lagrange multiplier and the old one, and the kernel function of the two vectors are given. The two values are computed by adding the error to the summation. The bias is updated by averaging the two biases.

8.3.4 Other Optimization Schemes

This section is concerned with the other schemes of optimizing the Lagrange multipliers. In the previous section, we already studied the SMO algorithm for optimizing the Lagrange multipliers as the traditional way. In this section, we will consider other schemes of optimizing the Lagrange multipliers. The SVM may be given as various versions, depending on the kind of the kernel function and the kind of the optimization technique. This section is intended to mention the three optimization schemes: the hill climbing, the local beam search, and the simulated annealing.

The hill climbing is mentioned as a heuristic approach to the Lagrange multipliers. They are initialized at random, keeping the constraints that are expressed in Eq. (8.26). The Lagrange multipliers are modified, and the misclassifications on the training examples are observed in the previous Lagrange multipliers and the modified ones. When the classification on the training examples is improved in the modified ones, the Lagrange multipliers are updated. However, in the approach, falling into a local minima is pointed out as its demerit.

Let us mention a slightly advanced optimization method, compared with the hill climbing. In the hill climbing, it starts with a single set of the Lagrange multipliers that are initialized at random for finding the optimized ones. This scheme starts with multiple sets of the randomly initialized Lagrange multipliers, and the hill climbing is applied to each set. This scheme is intended to avoid falling into a local minima with more probability by searching for the optimal from multiple starting points. This scheme is called local beam search that is viewed as the parallel algorithms, each of which is an independent hill climbing.

Let us mention the simulated annealing as an optimization algorithm for avoiding the local minima. In the optimization algorithms that are mentioned above, the optimum point is found by following more optimal points, so they tend to trap in the local optimal around which are less optimal. The idea of the simulated annealing is to search for the global minima with its more probability by allowing less optimal points. The temperature is given as its external parameter, and the probability of moving into less optimal points is proportional to the temperature. As the search for the optimal point proceeds, the temperature is decreased gradually.

The genetic algorithm may be used for optimizing the Lagrange multipliers. They are encoded into a bit string for using the genetic algorithm. The initial population of the bit strings is constructed at random, and the off-springs are generated by performing the cross over to bit strings. Each solution is evaluated with the fitness value as the classification accuracy on the training examples, and the solutions as many as the population size are selected for the next generation. In using the genetic algorithm, we need to consider assigning non-zero values to only training examples that are on the boundary, called support vectors.

8.4 Variants

This section is concerned with the more advanced version of SVM that was entirely studied in Sect. 8.3. In Sect. 8.4.1, we mention the version of SVM, called fuzzy SVM, where the fuzzy concepts are applied to the kernel function. In Sect. 8.4.2, we derive the version that is called pairwise SVM for applying SVMs for the multiple classification. In Sect. 8.4.3, we study the SVM version where its parameters are optimized by continuous error between its target output and its computed output. In Sect. 8.4.4, we mention the specific version of SVM for dealing with sparse vectors.

8.4.1 Fuzzy SVM

This section is concerned with a particular version of the SVM that allowed the fuzziness. In using the standard SVM that was entirely studied in Sects. 8.2 and 8.3, it is assumed that each item is classified exclusively into either of the positive class and the negative class. In this version, for each item, its membership values of the two classes are computed, rather than the exclusive assignment. The fuzzy SVM is known as the version that assigns the category membership values, rather than applying the fuzziness to the kernel function, as well as the classification. This section is intended to describe the SVM version with respect to the expansion from the initial version and its classification process.

Let us mention the expansion from the initial SVM version, which was entirely studied in Sect. 8.3, to the version. The given task is assumed to be a binary classification where -1 or $+1$ is exclusively assigned to each item, when studying the initial version. When studying the fuzzy SVM, we need to consider assigning a membership value of each category to each item. The output value in this version is given as a continuous value between -1 and $+1$.

Let us consider the process of classifying items with this version of the SVM. In the original version, -1 or $+1$ is assigned to each item as its output. In this version, a membership value is computed to each novice item as the difference of the fuzzy SVM. The membership value to the classified category is reversely proportionally to the distance from the hyperplane. There are dual outputs in using the fuzzy SVM: the classified category and the membership value.

Let us mention the dual problem in this version of the SVM. We consider the membership of each class as the difference from the standard version. The dual problem is expressed as Eq. (8.26). The optimization problem is defined almost identically to that of the standard version, only except the Lagrange multipliers that are less than or equal to the multiplication of the fuzzy membership and the capacity. It is required to associate each training example with its fuzzy membership value as well as the target output, for using the fuzzy SVM.

Let us make some remarks on this version of the SVM. In this version, it is assumed that each training example is labeled with its own category and its

membership value. It is not each to collect such kind of the training examples; it is very subjective to assign the membership values to the training examples, as well as the categories. The fuzziness is applied to other parts of the SVM; the fuzzy kernel function that processes the fuzzy vector or generates a fuzzy value may be considered. Various kinds of fuzzy SVM may be proposed, depending on where the fuzziness is applied in the SVM.

8.4.2 Pairwise SVM

The pairwise SVM is a variant that is applicable to the multiple classification task by defining classes as pairs. When studying the initial version of the SVM in Sect. 8.3, the given problem is assumed to be a binary classification, and the SVM is designed for performing a single binary classification. In this version, the given problem is assumed to be a multiple classification, and the predefined categories are set as category pairs. The decomposition of the multiple classification, which was explained in Sect. 1.2, is different from the process that is mentioned in this section. This section is intended to study the pairwise SVM with respect to the expansion and the differences from the initial version.

Let us mention the expansion of the standard SVM for a single binary classification into the pairwise version for the multiple classification. The positive class or the negative class is decided by Eq. (8.10) in the standard version: the positive class in $1 < b + \sum_{i=1}^n \alpha_i K(\mathbf{x}_i, \mathbf{x})$ and the negative class in $-1 > b + \sum_{i=1}^n \alpha_i K(\mathbf{x}_i, \mathbf{x})$. The predefined categories are assumed as a set $C = \{c_1, c_2, \dots, c_{|C|}\}$, and all possible pairs, $c_i, c_j, i < j$, are generated from the set, C . For each pair, the positive class and the negative class are set to the two categories, and the SVM is applied as a binary classifier. In this case, we need $\frac{|C|(|C|-1)}{2}$ SVMs.

Let us consider performing the multiple classification by the pairwise SVM. The classification equation of the original SVM is given as Eq. (8.9). The category, c_i , with the smaller index is set as the negative class, and the category, c_j , with the smaller index is set as the positive class, in the category pair, $(c_i, c_j), i < j$. The negative output value indicates the categorical score of the category, c_i , and the positive output value does the categorical score of the category, c_j . The categorical score becomes the important reference for deciding the category of the novice item in this multiple classification.

The categorical score is computed for each category after classifying an item in each SVM that is assigned to its own category pair. The categorical score of the category, c_i , is notated by $CS(c_i)$. The categorical score is computed in the binary classification of the category pair, c_i and c_j , by Eq. (8.27):

$$SC(c_i) = \begin{cases} -(b + \sum_{i=1}^n \alpha_i K(\mathbf{x}_i, \mathbf{x})) - 1 & \text{if } i < j \\ (b + \sum_{i=1}^n \alpha_i K(\mathbf{x}_i, \mathbf{x})) - 1 & \text{otherwise} \end{cases} \quad (8.27)$$

and the total categorical score is computed by Eq. (8.28):

$$TSC(c_i) = \sum_{i < j} SC(c_i) + \sum_{i > j} SC(c_i) \quad (8.28)$$

The category with the maximal categorical score is assigned to the item by Eq. (8.29):

$$c_{\max} = \underset{i=1}{\operatorname{argmax}}^M TSC(c_i) \quad (8.29)$$

In the pairwise scheme, $\frac{|C|(|C|-1)}{2}$, SVM classifiers are used for implementing the multiple classification.

Let us make some remarks on the pairwise SVM that is studied in this section. The SVM is initially designed as a binary classifier that separates the positive class and the negative class from each other with the dual hyperplanes with their maximal margin. When using the SVM for the multiple classification or the regression, it is required to modify the SVM for doing the task. All possible pairs are generated from more than two categories, in this SVM version. The SVM is applied by decomposing the multiple classification into binary classifications as described in Sect. 1.3, as the alternative way to the pairwise SVM.

8.4.3 LMS SVM

The least square SVM is defined as the SVM version that sets the dual hyperplanes by minimizing the error squares on the training examples. The traditional SVM is the classifier that defines the dual hyperplanes with the maximal margin for separating the two classes from each other completely. The idea of the least square SVM is to add the error square sum as its objective function to the constraints in the primal problem. The constant error that is allowed in the traditional SVM is with regardless of the training examples, whereas the variable error that depends on the training examples is allowed in this version. This section is intended to describe the least square SVM focusing on its expansion and difference from the traditional SVM.

Let us consider expanding the traditional SVM into the least square SVM. The traditional SVM was initially proposed by Vapnik [1], whereas the least square SVM was initially proposed by Suykens et al. [4]. The traditional SVM is proposed with the assumption of the linear separability in the mapped space, but if there is the classification error on the training examples even in the mapped space, the SVM is modified by introducing the slack variable. The traditional SVM is expanded into the least square SVM by minimizing the error on the training examples, rather than removing it. The means square error that is used as the objective function in learning

the MLP (Multiple Layer Perceptron) is attached to the traditional SVM in this version.

Let us mention the inequality constraints of the least square SVM, together with those of the traditional SVM with the slack variables. In the traditional SVM, Eq. (8.20) should be minimized with the constraint that is expressed in Eq. (8.19). Equation (8.20) is modified into Eq. (8.30):

$$\frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{2} \gamma \sum_{i=1}^M e_i \quad (8.30)$$

with the constraint that is given as Eq. (8.31):

$$y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - e_i \quad (8.31)$$

The slack variable, ξ , is replaced by the error e_i for each training example, in the constraints of the least square SVM. It is assumed that the error for classifying each training example is variable.

The primal problem will be described by modifying that of the traditional SVM, rather than by a proof. In the traditional SVM, the dual problem is defined as Eq. (8.26). The primal problem in the least square SVM is defined by adding the individual training example errors, as expressed in Eq. (8.32),

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i [y_i(\mathbf{w}^T \mathbf{x} + b) - 1 + e_i] \quad (8.32)$$

The dual problem in the least square SVM is generated as Eq. (8.33) from Eq. (8.32):

$$\begin{aligned} by_1 + \frac{1}{\gamma} \alpha_1 y_1 y_1 K(\mathbf{x}_1, \mathbf{x}_1) + \sum_{i=2}^N \alpha_i y_1 y_i K(\mathbf{x}_1, \mathbf{x}_i) &= 1 \\ by_2 + \frac{1}{\gamma} \alpha_2 y_2 y_2 K(\mathbf{x}_2, \mathbf{x}_2) + \sum_{i=1 \wedge i \neq 2}^N \alpha_i y_2 y_i K(\mathbf{x}_2, \mathbf{x}_i) &= 1 \\ \dots \\ by_N + \frac{1}{\gamma} \alpha_N y_N y_N K(\mathbf{x}_N, \mathbf{x}_N) + \sum_{i=1}^{N-1} \alpha_i y_N y_i K(\mathbf{x}_N, \mathbf{x}_i) &= 1 \end{aligned} \quad (8.33)$$

The dual problem is given as a quadratic form in the traditional SVM, whereas it is given as a linear equation system in the least square SVM.

Let us make some remarks on the least square SVM. In using the traditional SVM, the linear separability in the mapped space is assumed. In the least square SVM, the variable error in each training example is considered and the square error

is minimized. In the traditional version, the equation about the dual problem is given as the quadratic form, whereas the equation is expressed as a linear equation system in the least square SVM. One more factor, the variable errors of individual training examples are added in this version.

8.4.4 Sparse SVM

The sparse SVM is defined as the version that is specialized for processing very small number of the training examples. The SVM is known as a classifier that is tolerant to the sparse number of the training examples. The idea of the sparse SVM is to apply different kernel functions hierarchically to the training examples for solving their sparseness. In the traditional SVM, only one kernel function is applied to the training examples, whereas in the sparse SVM, multiple kernel functions are applied to them, hierarchically. This section is intended to describe the sparse SVM as one more SVM variant.

This SVM type is derived from the traditional type for dealing with a sparse number of training examples. The training set in the positive class is notated by $Tr^+ = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N_+}\}$, and each training example is given as a d dimensional vector, $x_i = [x_{i1} \ x_{i2} \ \dots \ x_{id}]$. The different kernel functions as many as the dimension are used for generating the mapped training examples $\mathbf{x}'_{ij} = \{K_1(\mathbf{x}_i, \mathbf{x}_j), K_2(\mathbf{x}_i, \mathbf{x}_j), \dots, K_d(\mathbf{x}_i, \mathbf{x}_j)\}$ where $i < j$. The number of training examples that are mapped by the above process is increased from N_+ to $\frac{N_+(N_+-1)}{2}$. By applying the process to the training examples to the negative class, the number of training examples is increased from N_- to $\frac{N_-(N_--1)}{2}$.

Let us consider the classification equation of the sparse SVM. The novice item is given as a d dimensional vector, $\mathbf{x} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_d]$, the original training set is given as $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and the training set is mapped into $Tr' = \{\mathbf{x}'_1, \mathbf{x}'_2, \dots, \mathbf{x}'_{N'}\}$ by the above process. The novice item is mapped into N vectors, $\mathbf{x}'_1, \mathbf{x}'_2, \dots, \mathbf{x}'_{N'}$ where $\mathbf{x}' = \{K_1(\mathbf{x}, \mathbf{x}_i), K_2(\mathbf{x}, \mathbf{x}_i), \dots, K_d(\mathbf{x}, \mathbf{x}_i)\}$. The classification rule is applied to each mapped vector, by Eq. (8.34),

$$b + \sum_{j=1}^{N'} \alpha_j K(\mathbf{x}'_i, \mathbf{x}'_j) \quad (8.34)$$

and the sum of the mapped N novice vectors is presented in Eq. (8.35):

$$bN + \sum_{i=1}^N \sum_{j=1}^{N'} \alpha_j K(\mathbf{x}'_i, \mathbf{x}'_j) \quad (8.35)$$

The value of y is the sign of Eq. (8.36):

$$y = \text{sign}(bN + \sum_{i=1}^N \sum_{j=1}^{N'} \alpha_j K(\mathbf{x}'_i, \mathbf{x}'_j)) \quad (8.36)$$

Let us mention the dual problem in this version of the SVM as the modified one of the traditional SVM. The dual problem in the traditional SVM version is defined in Eq. (8.26). The training examples, $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, are mapped into one, $Tr' = \{\mathbf{x}'_1, \mathbf{x}'_2, \dots, \mathbf{x}'_{N'}\}$. Equation (8.26) is modified into Eq. (8.37):

$$\text{minimize } Q(\alpha) = \sum_{i=1}^{N'} \alpha_i - \frac{1}{2} \sum_{i=1}^{N'} \sum_{j=1}^{N'} \alpha_i \alpha_j y_i y_j \mathbf{x}'_i{}^T \mathbf{x}'_j, \quad \sum_{i=1}^{N'} \alpha_i = 0, \quad 0 \leq \alpha_i \leq C \quad (8.37)$$

The preprocessing on the training examples is the difference of the sparse SVM from the traditional one.

Let us make some remarks on the sparse SVM that is mentioned as a SVM variant. The sparse SVM is intended to process a very sparse number of the training examples. The traditional SVM is actually designed as the learning algorithm that is tolerable to a small number of the training examples, requiring only support vectors. The traditional version of the SVM is used, instead of this version, by generating virtual training examples that are mentioned in Chap. 14. The definition of various kernel functions is required for using this version of SVM.

8.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We derived ones based on the Lagrange multipliers as the SVM from a linear equation based on its weights. We derived the equations of dual problem from the constraints on the SVM learning and optimized the Lagrange multipliers by the KKT condition. We mentioned some variants that are derived from the SVM, such as the fuzzy SVM, the least square SVM, and the sparse SVM. We studied the classification equation, the learning process, and the variants of the SVM in this chapter.

Let us consider the previous cases of applying the SVM to the text categorization. In 2002, Sebastiani presented that the SVM is the best approach to the text categorization by surveying approaches; the SVM outperformed over other machine learning algorithms in the text categorization [3]. The SVM was mentioned as the main approach to the text categorization in 2018, by Jo in this book on text mining [5]. In applying it to the text categorization, the string kernel that is installed in the SVM was proposed as the kernel function [2]. The SVM is the most popular machine learning algorithm in all application areas including the text classification.

Let us consider using more than one kernel function in the SVM. We have assumed only one kernel function in using the traditional SVM for a classification task. The linear combination of multiple kernel functions of two vectors may be

viewed as a single kernel function. The composite kernel function was applied in the sparse SVM that is covered in Sect. 7.4.4. In multiple kernel functions, we assumed mapping a vector in the original space into ones in various mapped spaces.

Let us consider applying the SVM to the regression task. Previously, we mentioned the process of mapping the regression into binary classification tasks by discretizing the output value. The output value is normalized between -1 and 1 , and the Lagrange multipliers are optimized for minimizing the regression error on the training examples. The output value computed from the SVM equation is demonized into one in the original scale. The value between -1 and 1 may be divided into intervals as many as categories in applying the SVM to the multiple classification task.

Let us consider modifying the SVM that is initially designed as a supervised learning algorithm into an unsupervised version. In the supervised learning, it is assumed that labels are not available in the training set. It is assumed to segment a group into two groups, and two items are selected at random as representatives of two groups. Respectively, we set the similarity of the representative one in a subgroup as a positive value, and the similarity with one in the other as a negative value. The others are assigned into one of the two subgroups based on whether the summation of linear combination is positive or negative.

This part is concerned with the unsupervised machine learning algorithms as the other main part. We mention the AHC algorithm, which proceeds clustering in the bottom-up direction, and the divisive algorithm, which does it in the top-down direction. We mention the EM (Expectation–Maximization) clustering algorithm including the k means algorithm with the two steps: E-Step and M-Step. The clustering index is mentioned as the clustering evaluation metric and used for tuning external parameters of clustering algorithms. This part is intended to describe the three main kinds of unsupervised learning algorithms as the approaches to data clustering, as the advanced clustering techniques.

This part consists of the four chapters. In Chap. 9, we describe the simple clustering algorithms such as the AHC algorithm, the divisive algorithm, and the online clustering algorithms. In Chap. 10, the k means clustering algorithm is mentioned as the most popular clustering algorithm. We describe the EM algorithm as the clustering frame, rather than as a particular clustering algorithm, mentioning the k means algorithm as the simplified version of EM algorithm. In Chap. 12, we describe the computation process of the clustering index and the clustering algorithms with the parameter tunings as the advanced ones.

References

1. V.N. Vapnik, The support vector method, in *International Conference on Artificial Neural Networks* (Springer, Berlin, Heidelberg, 1997), pp. 261–271
2. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, Text classification with string kernels. *J. Mach. Learn. Res.* **2**(2), 419–444 (2002)

3. F. Sebastiani, Machine learning in automated text categorization. *ACM Comput. Surv.* **34**(1), 1–47 2002.
4. J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle, *Least Squares Support Vector Machines* (World Scientific Publishing, Singapore, 2002)
5. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, New York, 2018)

Part III

Unsupervised Learning

Chapter 9

Simple Clustering Algorithms



9.1 Introduction

The data clustering is defined as the process of segmenting a group of entire items into subgroups of similar ones, are called clusters. The data items are initially given as unlabeled ones, and they are clustered depending on their similarities. Since the data items are not labeled (not supervised), the learning algorithms are applied to the problem called unsupervised learning algorithms. In this chapter, we mention the three kinds of simple clustering algorithms and do the advanced ones in subsequent chapters in this part. This section is intended to describe the three kinds of simple clustering algorithms: the bottom-up clustering algorithm, the top-down clustering algorithm, and the fast clustering algorithm.

Let us mention the bottom-up clustering algorithm such as the AHC (Agglomerative Hierarchical Clustering) algorithm as the first type of simple clustering algorithm. It starts with the singletons each of which contains only one item, as many as items. In each phase, it generates all possible pairs and merges the pair with the highest similarity into one cluster. The results from clustering items depend on the similarity metric between two clusters. The number of clusters and the similarity threshold are the termination conditions of iterating the pair generation and the pair merging.

Let us mention the type of clustering algorithm which proceeds the data clustering in the opposite direction to the above type. It starts with a single group of entire items. A particular item is selected from the group at random, and the group is partitioned into the two groups: one of items similar as the selected one, and the rest. The random selection of one and the partition of the group into two groups are iterated. The number of clusters and the similarity threshold are given as the termination condition like the above clustering algorithm.

An online linear clustering is mentioned as a simple and fast clustering algorithm. The algorithm was used for detecting the redundancy or the strong association among national research project proposals by Jo [1]. It was evaluated by Jo in 2006 as the very poor algorithm with respect to its clustering performance [2]; the fact

is the payment to its simplicity and clustering speed. When a very small number of clusters are generated, the time complexity of this algorithm is almost linear. This algorithm is adopted for implementing a clustering system as a light version where its clustering speed is more important than its performance.

The goal of this chapter is to understand some simple clustering algorithms. We need to understand the bottom-up clustering algorithm such as the AHC algorithm and its variants. We will understand the top-down clustering algorithm which is called divisive clustering algorithm. We will also understand the online linear clustering algorithm as the fast clustering algorithm. We will make further discussions about what is studied in this chapter for deriving more variants and hybrid clustering algorithms which are mixed with other types.

9.2 AHC Algorithm

This section is concerned with the AHC algorithm which is a clustering algorithm with the bottom-up direction. We define the similarity metric between two clusters in Sect. 9.2.1, and mention the initial version of AHC algorithm which is applicable to the hard clustering in Sect. 9.2.2. In Sect. 9.2.3, we mention the fuzzy clustering which allows the overlapping between clusters and explain how to apply the AHC algorithm to the fuzzy clustering. In Sect. 9.2.4, we mention the variants which are derived from the AHC algorithm. This section is intended to describe the bottom-up clustering algorithm, called AHC algorithm, together with its variants.

9.2.1 Cluster Similarity

This section is concerned with the process of computing the similarity between two clusters. The cosine similarity and its variants were mentioned as the similarity metric under the assumption of encoding raw data into numerical vectors. The cosine similarity between mean vectors of clusters may be defined as the similarity between clusters. Additionally we will mention some variants for computing the similarity between clusters. In this section, we review the similarity metric and mention the cluster similarity metric and its variants, for understanding how to compute the similarity between clusters.

Let us review the scheme of computing the similarity between two individual items, before accessing one between two clusters. Two items are represented into d dimensional numerical vectors, $\mathbf{x}_1 = [x_{11} \ x_{12} \ \dots \ x_{1d}]$, and $\mathbf{x}_2 = [x_{21} \ x_{22} \ \dots \ x_{2d}]$. The cosine similarity between the two vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$, is expressed as Eq. (9.1),

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} \quad (9.1)$$

The cosine similarity is always given as a normalized value between 0 and 1, as shown in Eq. (9.2),

$$0 \leq \cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} \leq 1 \quad (9.2)$$

When two vectors are orthogonal to each other, the cosine similarity becomes zero as shown in Eq. (9.3),

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} = 0 \quad (9.3)$$

When two vectors are identical to each other, the cosine similarity becomes one as shown in Eq. (9.4),

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|} = 1 \quad (9.4)$$

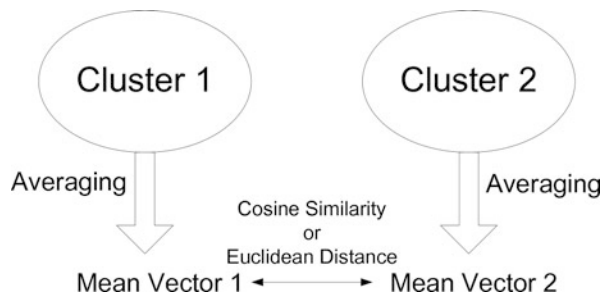
In this section, we adopt the cosine similarity for calculating the similarity between clusters.

The scheme of computing the similarity between two clusters is illustrated in Fig. 9.1. The two clusters are given as sets of input vectors: $C_1 = \{\mathbf{x}_{11}, \mathbf{x}_{12}, \dots, \mathbf{x}_{1|C_1|}\}$ and $C_2 = \{\mathbf{x}_{21}, \mathbf{x}_{22}, \dots, \mathbf{x}_{2|C_2|}\}$. The mean vectors of the two clusters are computed by averaging vectors as shown in Eq. (9.5),

$$\bar{\mathbf{x}}_1 = \frac{1}{|C_1|} \sum_{i=1}^{|C_1|} \mathbf{x}_{1i}, \bar{\mathbf{x}}_2 = \frac{1}{|C_2|} \sum_{i=1}^{|C_2|} \mathbf{x}_{2i} \quad (9.5)$$

The similarity of the two clusters, C_1 and C_2 , is computed as the similarity of two mean vectors, as shown in Eq. (9.6),

Fig. 9.1 Cluster similarity by mean vectors



$$sim(C_1, C_2) = cos(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2) \quad (9.6)$$

The inverse Euclidean distance or the cosine similarity of the two mean vectors becomes the similarity between two clusters.

The clusters are treated as the identical sized ones in the above scheme, so let us consider some variants of the above similarity metric. Instead of the cosine similarity, the inverse Euclidean distance between the two mean vectors may be used. The average over similarities of all possible pairs between two clusters may be used instead of the similarity between two mean vectors, and expressed in Eq. (9.7),

$$sim(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{i=1}^{|C_1|} \sum_{j=1}^{|C_2|} cos(\mathbf{x}_{1i}, \mathbf{x}_{2j}) \quad (9.7)$$

The minimum similarity or the maximum similarity, among similarities of all possible pairs of the two clusters, may be selected as the similarity between them. The similarity between two vectors may be selected at random as the similarity between the two clusters.

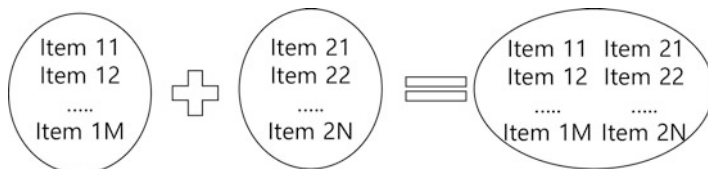
Let us make some remarks on the similarity between clusters which is described in this section. The similarity metric between individual data items is the basis for computing a similarity between clusters. It is required to define the similarity metric between items for computing the similarity between clusters. The cosine similarity and the inverse Euclidean distance are mentioned in this section as the similarity metrics. The schemes of computing the cluster similarity are the similarity between mean vectors of clusters and the average over the similarities of all possible pairs of items in two clusters.

9.2.2 Initial Version

This section is concerned with the initial version or the standard version of the AHC algorithm. It starts with singletons each of which has only one item. It proceeds clustering data items by generating all possible pairs and merging the pair with its highest similarity. The bottom-up is the direction of clustering data items by the AHC algorithm. This section is intended to describe the original version of AHC algorithm and it is expanded into advanced versions in subsequent sections.

The initialized status of the clusters in using the AHC algorithm is illustrated in Fig. 9.2. It is assumed that N data items are initially given as the clustering targets. The N clusters are created and each item is included in its own cluster. The cluster with a single item is called singleton. The AHC algorithm begins with the singletons as many as the items, in clustering data items.

Merging two clusters into a cluster is illustrated in Fig. 9.3. Most similar clusters are merged into one cluster in the process of clustering data items by the AHC algorithm. The items in cluster 1, $item11, \dots, item1M$, and the items in cluster

**Fig. 9.2** Initial clusters**Fig. 9.3** Cluster merge

2, $item_{21}, \dots, item_{2M}$, belong to the merged cluster, under the assumption that they are exclusive with each other. The union which is a set operation is applied to merging two clusters in the case of soft clustering. In the merged one, we need to select its representative member or compute its prototype.

Let us mention the process of clustering data items by the AHC algorithm. It starts with singletons, as many as items, as mentioned above. All possible pairs of clusters are generated, the similarity is computed for each pair, and the pair with the maximal similarity is merged into a cluster. Until the number of clusters reaches the desired one, the three steps are iterated. The complexity of clustering data items by the AHC algorithm is the quadratic complexity to the number of items, n , $O(n^2)$.

Let us make some remarks on the initial version of AHC algorithm in clustering data items. The bottom-up where it begins with singletons as many as individual data items is the clustering direction in this type of clustering algorithm. Clustering results depend strongly on the similarity metric; if the similarity metric is replaced by another, different clustering results are expected. Depending on the similarity metric between individual items and the scheme of computing the similarity between clusters, we can make various versions of AHC algorithm. The divisive cluster algorithm which is covered in Sect. 9.3 is opposite to the AHC algorithm with respect to the clustering direction.

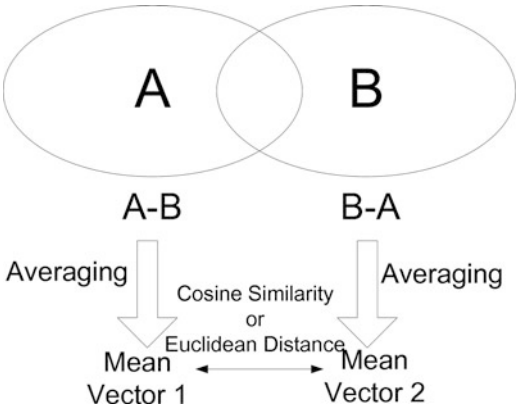
9.2.3 Fuzzy Clustering

This section is concerned with the fuzzy clustering using the AHC algorithm. The process of clustering data items exclusively using the AHC algorithm was already covered. We will study the process of clustering them allowing the overlapping between clusters, using the clustering algorithm. The two interpretations of the fuzzy clustering exist: the view of it into overlapping cluster graphically and view of it into an item-cluster matrix which consists of continuous cluster membership

	Cluster1	Cluster2	Cluster3	Cluster4
Item 1	O	X	X	O
Item 2	O	O	X	X
Item 3	X	X	O	O
Item 4	X	O	O	X
Item 5	X	O	O	X
Item 6	O	O	O	X
Item 7	X	O	O	X
Item 8	X	O	O	X

Fig. 9.4 Overlapping definition

Fig. 9.5 Similarity between overlapped clustering



values. This section is intended to describe the overlapping, the similarity between overlapping clusters, and the fuzzy clustering process.

The overlapping between clusters is visualized by Fig. 9.4. The eight items are given as rows, the four clusters are given as columns, and each cell indicates whether an item belongs to a cluster, or not. Item 1 belongs to cluster 1 and 4, and item 2 does to cluster 1 and 2. Item 6 belongs to all clusters, except cluster 4. Item 2 and 6 are items in the overlap between cluster 1 and 2.

The process of computing the similarity between the overlapped clusters is illustrated in Fig. 9.5. The items which belong to both clusters are removed from each cluster. After removing them, the mean vectors of two clusters are computed. The cosine similarity between the mean vectors is computed as the similarity between two clusters. The exclusive elements of two clusters are considered for computing the similarity, here.

The process of clustering items using the AHC algorithm, allowing the overlapping, is illustrated in Fig. 9.6. It starts with singletons like the case in the hard clustering by the AHC algorithm. All possible pairs of clusters are generated and the cluster pair with its maximum similarity is merged into a cluster. In the process

```

clusterItemList(List itemList, float similarityThreshold, int clusterNumber){
    for item in itemList{
        cluster = createCluster(item);
        clusterList.add(cluster)
    }
    do repeat until clusterList.size() == clusterNumber{
        maxSimilarity = 0; maxIndex1 = 0; maxIndex2 = 0;
        for each cluster[i] in clusterList{
            for each cluster[j > i]{
                similarity = cluster[i].(cluster[j]);
                if(maxSimilarity < similarity){
                    maxSimilarity = similarity;
                    maxIndex1 = i;
                    maxIndex2 = j
                }
            }
        }
        cluster[maxIndex1].merge(cluster[maxIndex2]);
        clusterList.replace(maxIndex1,cluster[maxIndex1]);
        clusterList.remove(maxIndex2);
    }
    for each cluster[i] in clusterList{
        for each cluster[j > i]
            fuzzifyClusters(cluster[i],cluster[j], similarityThreshold);
    }
}

fuzzifyClusters(Cluster cluster1, Cluster cluster2, float similarityThreshold){
    for each item in cluster1{
        similarity = cluster2.getSimilarity(item);
        if(similarity >= similarityThreshold)
            cluster2.add(item);
    }
    for each item in cluster2{
        similarity = cluster1.getSimilarity(item);
        if(similarity >= similarityThreshold)
            cluster1.add(item);
    }
}

```

Fig. 9.6 Clustering process for fuzzy clustering

of fuzzifying two clusters, the items with its higher similarity than the similarity threshold are put in both clusters. Adding the fuzzification process to the process of the hard clustering is the process of the fuzzy clustering.

Let us make some remarks on the AHC algorithm for the fuzzy clustering which is mentioned in this section. The bottom-up is basically the direction of clustering data items in both hard clustering and fuzzy clustering. The fuzzification on cluster pairs is added to the hard clustering version of the AHC algorithm. If the similarity threshold is set very high, almost hard clustering is resulted in. The exclusive cluster pair is considered to be mapped into an overlapping one, called partial merge.

9.2.4 Variants

This section is concerned with some variants which are derived from the AHC algorithm. We mentioned the initialization of AHC algorithm which are applied to the hard clustering and the fuzzy clustering, respectively, in Sects. 9.2.2 and 9.2.3. The variants are derived from the AHC algorithm by allowing the merge of more than two clusters, the multiple merges in each phase, and the partial merge. Both the initial version and its variants start with singletons as many as items. This section is intended to mention some variants with respect to their differences from the initial version.

The process of clustering data items by an AHC algorithm variant is illustrated in Fig. 9.7. It begins with singletons as many as items, like the initial version. A particular cluster is selected at random, its similarities with others are computed, and it is merged with ones which are more similar as itself than the similarity threshold. The above process is iterated until the desired number of clusters is reached. The similarity threshold and the final number of clusters are given as the external parameters in the variant.

The pseudo code of clustering data items by another AHC algorithm variant is illustrated in Fig. 9.8. The variant is characterized as the fact that more than two clusters are allowed to be merged in each phase. Each cluster merges other clusters with its higher similarities than the threshold; more than two clusters are merged into one cluster at a time. Multiple merges to all possible pairs of clusters happen. In this variant, the clustering speed is improved but the clustering quality is down-graded as its payment.

The pseudo code of merging two exclusive clusters into two overlapping ones, called partial merge, is illustrated in Fig. 9.9. Because the partial merge is viewed

```
clusterItemList(List itemList, float similarityThreshold, int clusterNumber){
    for item in itemList{
        cluster = createCluster(item);
        clusterList.add(cluster)
    }
    do repeat until clusterList.size() == clusterNumber{
        randomIndex = randomBetween(0,clusterList.size()-1)
        for each cluster[i != randomIndex] in clusterList{
            similarity = cluster[randomIndex].(cluster[i]);
            if(similarity >= similarityThreshold){
                cluster[randomIndex].merge(cluster[i]);
                clusterList.remove(i);
            }
        }
        clusterList.replace(randomIndex,cluster[randomIndex]);
    }
}
```

Fig. 9.7 Merge of more than two clusters

Fig. 9.8 Multiple cluster merges in each phase

```

clusterItemList(List itemList, float similarityThreshold){
    for item in itemList{
        cluster = createCluster(item);
        clusterList.add(cluster)
    }
    for each cluster(i) in clusterList{
        for each cluster(j > i){
            similarity = cluster[i].(cluster[j]);
            if(similarity >= similarityThreshold){
                cluster[i].merge(cluster[j]);
                clusterList.remove(cluster[j]);
            }
        }
        clusterList.replace(i, cluster[i]);
    }
}

partialMerge(Cluster c1, Cluster c2, real similarityThreshold){
    for each item in c2{
        if(c1.getSimilarity(item) >= similarityThreshold)
            c1.add(item);
    }
    for each item in c1{
        if(c2.getSimilarity(item) >= similarityThreshold)
            c2.add(item);
    }
}

```

Fig. 9.9 Partial merge of clusters

into mapping two exclusive clusters into two overlapping ones, it is applicable to only fuzzy clustering. For each item, the item belongs to the both clusters, if its similarities with the both clusters are more than the threshold. Two exclusive clusters are built with more probability in case of very high similarity threshold, but almost one cluster is made in the case of very low similarity threshold. The partial merge of two clusters is expanded into one of more than two clusters.

The parameter tuning may be installed into the AHC algorithm. It refers to the process of updating external parameters automatically based on the current clustering quality, instead of setting them manually or arbitrary. In the standard AHC algorithm with its parameter tuning, termination of iterating the similarity computation of all possible cluster pairs and merging the pair with its highest similarity is decided not by determined number of clusters, but by the clustering quality. The similarity threshold is updated through the parameter tuning, in the AHC variants, instead of fixing it initially. When installing the parameter tuning, it requires higher complexity for executing the AHC algorithm as the payment.

9.3 Divisive Clustering Algorithm

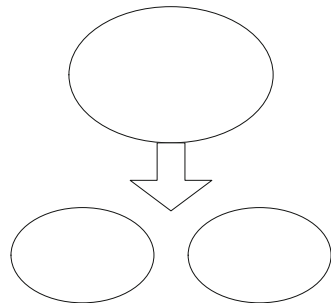
This section is concerned with the clustering algorithm, called divisive clustering algorithm, which clusters data items in the top down direction. In Sects. 9.3.1 and 9.3.2, we explain the binary clustering as the essential part of the divisive clustering algorithm. In Sect. 9.3.3, we describe the process of clustering data items by the standard version of divisive clustering algorithm. In Sect. 9.3.4, we mention some variants of divisive clustering algorithm. In this section, we focus on the divisive clustering algorithm as the alternative to the AHC algorithm.

9.3.1 Binary Clustering

The binary clustering is to partition a group into two subgroups based on the item similarities. There are two types of binary clustering: crisp binary clustering where a group is divided into two exclusive groups and fuzzy binary clustering where a group is divided into two groups with their overlapping. Two representative items are initially selected at random, and the others are arranged into either of the two clusters based on their similarities with the representative ones. As the alternative way, only one item is selected at random, and the others are arranged into the group which is similar as the selected one or the remaining one. This section is intended to describe the binary clustering as the preparation for studying the divisive clustering algorithm.

The binary clustering is illustrated in Fig. 9.10. It is viewed as the partition of a single group into two groups. Two items with their maximum distance are selected from the group, and the others are arranged based on the distances from each of them. One item is selected at random, and the group is divided into the two groups: one which is similar as the selected one and the other which is different from it. The latter is adopted for implementing the divisive clustering algorithm.

Fig. 9.10 Division of single group into two clusters




```

divideIntoTwoExclusiveClusters(Cluster cc, real similarityThreshold){
    randomIndex = randBetween(0,cc.size()-1);
    pivotItem = cluster.getItem(randomIndex);
    Cluster c1 = new Cluster();
    Cluster c2 = new Cluster();
    for each item in cc{
        if(pivotItem.getSimilarity(item) >= similarityThreshold)
            c1.add(item);
        else
            c2.add(item);
    }
    clusterList.add(c1);
    clusterList.add(c2);
    return clusterList;
}

```

Fig. 9.11 Division process for binary clustering

The process of dividing a group into two subgroups is illustrated as the pseudo code in Fig. 9.11. A single cluster and the similarity threshold are given as the arguments, and a particular item is selected as the pivot. The item which is more similar as the pivot than the threshold is arranged into cluster 1, and one which is less similar is done into cluster 2. Cluster 1 whose members are more similar as the pivot and cluster 2 whose members are less similar are generated as the output from the binary clustering. The division of cluster into the similar group and the dissimilar group is used for executing the divisive clustering algorithm.

The process of dividing a group into two groups, allowing the overlapping, is illustrated in Fig. 9.12. The two thresholds, similarity threshold 1 and similarity threshold 2, are used with the condition that similarity threshold 2 is greater than similarity threshold 1. The item with its higher similarity as the pivot than similarity threshold 2 is arranged in the similar group, one with its similarity between the two similarity thresholds is one into both groups, and one with its less similarity than similarity threshold 1 is done into the non-similar group. The interval between the two similarity thresholds indicates the overlapping degree between the two groups. If the two similarity thresholds are same to each other, the group is divided into two exclusive subgroups.

The fuzzy clustering is executed by assigning the cluster membership values to each item, rather than arranging items into clusters. It is assumed that the two clusters are given; one is similar as the pivot and the other is not. The membership values which are given as normalized ones between zero and one are assigned to each item. In the matrix, each row corresponds to an item, and each column corresponds to a cluster; $m \times 2$ is given as the results from clustering m items. Each element in the matrix is a membership value of an item to a cluster.

```

divideIntoTwoOverlapClusters(Cluster cc, real similarityThreshold1,
real similarityThreshold2){
    //similarityThreshold1 < similarityThreshold2
    if(similarityThreshold1 >= simialrityThreshold2)
        return null;
    randomIndex = randBetween(0,cc.size()-1);
    pivotItem = cluster.getItem(randomIndex);
    Cluster c1 = new Cluster();
    Cluster c2 = new Cluster();
    for each item in cc{
        if(pivotItem.getSimilarity(item) >= similarityThreshold1)
            c1.add(item);
        if(pivotItem.getSimilarity(item) < similarityThreshold2)
            c2.add(item);
    }
    clusterList.add(c1);
    clusterList.add(c2);
    return clusterList;
}

```

Fig. 9.12 Overlapped binary clustering

9.3.2 Evolutionary Binary Clustering

This section is concerned with the binary clustering which is performed by an evolutionary computation algorithm. A group is partitioned into two subgroups as the clustering process, and each item is given as an individual bit. The N items are represented as N bits; 0 means the belongingness of the first group, and 1 means the belongingness of the second group. The fitness value is defined for evaluating the quality of the two subgroups and the simple genetic algorithm as the simplest evolutionary computation algorithm is applied to the binary clustering. This section is intended to describe the binary clustering using the simple genetic algorithm.

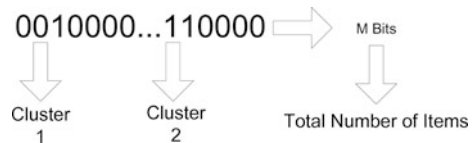
The simple genetic algorithm is illustrated as a pseudo code in Fig. 9.13. Each solution is given as a fixed sized bit string and the initial population is constructed at random. Two bit strings are selected at random from the population, they are cross-overed into off-springs, and they are mutated optionally. This algorithm evaluates the solutions with their fitness values and removes ones with lower fitness values, leading to the next generation. By iterating so, this algorithm searches for optimal solutions.

The representation of the binary clustering results into a bit string is illustrated in Fig. 9.14. The length of the bit string indicates the number of data items which are clustered. In each digit, 0 indicates that it belongs to cluster 1, and 1 indicates that it belongs to cluster 2. Each bit in the bit string corresponds to an individual item. The assumption that the given task is the exclusive binary clustering is inherent in representing the individual items as a bit string.

```

simpleGeneticAlgorithm(int popSize, int iterationNumber
real crossoverRate, real mutateRate){
    solutionList = initPopulation();
    generationNumber = 0;
    do until generationNumber == iterationNumber - 1{
        crossoverSize = popSize * crossoverRate * 0.5;
        for i = 0; i < crossoverSize; i++){
            randomIndex1 = randomBetween(0,popSize-1);
            randomIndex2 = randomBetween(0,popSize-1);
            parent1 = solutionList.getElement(randomIndex1);
            parent2 = solutionList.getElement(randomIndex2);
            offspringPair = parent1.crossover(parent2);
            randomRealValue = randomNormalized();
            if(randomRealValue <= mutateRate)
                offspringPair.mutate();
            solutionList.add(offspringPair);
        }
        for each solution in solutionList{
            solution.evauateFitness();
        }
        for i = 0; i < crossoverSize; i++){
            removalIndex = solutionList.selectPoor();
            selectionList.remove(removalIndex);
        }
    }
}

```

Fig. 9.13 Simple generic algorithm**Fig. 9.14** Example representation

The process of computing the clustering index as the fitness value from the bit string is illustrated in Fig. 9.15. The bit string which is decoded into two clusters of phenotypes is given as genotype. The inter-cluster similarity and the two intra-cluster similarities are computed from the two clusters. The average over the intra-cluster similarities is integrated with the inter-cluster similarity into the clustering index. Solutions in the next generation are decided based on the clustering index.

Let us make some remarks on the evolutionary binary clustering which is covered in this section. The bit string represents two clusters of items. The clustering index is computed for each bit string as its fitness value. The simple genetic algorithm is applied for searching for a bit sting with its higher clustering index. The different clustering results depend on the similarity metric between items.

Fig. 9.15 Fitness value

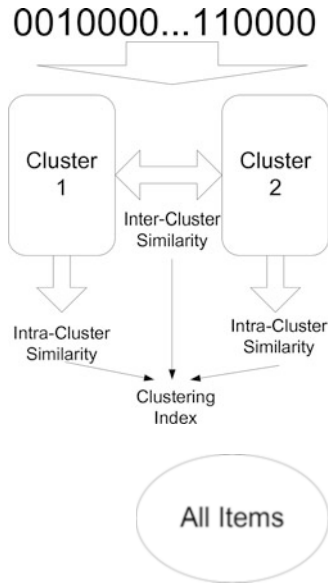


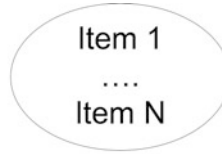
Fig. 9.16 Initial stage

9.3.3 Standard Version

This section is concerned with the process of clustering data items by the divisive clustering algorithm. We studied the process of clustering a group into two subgroups based on the similarities among items, in Sects. 9.3.1 and 9.3.2. It starts with an entire group of data items, and cluster data items by partitioning the rest group into two subgroups, the similar group and the rest, in each phase. In the AHC algorithm, data items are clustered in the bottom-up direction, whereas in this algorithm, they are clustered in the top-down direction. This section is intended to mention the initial version of the divisive algorithm for clustering data items.

The initial status in using the divisive clustering algorithm for clustering data items is illustrated in Fig. 9.16. The divisive algorithm starts with a single group of all items. It proceeds clustering data items in the top-down direction. The group is divided into two subgroups afterward, recursively. The initial status which is shown in Fig. 9.16 is opposite to the status in the AHC algorithm as shown in Fig. 9.2.

The process of computing the intra-cluster similarity from the initial group is illustrated in Fig. 9.17. The similarity between items is computed by the cosine similarity or the inverse Euclidean distance which are mentioned in Chap. 4. The similarities of all possible pairs in the group are computed and the average over them becomes the intra-cluster similarity of the group. It is impossible to compute the inter-cluster similarity from a single group, and the intra-cluster similarity is used as the evaluation metric of the initial group. It was memorized and compared with the evaluation metric in the next stage.

Fig. 9.17 Intra-cluster similarity

$$\text{Intra_Cluster_Similarity} = \frac{2}{N(N-1)} \sum_{i < j} \text{sim}(\text{Item}_i, \text{Item}_j)$$

```

clusterItemList(List itemList, real similarityThreshold, int clusterNumber){
    List clusterList = new List();
    clusterList = clusterList.add(itemList);
    repeat until clusterList.size() == clusterNumber{
        minIntraClusterSimilarity = 0.0;
        for each cluster in clusterList{
            intraClusterSimilarity = cluster.getIntraClusterSimilarity();
            if(minIntraClusterSimilarity > intraClusterSimilarity){
                minIntraClusterSimilarity = intraClusterSimilarity;
                minIndex = currentClusterIndex;
            }
        }
        minCluster = clusterList.getElement(minIndex);
        dividedClusterList = divideIntoTwoExclusiveClusters(minCluster, similarityThreshold);
        clusterList.remove(minIndex);
        clusterList.append(dividedClusterList);
    }
}

```

Fig. 9.18 Cluster division

The process of clustering data items by the divisive clustering algorithm is illustrated as a pseudo code in Fig. 9.18. The intra-cluster similarity is computed for each cluster, and one with the minimal intra-cluster similarity is selected. The cluster is divided into two clusters. Until the number of clusters reaches the desired one, the computation of the intra-cluster similarity and the division of it into two clusters are iterated. This clustering algorithm proceeds clustering data items in the opposite direction to the AHC algorithm.

Let us make some remarks on the process of clustering data items using the divisive cluster algorithm. It starts with an entire group of data items. The process of dividing a group into two subgroups, based on the similarities among data items, was already studied in Sect. 9.3.1. The cluster with the minimal intra-cluster similarity as the poorest cohesion of items is divided into two clusters. Some variants may be derived from the algorithm and will be studied in Sect. 9.3.4.

9.3.4 Variants

This section is concerned with some variants of the divisive clustering algorithm. We already studied its initial version in Sect. 9.3.3. Some variants may be derived

```

divideIntoMultipleClusters(Cluster cc, real similarityThreshold){
    randomItemList = cc.randomSelect();
    for each item in randomItemList{
        Cluster dividedCluster = new Cluster(item);
        dividedClusterList.add(dividedCluster);
    }
    for each item in cc
        dividedClusterList.arrangeIntoNearestCluster(item);
    return dividedClusterList;
}

```

Fig. 9.19 Division into more than two clusters

```

clusterItemList(List itemList, real similarityThreshold,
real intraClusterSimilarityThreshold, int clusterNumber){
    List clusterList = new List();
    clusterList = clusterList.add(itemList);
    repeat until clusterList.size() >= clusterNumber{
        for each cluster in clusterList{
            intraClusterSimilarity = cluster.getIntraClusterSimilarity();
            if(intraClusterSimilarity < intraClusterSimilarityThreshold){
                dividedClusterList = divideIntoTwoExclusiveClusters(cluster, similarityThreshold);
                clusterList.remove(cluster);
                clusterList.append(dividedClusterList);
            }
        }
    }
}

```

Fig. 9.20 Multiple divisions in each phase

from the initial version. One cluster may be divided into more than two clusters, or more than one cluster is divided at same time in each iteration. This section is intended to mention the three divisive clustering algorithm variants.

The process of partitioning a cluster into several nested ones is illustrated as a pseudo code in Fig. 9.19. Among items in the given cluster, several ones are selected at random. For each selected item, its own cluster is created and the item is set as the cluster prototype. For each item in the cluster, it is arranged into the nested cluster whose prototype is most similar. The *k* means algorithm is applied to the current cluster for dividing it into several nested ones, and it is described in detail in Chap. 10.

The process of clustering data items with the second variant of the divisive clustering algorithm is illustrated in Fig. 9.20. In the first variant, the cluster division happens only one time in each iteration, and the cluster is divided into more than two clusters. In this variant, the cluster is divided into two nested cluster, more than one time in each iteration. The argument, *similarityThreshold*, is used for calling the function, *divideTwoExclusiveClusters*. It is possible to implement one more variant where the cluster division happens multiple times in each iteration and each cluster

```

cluserItemList(List itemList, real similarityThreshold1, real similarityThreshold2, int iterationNumber){
    List clusterList = new List();
    clusterList = clusterList.add(itemList);
    repeat with iterationNumber{
        for each cluster in clusterList{
            dividedClusterList = divideIntoTwoOverlapClusters (cluster,similarityThreshold1,similarityThreshold2);
            clusterList.remove(cluster);
            clusterList.append(dividedClusterList);
        }
    }
}

```

Fig. 9.21 Division with overlapping

is allowed to be divided into more than two nested clusters, by combining the two variants with each other.

The process of clustering data items hierarchically allowing the overlapping between clusters is illustrated in Fig. 9.21. The two similarity thresholds are used differently from the clustering algorithm which is shown in Fig. 9.20; one is for dividing a cluster, and the other is for allowing the overlap between the clusters in the division. The process of clustering data items is identical to the version in Fig. 9.20, except dividing the cluster into two with their overlapping. It is possible to expand the algorithm which is presented in Fig. 9.21, into one where each cluster may be divided into more than two. Because the nested matrix is needed in addition, the multiple item–cluster matrices should be defined in the fuzzy hierarchical clustering.

Let us consider the divisive clustering algorithm where the current results are evaluated. The threshold is set as an external parameter, and the intra-cluster similarity is higher than the threshold, the iteration is terminated. A cluster is divided into two cluster: one is similar as the selected item; the other is not similar. The intra-cluster similarity of the group of the rest items is computed, and when the rest have less intra-cluster similarity than the threshold, it is divided into two groups. If the threshold is closely to zero, it results into a small number of big clusters.

9.4 Online Linear Clustering Algorithm

This section is concerned with the third simple clustering algorithm which is called online linear clustering algorithm. In Sect. 9.4.1, we mention some schemes of selecting the representative item in each cluster. In Sect. 9.4.2, we explain the initial version of the online linear clustering algorithm. In Sect. 9.4.3, we mention the process of applying the clustering algorithm to the fuzzy clustering. In Sect. 9.4.4, we mention some variants which are derived from the clustering algorithm which is entirely covered in this section.

9.4.1 Representative Selection Scheme

This section is concerned with the schemes of selecting a representative one as a cluster prototype among items. In the previous section, we studied the two clustering algorithms: the AHC algorithm and the divisive clustering algorithm. In this section, we study one more simple clustering algorithm which characterizes a fast clustering algorithm, and consider the decision of the representative items, before studying it. The representative one is used for computing the similarity of each item with the cluster, in executing the clustering algorithm. This section is intended to mention the three schemes of setting the representative one for each cluster.

The first scheme of selecting the representative one is to set the initial item as the representative one in creating a new cluster. For each item, in executing the online linear clustering algorithm, one more cluster is created or it is included in one or some among the existing ones. When a new cluster is created, the initial item becomes the cluster prototype. This scheme may be also used in initializing the clusters in the k means algorithm, as well as the online linear clustering algorithm. The merit of this scheme is the high speed of clustering items, but the demerit is the poor quality of the clustering results.

As the most popular case, the mean vector is selected as the representative of the cluster. By using the k means algorithm except the initial phase, this scheme of setting cluster representative vector is adopted. The mean vector as average over the vector is for representing its cluster. The member which is most similar as the mean vector may be selected in a variant of the scheme. This scheme is not suitable for the situation where items are not numerical vectors and it is impossible to compute the mean vector.

Let us consider the last item as the representative one of the cluster in using this clustering algorithm. When the mean vector is adopted as the representative one, the reliability is improved, but the speed should be sacrificed in this version. Whenever an item belongs to one of the existing one, the representative one is updated as it joins, in executing the algorithm. We may consider the mid-item between the first item and the last item as the hybrid scheme of the first scheme and the third scheme. Compared with the second one, it takes less time in adopting the first and this scheme.

Let us make some remarks on the scheme of setting a representative vector for each cluster, in executing the online linear clustering algorithm. The similarity between a cluster and an individual item is a very important factor for deciding whether one more cluster is created, or it is included into one of the existing one. It is more efficient to compute the similarity with the only representative one in the cluster, than with all members. Because the online linear cluster algorithm is intended as a fat algorithm, the initial item or the last item are mentioned as the cluster representative. Adopting the mean vector as the representative vector is to sacrifice the clustering speed for improving the clustering quality.

9.4.2 Initial Version

This section is concerned with the process of clustering data items by the online linear clustering algorithm. We studied the scheme of selecting the representative item from each cluster for computing the similarity between it and a particular item. Whether one more cluster is created, or the item is arranged into one or some among existing clusters, depends on the maximum similarity. From this algorithm, we expect the high speed but not good quality in clustering data items. This section is intended to study the process of clustering data items by the online linear clustering algorithm, knowing how to compute the similarity between a cluster and an item.

The initial stage in applying the online linear clustering algorithm is illustrated in Fig. 9.22. The AHC algorithm starts with the individual singletons as many as individual items, whereas the divisive algorithm starts with a group of all items. This algorithm starts with a cluster with an item, and the others wait for being clustered. In the initial stage, one cluster is created and the initial item is given as its cluster prototype. Choosing whether one more cluster is created or the item joins into an existing one is applied to the second item.

The process to the second item in using the online linear clustering algorithm is illustrated in Fig. 9.23. As shown in Fig. 9.22, the initial cluster is created and the first item is included in the cluster as the process to the first item. The choice of joining into the first cluster or creating one more cluster is given after the process to the first item. The similarity between the second item and the first cluster whose prototype is the first item, and if the similarity is higher than the threshold, it joins into the cluster; otherwise, one more cluster is created. Afterward, more than two clusters may be given in processing subsequent items.

The process to the n item in clustering data items using this algorithm is illustrated in Fig. 9.24. It is assumed that while the $n - 1$ items are processed, more than one cluster is constructed. The similarities with the existing clusters are computed, and the maximum similarity is selected among them. If the maximum similarity with the cluster is greater than the threshold, the item is included into the corresponding cluster, and otherwise, one more cluster is created. The number of clusters and the cluster size depends on the similarity threshold.

Fig. 9.22 Initial stage

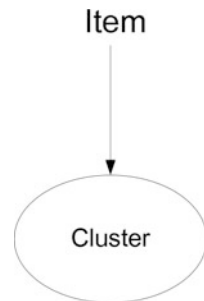


Fig. 9.23 Second item

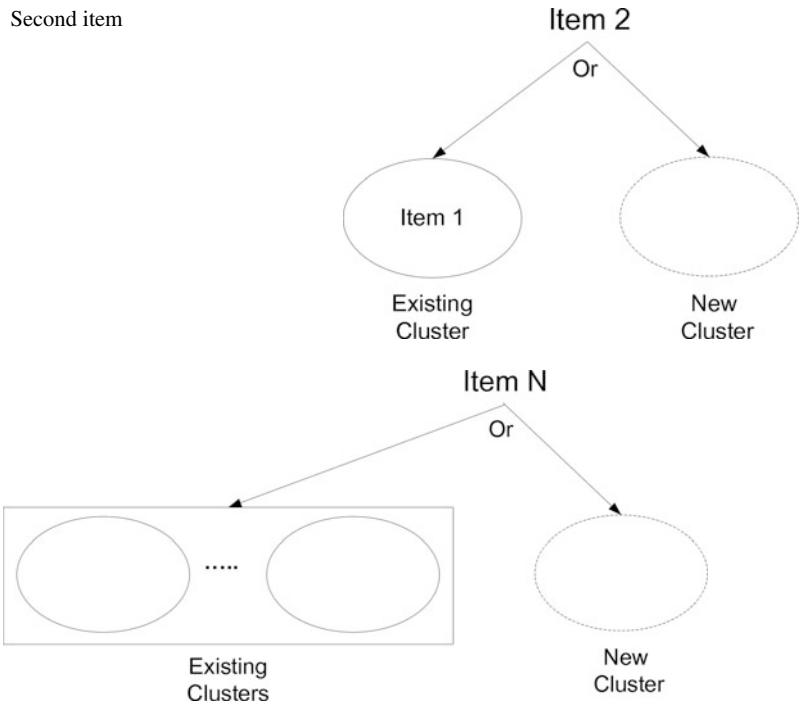


Fig. 9.24 Nth item

Let us make some remarks on the online linear clustering algorithm which is covered in this section. The online linear clustering algorithm is fast in almost linear complexity to the number of items; the data items are usually clustered into a small number of groups. This clustering algorithm was validated as the clustering results with the poor quality, compared with the k means algorithm, by Jo in 2006 [2]. The number of clusters is set automatically during its execution; the similarity threshold is used as an external parameter, instead of the number of clusters. This clustering algorithm is recommended to the case where the high speed clustering is needed but the clustering quality is less important.

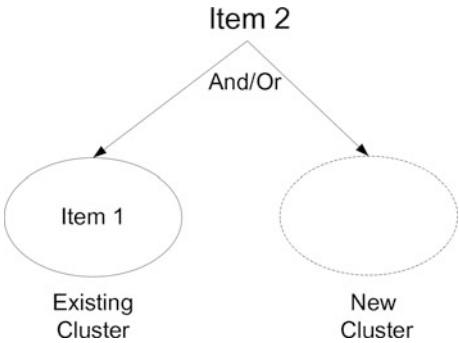
9.4.3 Fuzzy Clustering

This section is concerned with the version of the online linear clustering algorithm which is modified for the fuzzy clustering. In Sect. 9.4.2, we already studied the online linear clustering algorithm, assuming the given problem as a hard clustering. In this section, the given problem is assumed to a fuzzy clustering, and the clustering algorithm is modified into the version which is suitable for the problem. The fuzzy

similarityThreshold1 < similarityThreshold2
If maximumSimilarity < similarityThreshold1, then create New Cluster
If maximumSimilarity between similarityThreshold1 and similarityThreshold2
 then create New Cluster and arrange it into one of Existing Ones
If maximumSimilarity > similarityThreshold2, then arrange it into some of Existing Ones

Fig. 9.25 Parameters in fuzzy clustering

Fig. 9.26 Second item



clustering is viewed into the clusters of data items with their overlapping and the item–cluster matrix which consists of cluster membership values. The former is adopted in clustering data items, and this section is intended to describe the clustering process by this algorithm.

The two similarity thresholds are illustrated as the parameters for using this clustering algorithm for the fuzzy clustering in Fig. 9.25. The first similarity threshold, similarityThreshold1 in Fig. 9.25, is always less than the second, similarityThreshold2. If the maximum similarity with an existing cluster is less than the smaller similarity threshold, one more cluster is created. If the maximum similarity is between the two thresholds, one more cluster is created and the item joins into one among the existing clusters. The idea of the fuzzy clustering is to open the change to belong to more than one cluster.

The choice of the second item is illustrated in Fig. 9.26 by using the online linear clustering algorithm for the fuzzy clustering. By applying the algorithm for the hard clustering, a new cluster is created or the item joins into the existing one. If the algorithm is applied to the fuzzy clustering, one more choice is added; one more cluster is clustered, and the second item belongs to both the new cluster and the existing one. The chance to belong to the both clusters is available in this case.

The choices to the n th item are illustrated in using the online linear clustering algorithm for the fuzzy clustering in Fig. 9.27. The three choices are considered in the n th item in this version. Creating one more cluster and joining into one of existing ones were already mentioned. Both of them are an additional choice in this clustering algorithm for the fuzzy clustering. It is allowed to join into more than one among the existing clusters with their higher similarities.

Let us make some remarks on the application of the online linear clustering algorithm to the fuzzy clustering. It is allowed to belong to more than one

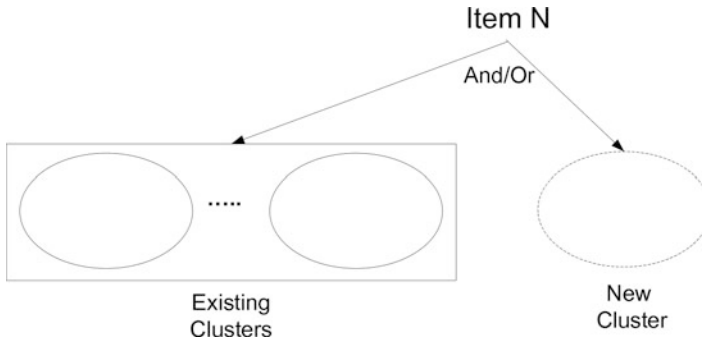


Fig. 9.27 Nth item

cluster by providing the choice of both creating one more cluster and joining into one of existing clusters. The item–cluster matrix whose elements are the cluster membership values is the output of the fuzzy clustering. If the similarity with a cluster is given as a normalized value between zero and one, the similarity is used as the cluster membership value. The similarity with the initial cluster is pointed out as an issue in applying the online linear clustering algorithm to the fuzzy clustering which generates the item–cluster matrix.

9.4.4 Variants

This section is concerned with some variants which are derived from the online linear clustering algorithm. In Sects. 9.4.2 and 9.4.4, we studied the online linear clustering algorithm, respectively, for the hard clustering and the fuzzy clustering. Some variants may be derived from the online linear clustering algorithm by changing from the creation of a single cluster, the join into a single cluster, and a single representative item for each cluster, into the creation of multiple clusters, the join into multiple clusters, and multiple representative items for each cluster. The online linear clustering algorithm may be considered for generating the item–cluster matrix, but it is left as the next study. This section is intended to mention some variants of the online linear clustering algorithm.

The variant which initially creates more than one cluster is illustrated in Fig. 9.28. A single cluster is initially created by selecting an item at random in the initial version. In this version, k items are selected at random, and k clusters are created at same time. It is similar as the initialization in the k means algorithm, in that multiple items are selected as the initial mean vectors at random, and multiple clusters are created. In this variant, we need to consider deleting a cluster as well as adding a cluster.

The process of arranging an item into more than one cluster in case of its higher similarities is illustrated in Fig. 9.29. In the initial version of the online linear

Fig. 9.28 Initial multiple clusters

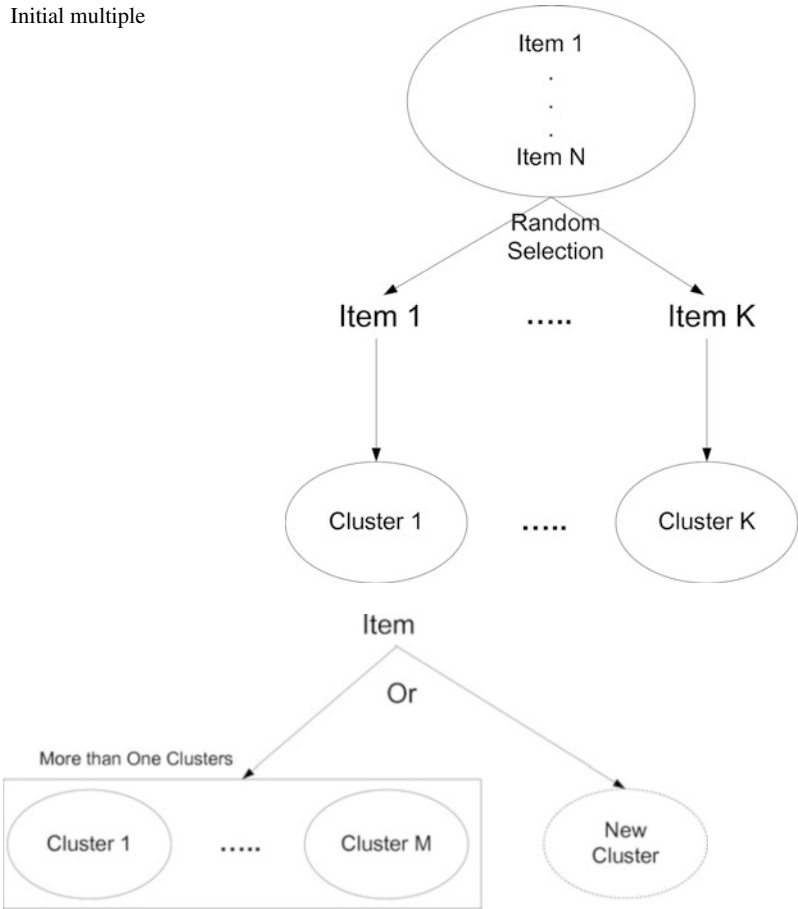


Fig. 9.29 Arrangement into multiple clusters

clustering algorithm, only one cluster is selected. In this version, it is allowed to select clusters to join whose similarities are more than the threshold. There is chance to join into more than one cluster, so the fuzzy clustering results from this variant. If their values are given as normalized values between zero and one, the similarities may be used as the cluster membership values.

The multiple items which represent a cluster are illustrated in Fig. 9.30. Only one item represents cluster in the traditional version of the online linear clustering algorithm. The average over similarities of the item with the representative ones is the similarity between an item and the cluster in executing the clustering algorithm. For example, the initial vector, the mean vector, and representative members are set as multiple cluster prototypes. The schemes of selecting the representative member were mentioned in detail in Sect. 10.4.2.

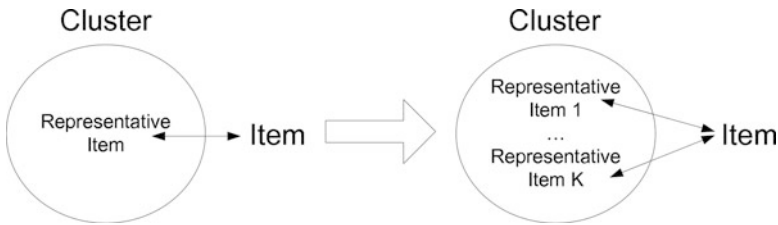


Fig. 9.30 Multiple representative items

Let us consider the reverse version of the initial one as a variant. In the initial version, one cluster of one item is created and clustering data items proceeds by increasing more clusters. The AHC algorithm may be considered as the opposite to the online linear clustering algorithm in the number of clusters, in that it starts with singletons as many as data items. The divisive clustering algorithm which is covered in Sect. 9.3 may be considered as the opposite to it in the number of items in each cluster, in that it starts with individual items. The three kinds of clustering algorithms which are covered in this chapter start with the different conditions, to cluster data items.

9.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. The AHC algorithm starts with the singletons as many as individual items and proceeds its clustering process in the bottom-up direction. The divisive clustering algorithm starts with a single group of all individual items, and proceeds the clustering in the top-down direction, oppositely to the AHC algorithm. The online linear clustering algorithm starts with one singleton, and proceeds its clustering process, sequentially to the items. In this chapter, we studied the three kinds of clustering algorithms as the heuristic approaches to the data clustering.

Let us consider the hierarchical clustering where nested clusters are allowed in a particular cluster. The case where clustering results are given as a flat list of clusters is called flat clustering where any nested cluster is not allowed. In using the AHC algorithm or the divisive algorithm we presented the hierarchical structure of clusters as the visualized form of proceeding the clustering. We need to modify the algorithm itself by using the online linear clustering algorithm. The reason of applying clustering algorithms to the flat clustering rather than the hierarchical clustering is to evaluate the clustering performances easily.

Let us consider the multiple viewed clustering as a clustering type. Although subjectives cluster data items manually, different clustering results are expected. They are caused by different clustering algorithms or different parameters in a same clustering algorithm. This clustering type is intended to accommodate these

different results from clustering data items. We need to integrate multiple results into a single for presenting them simply for users [3].

Let us mention the online clustering as the opposite to the offline clustering which is assumed in describing the clustering algorithms. In the offline clustering, all of data items as the clustering targets are given at a time, whereas in the online clustering, data items are given intermittently. In the offline clustering, data items are clustered as the entire task, whereas in the online clustering data items are given as a stream and clustered continually. In the online clustering, data items which are added newly are clustered incrementally. In order to apply existing clustering algorithms to the online clustering, we need to modify them which was initially designed for the offline clustering.

Let us consider the constraint clustering as a special clustering type. It refers to one where the labeled examples are available, and data items are clustered based on them. The labeled examples are given as the constraints for clustering data items; the fact becomes the reason of calling the clustering type constraint clustering. Using the labeled examples, the initial clusters and their cluster prototypes are constructed and the cluster prototypes are updated by arranging the unlabeled items into clusters. The possibility of additional clusters beyond ones in the labeled examples is opened.

References

1. T. Jo, The application of text clustering techniques to detection of project redundancy in national R&D information system, in *The Proceedings of 2nd International Conference on Computer Science and its Applications* (2003)
2. T. Jo, The implementation of dynamic document organization using text categorization and text clustering. PhD Dissertation, University of Ottawa, 2006
3. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, New York, 2018)

Chapter 10

K Means Algorithm



10.1 Introduction

The k means algorithm is the clustering tool that clusters data items depending on mean vectors of clusters. The parameter, k , in the algorithm means the number of clusters; it should be decided in advance in using the algorithm. The cluster mean vectors are given as the cluster prototypes, in executing the algorithm. The k medoid algorithm where one of the cluster members is selected as its representative will be considered as a variant of the k means algorithm. This section is intended to mention the unsupervised learning, the k means algorithm, and the k medoid algorithm for understanding the clustering process.

Let us explain the unsupervised learning before covering the k means algorithm. It is the learning paradigm where unlabeled examples are learned based on their similarities. The learning process is the process of optimizing the cluster prototypes based on the similarities between prototypes and individual items. The unsupervised learning algorithms are applied to the data clustering tasks. In the k means algorithms, the mean vectors of clusters are given as the cluster prototypes.

Let us mention the k means algorithm as the most typical clustering algorithm. The number of clusters, k , is determined, and k items are selected at random as the initial mean vectors. Each item is arranged into the cluster whose mean vector is most similar. Updating the cluster mean vectors and arranging items so are iterated until the mean vectors converge. The clustering process will be explained in detail in Sect. 10.3.

Let us mention the k medoid algorithm as variants of the k means algorithm. In the k means algorithm, the mean vector of each cluster is the representative one, whereas one among the cluster members is selected as its representative one, in the k medoid algorithm. For each cluster, one that has almost constant distance with the others is selected as the representative item. Its clustering process is same as that of the k means algorithm except selecting a member as the cluster representative. The schemes of selecting one among the cluster members are explained in Sect. 10.4.1.

The goal of this chapter is to understand the k means algorithm as the most popular clustering algorithm. We understand conceptually the unsupervised learning as a machine learning paradigm. We will understand the process of clustering data items by the k means algorithm. We will also understand the variants of k means algorithm and the semi-supervised versions. We make the further discussions on the k means algorithm for deriving more advanced versions and hybrid versions that are mixed with other types of clustering algorithms.

10.2 Supervised and Unsupervised Learning

This section is concerned with both the supervised learning and the unsupervised learning and the relation between them. In Sect. 10.2.1, we mention the transition between the two kinds of learning, in order to show that machine learning algorithms are able to transit between them. In Sects. 10.2.2 and 10.2.3, we describe the unsupervised version and the semi-supervised version of KNN that is known to as a typical supervised version as the case of transiting the supervised learning algorithm into the unsupervised one. In Sect. 10.2.4, we introduce the dynamic data organization system that was proposed by Jo in 2006 as the compound system of the text classification and the text clustering [1]. This section is intended to present modifiability of machine learning algorithms between the supervised learning and the unsupervised learning.

10.2.1 Learning Paradigm Transition

This section is concerned with the transition between the supervised learning and the unsupervised learning. The KNN algorithm, the Naive Bayes, the decision tree, and the SVM, which are covered in Part II, are known as the supervised learning algorithms. The k means algorithm, the EM algorithm, and the Kohonen Networks, which are covered in Part III, belong to the unsupervised learning algorithms. It is possible to transition between the two learning paradigms by modifying each algorithm. This section is intended to review the supervised learning algorithms and the unsupervised learning algorithms and describe the transition, briefly.

Let us review the supervised learning that was studied in Part II. All of the training examples are labeled with one or some of the predefined categories. Two kinds of output value, the target output that is initially assigned to each training examples and the computed output that is computed from processing it, are available, and the learning process is executed to minimize the difference between them. The KNN, the Naive Bayes, the SVM, and the decision tree belong to the supervised learning. The classification and the regression are the tasks to which the supervised learning algorithms are applied.

Let us describe briefly the unsupervised learning that is covered in this part. The training examples are not labeled, initially. The similarity metric between input vectors is defined such as the cosine similarity, and they are learned for stabilizing the cluster prototypes by analyzing the similarities among the training examples. The AHC algorithm, the k means algorithm, and the Kohonen Networks belong to this learning type. Clustering of data items is the task to which we apply unsupervised learning algorithms.

Let us consider modifying the supervised learning algorithm into its unsupervised version. All of the training examples are assumed to be not labeled and clusters are initialized, at random or depending on the prior knowledge. For each cluster, virtual examples are generated or the unlabeled training examples are assigned to clusters based on their similarities with the clusters for building labeled examples. The supervised learning algorithm learns the labeled training examples. The number of clusters, the initial definition of clusters, and the similarity metric are important issues in transiting the supervised learning algorithms into their unsupervised versions.

Let us make some remarks on the transition between the two learning paradigms. We mentioned the supervised learning algorithms and the unsupervised ones, above. The transition between them is possible with only somewhat modification. The transition from the unsupervised learning algorithm to the supervised one is easier than the opposite one. The use of the unsupervised learning algorithms as semi-supervised ones will be covered in Chap. 14.

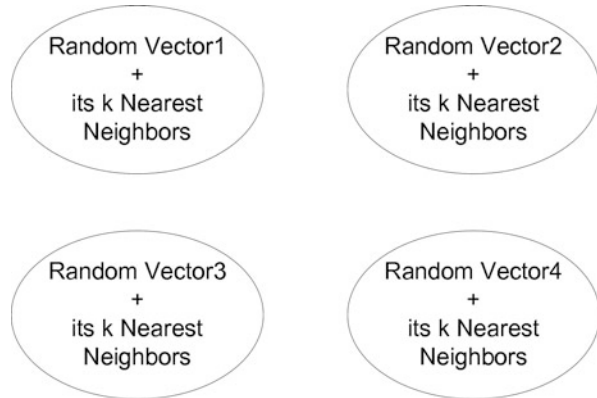
10.2.2 *Unsupervised KNN*

This section is concerned with the unsupervised version of the KNN. We already mentioned the KNN in Chap. 5, as a supervised learning algorithm. By allowing the random selection among unlabeled examples in each cluster, we try to modify the KNN into its unsupervised version. It is possible to use the KNN as the semi-supervised version that is mentioned in Chap. 14, as well as the supervised and the unsupervised. This section is intended to describe the modification of the KNN into the unsupervised version and its application to the data clustering.

The initial status in using the KNN algorithm as a clustering algorithm is illustrated in Fig. 10.1. A fixed number of clusters are initially decided, and data items are selected as initial prototype of clusters at random. For each cluster, the similarities of each prototype with data items are computed, and k most similar data items are arranged into its own cluster. In the fuzzy clustering, the nearest neighbor that belongs to more than one cluster is allowed. The clusters, each of which consists of its prototype vector and its nearest neighbor, are the start of clustering data items, using the KNN algorithm.

The process of clustering data items by the unsupervised version of the KNN is illustrated in Fig. 10.2. In advance, the items as clustering targets and the number of clusters should be given. Each cluster is initialized with the cluster prototype

Fig. 10.1 Initialization of unsupervised KNN



```
clusteringItemsByKNN(List itemList, int clusterNumber){
    clusteredItemList = initializeClusters(itemList, clusterNumber);
    repeat do until Convergence{
        for each item in itemList{
            nearestNeighborList = clusteredItemList.findNearestNeighbors(item);
            int clusterID = nearestNeighborList.vote();
            clusteredItemList.addItem(clusterID, item)
        }
    }
}
```

Fig. 10.2 Clustering process by unsupervised KNN

and its nearest neighbor, and data items are arranged into their own clusters by applying the KNN algorithm. The arrangement is iterated until the cluster prototypes are converged. The number of clusters and the initialized status influence on the clustering results.

Let us mention some variants that are derived from the unsupervised KNN algorithm. A different number of nearest neighbors may be assigned to each cluster, depending on the distance from its prototype; the variant is called unsupervised RNN (Radius Nearest Neighbors). The attributes may be discriminated for computing the similarity between two vectors by assigning different weights to the attributes, depending on their importance degrees. The initial cluster prototype vectors are selected, considering their distances. We may consider the indirect neighbors that are neighbors from a particular neighbor, for clustering data items.

Let us make some remarks on the unsupervised KNN that is used for clustering data items. The KNN algorithm, which is covered in Chap. 5, is known as a supervised learning algorithm. The KNN may be changed into the unsupervised version, as the approach to the data clustering, in this section. It is also possible to transit the unsupervised learning algorithm into the supervised version like the case of transiting the Kohonen Networks, which is covered in Chap. 14, into the supervised version. The initial status, where each cluster consists of its prototype

and its nearest neighbors, is viewed as the collection of artificial training examples for classifying others.

10.2.3 *Semi-supervised KNN*

This section is concerned with the semi-supervised version of the KNN algorithm. In Sect. 10.2.2, we show the possibility of transition between the two learning paradigms by some modifications. The semi-supervised learning is aimed for the classification and the regression like the supervised learning, but it pursuits for using unlabeled examples as well as labeled ones as training examples. The semi-supervised learning will be described in detail in Chap. 14. This section is intended to describe the process of applying the KNN to the classification task using both labeled examples and unlabeled ones.

The initial status in using the semi-supervised version of KNN algorithm is illustrated in Fig. 10.3. It is assumed that both labeled examples and unlabeled ones are given as the training examples for the semi-supervised learning. The labeled examples are arranged into their own categories, and unlabeled ones are arranged by applying the KNN algorithm. In arranging the labeled examples, the target label becomes the criteria, and the similarity with labeled ones is the criterion in arranging the unlabeled ones. As shown in Fig. 10.3, the semi-supervised learning of the KNN starts with the labeled ones and the unlabeled ones that are arranged initially into their own groups.

The process of classifying an item by the semi-supervised version of the KNN algorithm is shown in Fig. 10.4. Both the labeled examples and the unlabeled examples are given as the training examples, and the labeled ones are prepared for building the initial clusters. The unlabeled training examples are labeled by applying the KNN based on the labeled examples, and the subsequently labeled examples are jointed into the originally labeled ones. A novice item is classified by applying

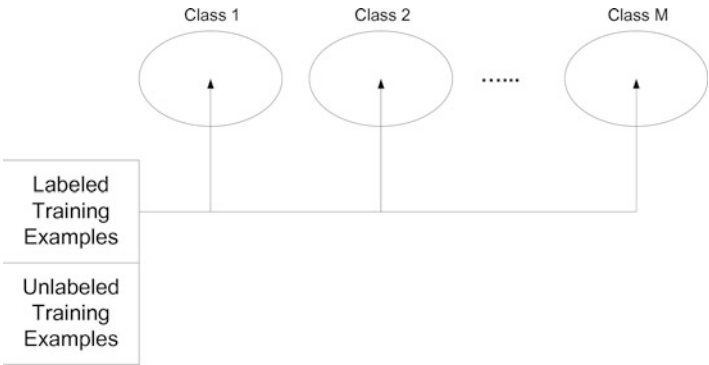


Fig. 10.3 Initialization of semi-supervised KNN

```

learnItemsByKNN(List labeledItemList, List unlabeledItemList){
    for unlabeledItem in unlabeledItemList{
        categoryName = classifyItem(unlabeledItem);
        labeledItemList.add(unlabeledItem,categoryName);
    }
    repeat do until Convergence{
        for each item in labeledItemList{
            categoryName = classifyItem(item);
            labeledItemList.add(unlabeledItem,categoryName);
        }
    }
}

```

Fig. 10.4 Learning process

the KNN based on the both sets of training examples. The process of labeling the unlabeled examples using the original labeled ones is the process that is added for the semi-supervised learning.

Let us mention the constraint clustering that clusters items considering the labeled items. In the both learning paradigms, both kinds of the training examples, the labeled examples and the unlabeled examples, are used, but the semi-supervised learning is aimed at the classification and the regression, whereas the constraint clustering is aimed at the clustering. The labeled examples are arranged into their own clusters by their target labels, and the unlabeled ones are arranged by their similarities, subsequently. The process of applying the KNN for the constraint clustering is identical to that of applying it for the semi-supervised learning, but the final goal is to cluster the labeled items and the unlabeled items. The constraint clustering is intended to cluster data items, by referring the labeled examples that are given accidentally.

Let us make some remarks on the semi-supervised version of the KNN that is mentioned in this section. The labeled examples and the unlabeled ones are given as the training examples in the semi-supervised learning. It is intended to model the classification or the regression using the unlabeled examples as well as the labeled ones. In the semi-supervised learning, it is expected that the unlabeled examples contribute to the classification performance and the regression performance. The semi-supervised learning is similar to the constraint clustering in using both kinds of training examples.

10.2.4 Dynamic Data Organization

Let us mention the DDO (Dynamic Data Organization) that is proposed by Jo [1]. We previously mentioned the three KNN versions: the supervised version, the unsupervised version, and the semi-supervised version. This section covers

Fig. 10.5 Initial maintenance mode

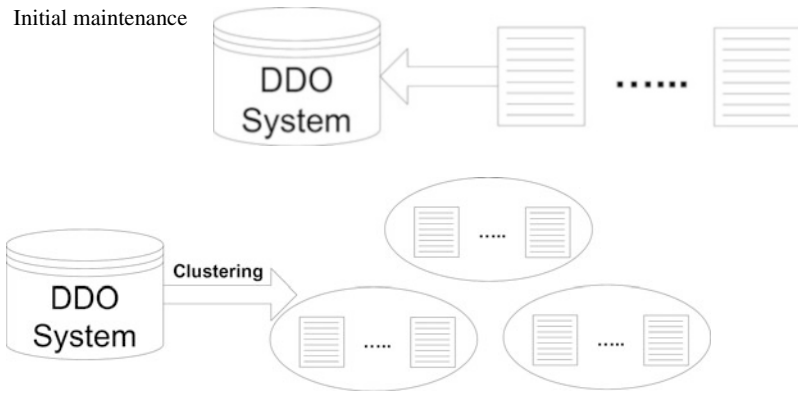


Fig. 10.6 Creation mode

how to use the supervised and the unsupervised versions for implementing the DDO system. It automates the process of managing data items by combining the classification and the clustering with each other. This section is intended to describe the execution of the DDO system, under the assumption of the supervised and the semi-supervised versions of the KNN as the approaches.

The initial maintenance mode of the DDO system is illustrated in Fig. 10.5. Data items are piled in this mode, continually. The mode is viewed into the classification of each item to the category that is given as a single. The mode of classifying items to one of the several categories is called the subsequent maintenance mode. The initial maintenance mode is intended to build the initial group of data items in the DDO system.

The creation mode of the DDO system is illustrated in Fig. 10.6. It means the phase of organizing data items by clustering them. The group of data items that are stored in the system is partitioned into subgroups, each of which contains similar ones, by clustering them in this mode. The k means algorithm, the online linear clustering algorithm, and the Kohonen Networks are used as approaches in the system [1]. An alternative scheme in the creation mode is to modify clusters of items by merging multiple clusters or divide a cluster into multiple clusters.

The maintenance mode after the creation mode is illustrated in Fig. 10.7. In the initial maintenance mode, which is presented in Fig. 10.5, incoming items are piled simply in a single group, whereas in the subsequent maintenance, which is shown in Fig. 10.7, they are classified to one of the clusters. A supervised learning algorithm is used as the approach, and it is trained by the results from the creation mode. Incoming items are classified by their own cluster names and they arranged into their corresponding clusters. We need to monitor the data organization quality for the transit from the maintenance mode to the creation mode.

Let us make some remarks on the dynamic data organization that was initially proposed by Jo in 2006. The system is intended to manage the data organization automatically by combining the classification and the clustering with each other.

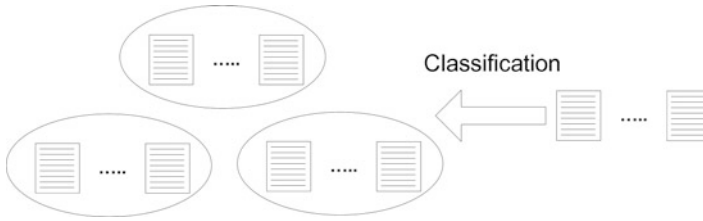


Fig. 10.7 Maintenance mode

The transition rules from the maintenance mode to the creation mode should be defined clearly. In the creation mode, the hard organization that clusters entire data items or the soft organization that governs data items by partitioning or merging data clusters should be decided. In implementing the system, we need to consider more issues that are not covered in the classification or the clustering.

10.3 Clustering Process

This section is concerned with the process of clustering data items by the k means algorithm. In Sect. 10.3.1, we explain the initialization for clustering items by the k means algorithm. In Sect. 10.3.2, we describe the process of clustering data items by the standard version of the k means algorithm. In Sects. 10.3.3 and 10.3.4, we mention the applications of the k means algorithm to the two advanced clustering tasks, the fuzzy clustering and the hierarchical clustering, respectively. This section is intended to describe the hard clustering, the fuzzy clustering, and the hierarchical clustering, by the k means algorithm.

10.3.1 Initialization

This section is concerned with the initialization for using the k means algorithm for clustering data items. A group of items is initially given as the input, and the number of clusters should be decided initially. Items as many as clusters are selected from the group at random. Clusters, each of which has its own selected item, are initial ones in using the k means algorithm. This section is intended to mention the schemes of initializing clusters for using the k means algorithm.

The first step of the k means algorithm is to initialize the mean vectors as many as clusters. In using the k means algorithms for clustering data items, it is assumed that the number of clusters is decided in advance. The initial mean vectors that are selected from the items at random are notated by $\mu_1(0), \mu_2(0), \dots, \mu_k(0)$, where k is the number of clusters and 0 indicates the initial state. Each mean vector that is

updated in the k means algorithm is notated by $\mu_i(t) \leftarrow \mu_i(t + 1)$. The two initial mean vectors that are similar to each other may be selected in this scheme.

We need to consider the distances among the initial mean vectors for more reliable clustering. The discrimination among the clusters is damaged by some very similar initial mean vectors. The distance threshold is set as an external parameter, and if the initial mean vector is less distant from one among the others, at least, than the threshold, another vector is selected at random, as the initial mean vector, again. All possible pairs of cluster mean vectors are generated and the pair with the maximum distance is selected as initial mean vectors, as an alternative way. In adopting the latter, it takes the quadratic complexity for initializing the mean vectors.

In some application areas, the prior knowledge may be used for initializing the cluster mean vectors. We can know the number of clusters and cluster prototypes in advance by the prior knowledge. The cluster mean vectors are initially decided by the prior knowledge, instead of selecting some among the data items at random. We may consider the case where the cluster mean vector may be decided in some clusters, but not in the others. The prior knowledge is given for covering some clusters rather than all clusters; this case is usual.

Let us make some remarks on the initialization of the mean vectors in using the k means algorithm as a clustering tool. Items as many as clusters are selected randomly as the initial mean vectors in the initial version. In the subsequent versions, items are selected, considering their distances. The mean vectors are decided in advance by the prior knowledge or the intuition. The initial mean vectors of the clusters become the seeds for executing the data clustering by the k means algorithm.

10.3.2 *Hard Clustering*

This section is concerned with the process of applying the k means algorithm to the hard clustering. It is defined as the process of segmenting a group into exclusive subgroups of similar items; the overlapping between clusters is not allowed in this clustering type. Each item is arranged into only one cluster, based on its similarities with the cluster mean vectors. The membership values of each item are given as one to a cluster and zeros to the others. This section is intended to describe the process of clustering data items exclusively by the k means algorithm.

The first step in applying the k means algorithm for clustering data items is to estimate the cluster mean vectors as shown in Fig. 10.8. Among data items that are given as d dimensional vectors, vectors are selected at random, as many as clusters as the initial cluster mean vectors. The mean vector for each cluster is estimated by averaging numerical vectors in the cluster, after arranging vector, as shown in Eq. (10.1),

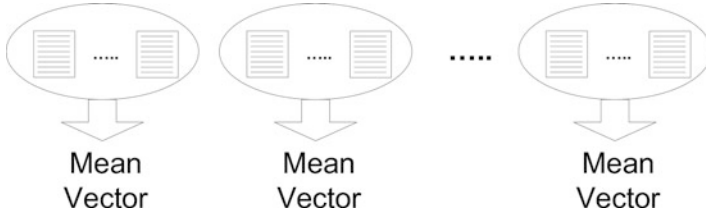


Fig. 10.8 Mean vector estimation

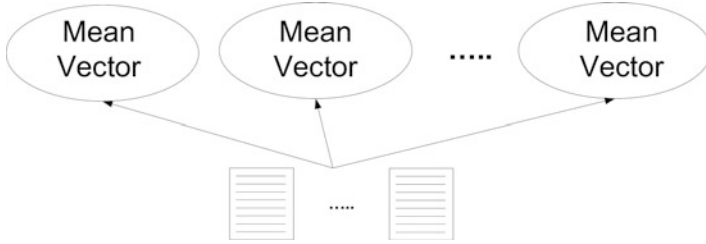


Fig. 10.9 Item arrangement

$$\mu_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x} \quad (10.1)$$

The cluster mean vectors, $\mu_1, \mu_2, \dots, \mu_{|C|}$, are given as the prototypes, each of which characterizes its own cluster. The cluster prototypes become the basis for arranging items into clusters.

Let us explain the process of arranging items into their own clusters, as illustrated in Fig. 10.9. The cosine similarity is adopted as the similarity metric, and the similarity with a mean vector is computed for each cluster, by Eq. (10.2),

$$\text{sim}(\mathbf{x}, \mu_i) = \frac{\|\mathbf{x} \cdot \mu_i\|}{\|\mathbf{x}\| \|\mu_i\|} \quad (10.2)$$

A data item, $\mathbf{x}$, is arranged into the cluster whose mean vector is most similar, as expressed by Eq. (10.3),

$$C_{\max} = \underset{i}{\operatorname{argmax}} \text{sim}(\mathbf{x}, \mu_i) \quad (10.3)$$

When the Euclidean distance is adopted, the item is arranged into the cluster whose distance is minimum, as expressed by Eq. (10.4),

$$C_{\min} = \underset{i}{\operatorname{argmin}} \text{dist}(\mathbf{x}, \mu_i) \quad (10.4)$$

The process of clustering the data items belongs to the hard clustering.

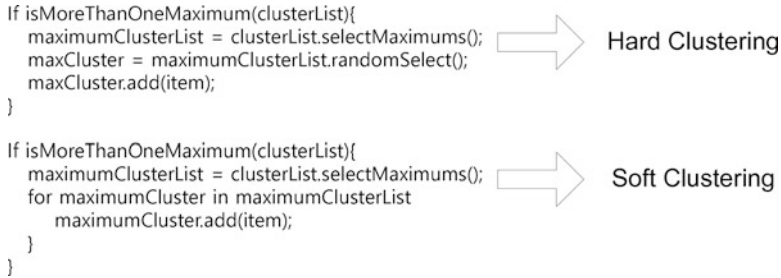


Fig. 10.10 Overlapping allowance

An issue of clustering data items is whether the overlapping between clusters is allowed, or not. There is the possibility of more than one cluster whose mean vectors have a same similarity with a particular item. As shown in Fig. 10.10, there are two policies: one is to arrange the item into only one cluster by selecting one at random, and the other is to arrange it into more than one cluster with the maximum similarity, allowing the overlapping. Even if the second policy is adopted, only limited overlapping between clusters is allowed; only very small number of items belongs to more than one cluster. The limited overlapping should be distinguished from the results from the fuzzy clustering.

All of the examples should belong to only one cluster in the hard clustering. There is difference between the clustering items with very limited overlapping and the fuzzy clustering where unlimited overlapping is allowed. Each item is arranged into clusters where the similarity as the mean vector is higher than the threshold. The cluster membership values of each item are estimated; the cluster–item matrix whose elements are membership values is constructed as the results. We will study the fuzzy clustering of data items using the k means algorithm, called the fuzzy k means algorithm, in Sect. 10.3.3.

10.3.3 Fuzzy Clustering

This section is concerned with the fuzzy clustering with the k means algorithm. In Sect. 10.3.2, we studied the process of clustering data items, exclusively, using the k means algorithm. In order to use the k means algorithm for the fuzzy clustering, we need to modify the initial version; the modified version is called the fuzzy k means algorithm. It is closer to the general version of the EM algorithm, which is covered in Chap. 11. This section is intended to describe the process of clustering data items, using the fuzzy k means algorithm.

The process of computing the similarities of each item with the mean vectors is illustrated in Fig. 10.11. The mean vector of cluster, C_i , is set by selecting a particular item at random or computing items in the previous stage by Eq. (10.1). The similarity of a data item with the mean vector is computed by Eq. (10.2). The

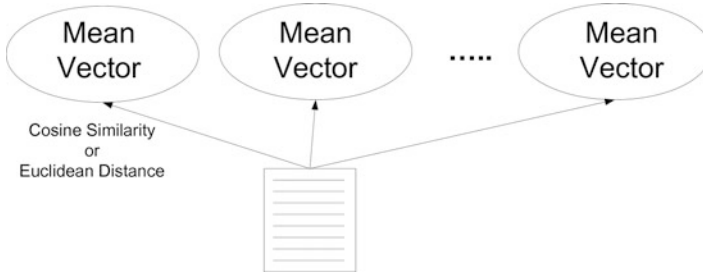


Fig. 10.11 Similarity computation with mean vectors

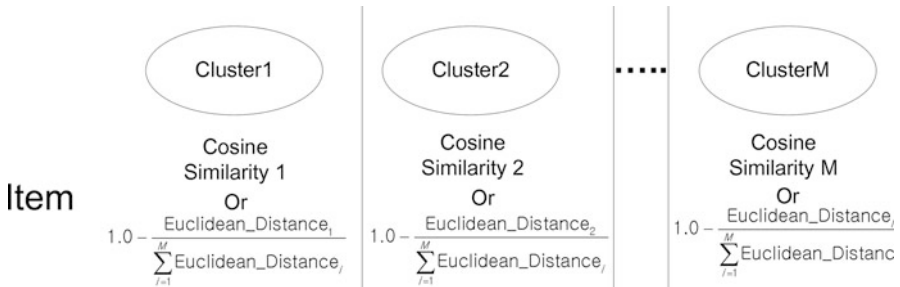


Fig. 10.12 Membership value computation

data item is arranged into the cluster whose mean vector is most similar in the case of the hard clustering, as shown in Eqs. (10.3) and (10.4). The membership values of all clusters are computed for each item in the case of the fuzzy clustering.

The process of computing the cluster membership values for each item is illustrated in Fig. 10.12. The cosine similarity with or the Euclidean distance from it is computed for each item. The cosine similarity is given as a normalized value by itself and used directly as a membership value. The Euclidean distance should be normalized by the total Euclidean distance by the cluster prototypes, and its inverse is taken by subtracting it from 1.0. The normalized values that are the similarities with the cluster mean vectors or the inverse Euclidean distances from them are used as the cluster membership values of each item in the fuzzy clustering.

The process of computing the cluster mean vector in the fuzzy clustering is illustrated in Fig. 10.13. The item–cluster matrix whose element is a membership value of an item to a cluster is defined through the process that is shown in Fig. 10.12. The mean vectors are computed by the linear combination of the cluster membership values and the vectors, column by column in the item–cluster matrix. The cluster mean vectors are used for updating the cluster membership values for each item. The process that is illustrated in Figs. 10.12 and 10.13 is iterated until the convergence of the cluster membership values and the cluster mean vectors.

Let us make some remarks on the process of applying the k means algorithm to the fuzzy clustering. It should be distinguished from the clustering results where

Fig. 10.13 Mean vector computation for fuzzy clustering

	Cluster 1	Cluster 2	...	Cluster M
Item 1	$\mu_{C_1}(X_1)$	$\mu_{C_2}(X_1)$	$\dots$	$\mu_{C_M}(X_1)$
Item 2	$\mu_{C_1}(X_2)$	$\mu_{C_2}(X_2)$	$\dots$	$\mu_{C_M}(X_2)$
	$\dots$	$\dots$	$\dots$	$\dots$
Item N	$\mu_{C_1}(X_N)$	$\mu_{C_2}(X_N)$	$\dots$	$\mu_{C_M}(X_N)$

$$Mean_Vector_i = \sum_{K=1}^N \mu_{C_i}(X_i) \cdot X_i$$

only limited overlapping is allowed. The condition of the hard clustering is that no overlapping exists among clusters at all. The clustering results where at least one item belongs to more than one cluster are regarded as the fuzzy clustering. In the narrow view, it is defined as the process of building the item–cluster matrix that consists of the cluster membership values, instead of clusters.

10.3.4 Hierarchical Clustering

This section is concerned with the process of clustering data items hierarchically, using the k means algorithm. It is initially designed as the tool of the flat clustering. In this case, the k means algorithm is assigned to each cluster for making nested clusters. The AHC algorithm and the divisive clustering algorithm, which are covered, respectively, in Sects. 9.2 and 9.3, are used for clustering data items hierarchically. This section is intended to describe the process of applying the k means algorithm to the hierarchical clustering, in spite of its design for the flat clustering.

Let us review the process of building the flat clusters by the k means algorithm. The items as many as the decided number of clusters are selected at random, and they become the cluster prototypes. It is arranged into the cluster whose prototype is most similar for each item, and the mean vectors are computed for each cluster. Until the mean vectors converge to fixed values, the arrangement and the computation of the mean vectors are iterated. The k means algorithm that is designed for the flat clustering is applied to each cluster in the hierarchical clustering.

We need to select a cluster by applying the k means algorithm to it. The intra-cluster similarity becomes the criteria for selecting the cluster in which the k means algorithm is applied. The threshold as one more external parameter is set arbitrary, the clusters with their intra-cluster similarity less than the threshold are selected, and their nested clusters are generated for each selected one. The cluster with its lower intra-cluster similarity is one with its poor cohesion of its members; it is necessary

to cluster data items, further. The data items are clustered hierarchically by applying the k means algorithm, nestedly.

Let us consider the three ways of clustering data items hierarchically. The first way is to cluster data items hierarchically and exclusively; no item is not allowed to belong to more than even inner nested one. As the medium case, no overlapping is allowed among clusters in the high level, but the overlapping is allowed only among nested ones. As the other case, all of the data items are allowed to belong to more than one cluster in every level. The case of the fuzzy and hierarchical clustering is to define multiple item–cluster matrices, each of which consists of cluster membership values.

Let us make some remarks on the process of applying the k means algorithm to the hierarchical clustering. The k means algorithm is initially designed for doing the flat clustering, rather than the hierarchical clustering. The AHC algorithm and the divisive clustering algorithm, which are covered in Chap. 9, are inherently designed for the hierarchical clustering. It is required to decide the number of clusters in advance for using the k means algorithm. In each cluster, we need to decide the number of clusters for performing the nested clustering.

10.4 Variants

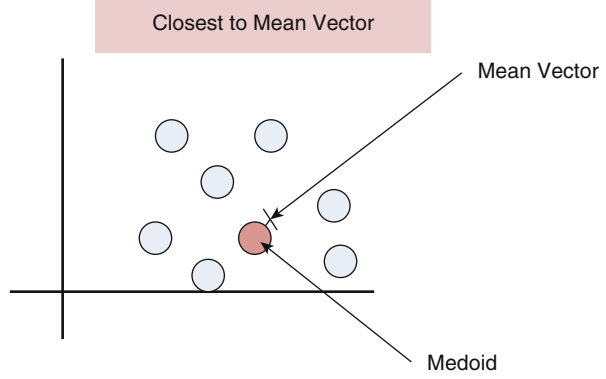
This section is concerned with the variants that are derived from the k means algorithm that was entirely studied in Sect. 10.3. In Sect. 10.4.1, we study the k medoid algorithm by explaining how to select cluster prototypes. In Sect. 10.4.2, we mention the dynamic k means algorithm where the number of clusters is set dynamically. In Sect. 10.4.3, we describe the semi-supervised version of the k means algorithm. In Sect. 10.4.4, we mention the constraint clustering to which we apply the k means algorithm.

10.4.1 K Medoid Algorithm

This section is concerned with the k medoid algorithm as the variant of the k means algorithm. We studied entirely the k means algorithm in Sect. 10.3. The process of clustering items by the k medoid algorithm is identical to that of doing them by the k means algorithm, excepting using a member of a cluster as its prototype, instead of its mean vector. The k medoid algorithm is applied to the data clustering in the situation, where mean vectors are not able to be computed; the similarities among raw items are computed, but they cannot be averaged. Because the clustering process was already described in studying the k means algorithm, let us mention only schemes of selecting the representative ones in this section.

The first scheme of selecting a member as the cluster prototype is illustrated in Fig. 10.14. The cluster is expressed as a set of items, $C_k = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|C_k|}\}$, and

Fig. 10.14 Closet to mean vector



its mean vector is computed for each cluster, as shown in Eq. (10.5),

$$\mu_k = \frac{1}{|C_k|} \sum_{i=1}^{|C_k|} \mathbf{x}_i \quad (10.5)$$

The member that is close to the mean vector is selected as the cluster representative one, as shown in Eq. (10.6),

$$\mathbf{r}_k = \underset{i=1}{\operatorname{argmin}}^{|C_k|} |\mu_k - \mathbf{x}_i| \quad (10.6)$$

As a variant of this scheme, we consider selecting the member that is most similar as the mean vector, by computing the cosine similarity by Eq. (10.7),

$$\mathbf{r}_k = \underset{i=1}{\operatorname{argmax}}^{|C_k|} \frac{\mu_k \cdot \mathbf{x}_i}{\|\mu_k\| \cdot \|\mathbf{x}_i\|} \quad (10.7)$$

It is required to compute the mean vectors for using this scheme.

The second scheme of selecting a member as the representative vector is illustrated in Fig. 10.15. It is assumed that it is possible to compute the similarity between two items. For each item, its distances from the others are computed, and its maximal distance is selected as expressed in Eq. (10.8),

$$\max_{i \neq j} |\mathbf{x}_i - \mathbf{x}_j| \quad (10.8)$$

The item with the minimum of the maximal distance is selected as the representative one, as expressed in Eq. (10.9),

$$\mathbf{r}_k = \underset{i=1}{\operatorname{argmax}}^{|C_k|} (\max_{i \neq j} |\mathbf{x}_i - \mathbf{x}_j|) \quad (10.9)$$

Fig. 10.15 Minimum maximal distance

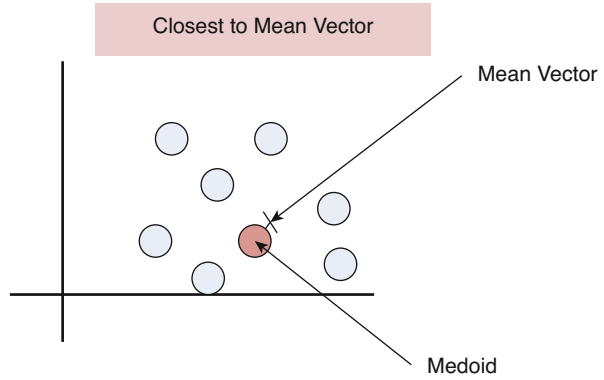
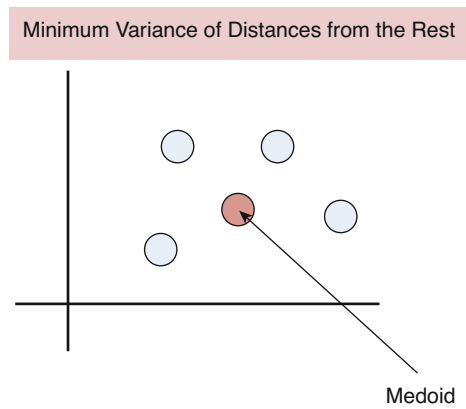


Fig. 10.16 Minimum distance variance



The vector that positions in the almost center of the cluster is selected as the cluster representative one in this scheme.

The third scheme of selecting the representative in the k medoid algorithm is illustrated in Fig. 10.16. For each item, its average distance is computed by averaging over the distances from the others, as shown in Eq. (10.10),

$$avgdist(\mathbf{x}_i) = \frac{1}{n-1} \sum_{i \neq j}^{|C_k|} |\mathbf{x}_i - \mathbf{x}_j| \quad (10.10)$$

For each item, the variance over the distances is computed by Eq. (10.11),

$$vdist(\mathbf{x}_i) = \frac{1}{n-1} \sum_{i \neq j}^{|C_k|} (avgdist(\mathbf{x}_i) - |\mathbf{x}_i - \mathbf{x}_j|)^2 \quad (10.11)$$

The item is selected as the representative one in each cluster, as expressed by Eq. (10.12),

$$\mathbf{r}_k = \underset{i=1}{\overset{|C_k|}{\operatorname{argmin}}}(vdist(\mathbf{x}_i)) \quad (10.12)$$

This scheme is intended to select most neutral item that has its most constant distances from others.

Let us make some remarks on the k medoid algorithm that was mentioned as a variant of the k means algorithm. The process of clustering data items by the k medoid algorithm is identical to that by the k means algorithm. Selecting a cluster member as the cluster prototype vector, instead of the mean vector, is the difference from the k means algorithm. So, we mentioned the three schemes of selecting a member as the cluster prototype vector, omitting the clustering process. The k medoid algorithm is applied to the clustering tasks, in the case where the similarity between items may be computed, but the average over them may not be computed.

10.4.2 Dynamic K Means Algorithm

This section is concerned with a variant of the k means algorithm that controls the number of clusters, automatically. The initial decision of the optimal number of clusters becomes an issue in applying the k means algorithm to a clustering task. The clustering index that will be described subsequently is used for finding the optimal number of clusters in this version. The optimal number of clusters is discovered by incrementing or decrementing the number of clusters, observing the clustering index. This section is intended to study the improved version of the k means algorithm that optimizes the number of clusters, automatically.

The process of computing the clustering index from the clustering results is illustrated in Fig. 10.17. The direction of clustering data items is to minimize the inter-cluster similarity and maximize the intra-cluster similarity. The average inter-cluster similarity and the average intra-cluster similarity are computed from the clustering results. The average intra-cluster similarity and the minus of the average inter-cluster similarity from 1.0 as its reverse are combined into the clustering index. The clustering index will be studied with respect to its computation process in Chap. 12.

The process of clustering items by the k means algorithm with the gradual increment of the number of clusters is illustrated as a pseudo code in Fig. 10.18. After defining the evaluation metric of the current clustering results that is presented conceptually in Fig. 10.17, we need to search for the optimal number of clusters in using the k means algorithm. The data items are clustered with the k clusters and $k + 1$ clusters, and both results are evaluated. If the results in the $k + 1$ clusters are better, the iteration is continued. The hill climbing is used for optimizing the number of clusters in using the k means algorithm.

The multiple k means algorithms that are given parallel are illustrated in Fig. 10.19. They are discriminated by the number of clusters. The k means

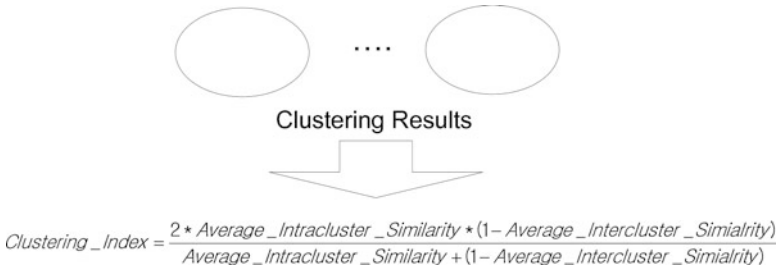


Fig. 10.17 Clustering index

```
clusterItemListByIncrementalKMeans(List itemList){
    return recursiveClusterItemListByKMeans(itemList, 2);
}
recursiveClusterItemListByKMeans(List itemList, int clusterNumber){
    clusterList1 = clusterItemListbyKMeans(itemList, clusterNumber);
    clusterIndex1 = clusterList1.getClusterIndex();
    clusterList2 = clusterItemListbyKMeans(itemList, clusterNumber+1);
    clusterIndex2 = clusterList2.getClusterIndex();
    if(clusterIndex1 >= clusterIndex2)
        return clusterList1
    return recursiveClusterItemListByKMeans(itemList, clusterNumber + 1);
}
```

Fig. 10.18 Increment and decrement of cluster numbers

algorithms are executed, independently, in parallel, and their cluster results are generated. Each of the results is evaluated with the clustering index, and one with its maximal clustering index is selected. The multiple k means algorithms are recommended as the scheme of finding the optimal number of clusters in the parallel and distributed platform.

Let us make some remarks on the dynamic k means algorithm that is mentioned in this section. In using the k means algorithm, the number of clusters should be decided as its limit. Because it is necessary to explore the multiple clustering results in the dynamic k means algorithm, it takes much more time for implementing it. It takes much more resource in implementing the multiple k means algorithm with their different numbers of clusters, parallel. The integration of the results from multiple clustering algorithms was previously tried [2].

10.4.3 Semi-supervised Version

This is concerned with the semi-supervised version of the k means algorithm. It is initially designed for clustering data items as an unsupervised learning algorithm. The unsupervised learning algorithms may be modified into their supervised version

Fig. 10.19 Parallel multiple versions

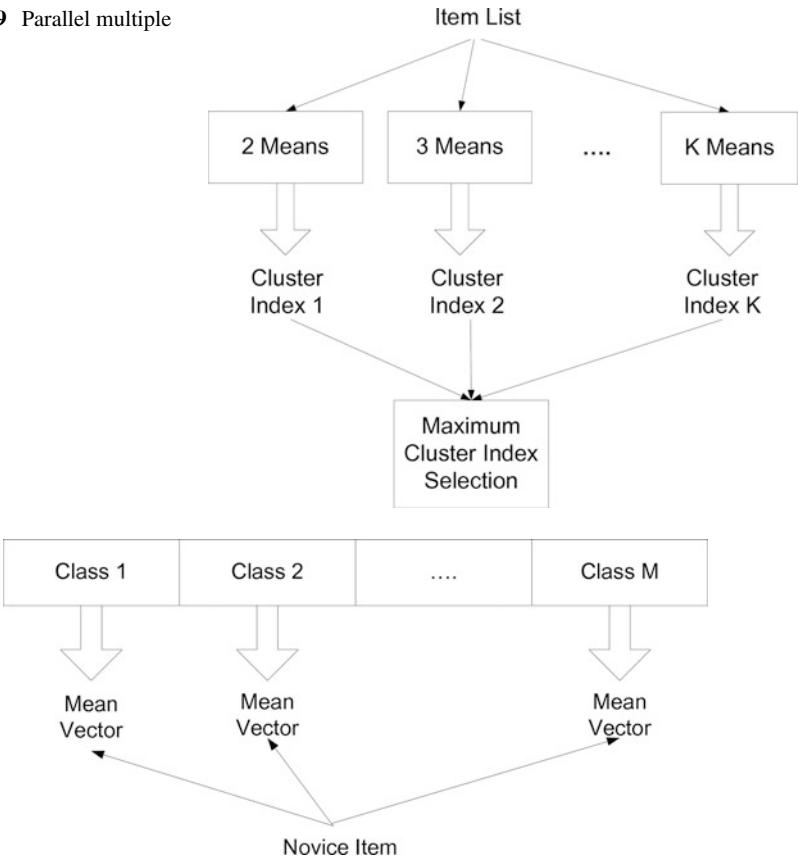


Fig. 10.20 Supervised version of the k means algorithm

more easily than that from the supervised learning into the unsupervised learning. Once it is possible to convert into the supervised version, it is utilized as the semi-supervised version. This section is intended to describe the modification of the k means algorithm into the supervised version and its utilization as the semi-supervised version.

The supervised version of the k mean algorithm is illustrated in Fig. 10.20. The labeled examples are prepared in this case. The mean vectors are computed category by category, instead of clusters. A novice example that is given subsequently is classified based on its similarities with the mean vectors. The classification process by the supervised version of the k means algorithm is same as that by the Bayes classifier that is covered in Sect. 6.2.

The process of constructing the initial clusters in using the semi-supervised version of the k means algorithm is illustrated in Fig.10.21. We previously mentioned the labeled examples and the unlabeled examples that are prepared for the semi-supervised learning. The initial clusters are constructed with the labeled

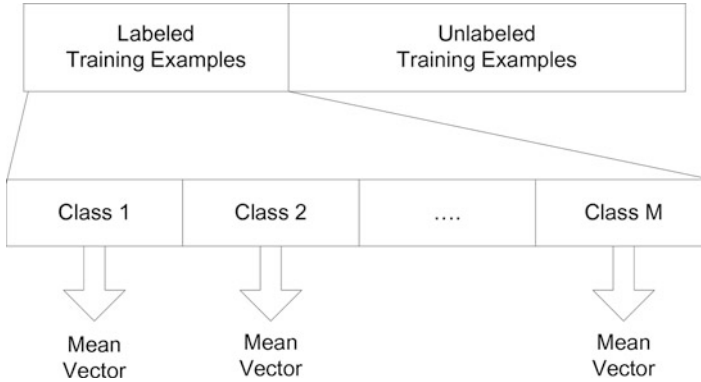


Fig. 10.21 Initial mean vectors from labeled examples

examples, as presented in Fig. 10.21. The initial clusters are constructed by selecting unlabeled examples at random in the case of the unsupervised learning. It is possible to implement the semi-supervised learning by modifying an existing unsupervised learning algorithm.

Updating the mean vectors in the clusters using the unlabeled examples is illustrated in Fig. 10.22. The initial clusters are constructed by the originally labeled examples with the assumption of no empty category. The unlabeled examples are arranged depending on their similarities with the mean vectors, and for each cluster, its mean vector is updated. The two steps are iterated until the mean vectors converge. The unlabeled training examples are classified by iterating them, based on the labeled examples.

Let us make some remarks on the semi-supervised version of the k means algorithm that is covered in this section. It is possible to modify the unsupervised learning algorithm into its supervised version; the types of the machine learning algorithms are defined depending on their initial intention. Once it is possible to convert the unsupervised learning algorithms into their supervised versions, the unsupervised learning algorithms may be also converted into their semi-supervised versions. The semi-supervised learning is intended to solve the insufficient number of the labeled training examples by filling the unlabeled ones. The semi-supervised learning will be studied in detail in Chap. 14.

10.4.4 Constraint Clustering

The constraint clustering is defined as the process of clustering data items based on some labeled examples. Both the labeled examples and the unlabeled ones coexist like the case in the semi-supervised learning. The semi-supervised learning is intended for performing the classification and the regression, whereas the constraint

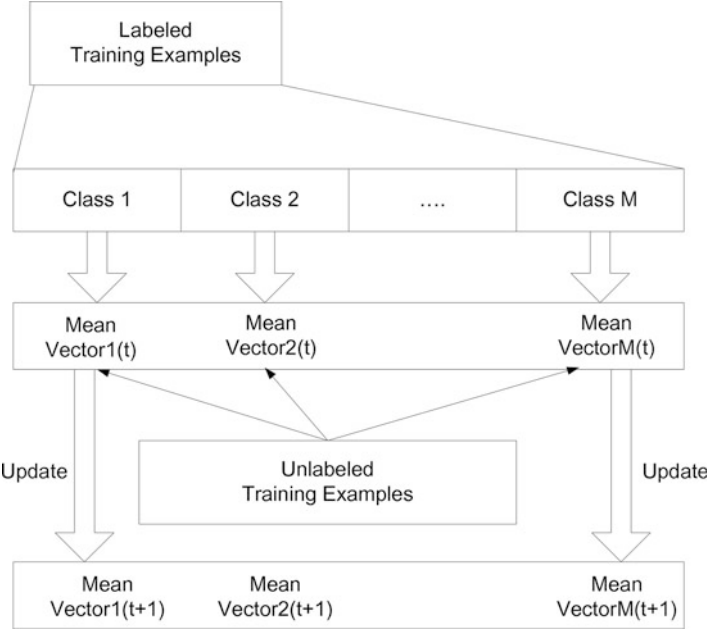


Fig. 10.22 Update of mean vectors by unlabeled examples

clustering is intended for performing the clustering. A very sparse number of labeled examples are usually available in the constraint clustering, compared with the number of unlabeled examples. This section is intended to describe the constraint clustering that is distinguished from the semi-supervised learning.

The process of constructing the initial mean vectors from the labeled examples and unlabeled examples is illustrated in Fig. 10.23. We need to open the possibility of additional clusters to the clusters of the labeled examples. The initial clusters are constructed by the labeled examples, and the additional ones are constructed by selecting several examples from the unlabeled ones as their mean vectors. The K clusters are built by the labeled examples, and the $M - K$ clusters are built by the unlabeled examples, as in Fig. 10.23. The reason of building the additional clusters is the assumption of the sparse number of labeled examples, compared with the number of the unlabeled ones.

The process of updating the mean vectors after arranging the unlabeled examples is illustrated in Fig. 10.24. The labeled examples are arranged as the constraints into the clusters that correspond to their target labels. The unlabeled examples are arranged by their similarities with the clusters; each unlabeled example joins into the cluster whose mean vector is most similar. The mean vector is updated by arranging the labeled examples and the unlabeled examples by averaging the members for each cluster. The unlabeled examples are rearranged into clusters, based on the updated mean vectors.

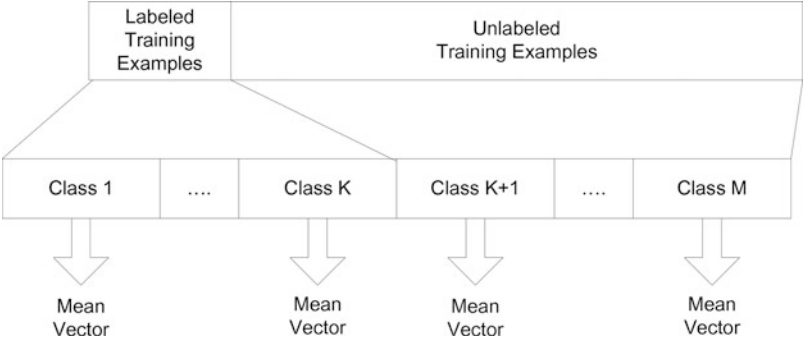


Fig. 10.23 Initial mean vectors

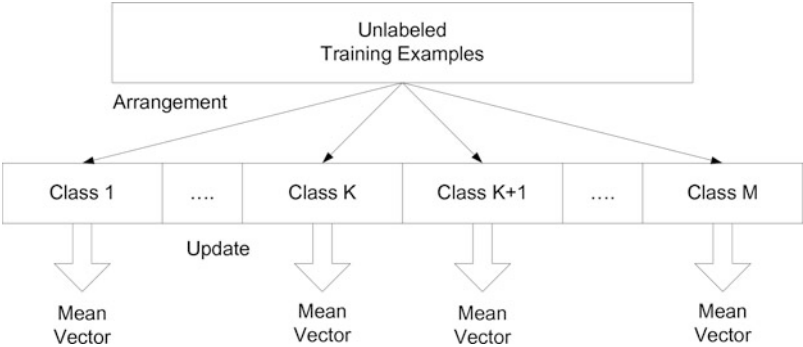


Fig. 10.24 Update of mean vectors

The semi-supervised learning and the constraint clustering are illustrated in Fig. 10.25, for comparing them with each other. The labeled examples and the unlabeled ones are used in the both paradigms, but they are different from each other with respect to their goals. The originally unlabeled examples are labeled through the clustering algorithm and added to the set of the originally labeled examples for training the supervised learning algorithm, in the semi-supervised learning. In the constraint clustering, the initial clusters are constructed using the labeled examples and the clusters are updated using the unlabeled ones. The semi-supervised learning is intended for the classification and the regression, and the constrained clustering is intended for clustering data items.

Let us make some remarks on the constraint clustering that looks as the mixture of the supervised learning and the unsupervised learning. In both the semi-supervised learning and the constraint clustering, the labeled examples and the unlabeled ones are used. The semi-supervised learning is intended for the classification and the regression, whereas the constraint clustering is intended for clustering data items. In the constraint clustering, we consider the possibility of

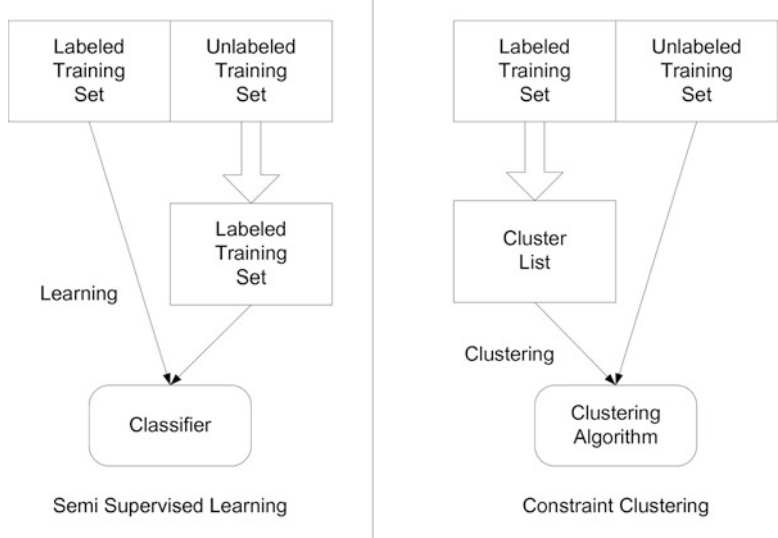


Fig. 10.25 Comparison with semi-supervised learning

more clusters as the addition to those that are presented in the labeled examples. We also consider the noise in the labeled examples for building the initial clusters.

10.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We pointed out the possibility of transition between the supervised learning and the unsupervised learning by presenting the case of KNN algorithm. We mentioned the k means algorithm as the most popular clustering algorithm and applied it to the three types of clustering: the hard clustering, the fuzzy clustering, and the hierarchical clustering. We mentioned the k medoid algorithm and the dynamic k means algorithm as the variants and modified the k means algorithm into the semi-supervised version and the constraint clustering version. In this chapter, we studied the transition between two types of machine learning, the standard version of the k means algorithm, and its variants.

The means vectors or ones that represent their own clusters are called representatives. The selection of initial mean vectors or initial representatives becomes the issue in using the k means algorithm. It requires the diversity among initial mean vectors for doing clustering robustly. The selection of two or more similar vectors as the initial representative ones is the case of separating items that should belong to same cluster into different clusters. We need to check similarities among initial representatives before proceeding next steps.

In this chapter, we mentioned the k medoid algorithm as well as the k means algorithm, considering various schemes of selecting representatives. The mean vector is given as a cluster representative vector in the k means algorithm. In the k medoid algorithm, we mentioned the various schemes of selecting a member as the representative vector. We need to remove outliers in each cluster and compute the mean vector for implementing more smoother version. Depending on the scheme of selecting cluster representative vectors, various versions of the k means algorithm are derived.

Arranging data items into clusters depends on the similarities between vectors. We consider using multiple similarity metrics for doing so. The similarities with the cluster mean vectors are computed for each item in order to decide which of the clusters it is arranged to. The similarities are computed using multiple metrics, and they are integrated into a metric by averaging them. Using multiple similarity metrics is intended to minimize the bias toward a particular cluster.

The k means algorithm is viewed as the simplified version of the EM algorithm that will be studied in the next chapter. The EM algorithm refers to the type of clustering algorithm that performs the clustering by the two steps: the estimation step and the maximization step. The E (Estimation) step is the computation of mean vectors for each cluster, and the M (Maximization) step is the arrangement of each item into its corresponding cluster whose mean vector is most similar, in clustering data items by the k means algorithm. In generalizing the E-step and the M-step, the former is to characterize each cluster and the latter is to define cluster membership values for each item. In Chap. 11, we will study the general version of the EM algorithm in detail.

References

1. T. Jo, The implementation of dynamic document organization using text categorization and text clustering, PhD Dissertation, University of Ottawa, 2006
2. Z. Zhou, W. Tang, Cluster ensemble. *Knowl. Based Syst.* **19**(1), 77–83 (2006)

Chapter 11

EM Algorithm



11.1 Introduction

The EM algorithm is, rather than a specific clustering algorithm, the clustering frame which clusters data items with the two steps: E-step and M-step. The E-step means the process of estimating the probabilities the likelihoods of individual items to clusters, assuming that each cluster has its own probability distribution. The M-step is the process of estimating the parameters of cluster distributions based on the likelihoods which are estimated in the E-step. The E-step and the M-step are iterated until the convergence of the distribution parameters from the M-step and the likelihoods from the E-step, in the EM algorithm. This section is intended to describe the basic concepts, which are necessary for understanding the EM algorithm.

The distributions over clusters should be determined in applying the EM algorithm. For each cluster, a normal distribution is usually assumed. As its parameters, its mean vector and its covariance matrix are estimated in the normal distribution of each cluster. The probability that each item belongs to the cluster is estimated based on the normal distribution. By using the EM algorithm, we may consider other distributions such as a uniform distribution, a triangle distribution, and a trapezoid distribution.

The cluster-item matrix where its columns correspond to clusters and its rows correspond to items. Each element in the cluster-item matrix corresponds to the conditional probability of cluster given an item, $P(C_i|\mathbf{x}_j)$. The E-step in the EM algorithm is to estimate the elements in the cluster-item matrix. The conditional probability is computed under the assumption of cluster distributions such as normal distributions. The row which corresponds to a data item in the cluster-item matrix in the k means algorithm has one value in only one cluster and zero values to the others.

The parameters of each clustering distribution are estimated based on the cluster memberships of examples in the M-step. The membership of each example in each cluster is decided by the probability of the cluster given the example, $P(C_i|\mathbf{x}_j)$,

which is estimated in the E-step. Each cluster is assumed as a normal distribution, and the mean vector and the covariance matrix are computed by the memberships of examples. The iteration of estimating the memberships and the parameters is the clustering process of the EM algorithm. The initialization of cluster distribution parameters at random is called the initial M-step.

The goal of this chapter is to understand the EM algorithm which is given as a clustering frame, rather than a clustering algorithm. We understand various kinds of distributions which are assumed as cluster distributions, before applying the EM algorithm. We need to understand the process of clustering items using the EM algorithm and its issues. We will also understand the process of applying the EM algorithm as the semi-supervised learning as well as the unsupervised learning. We will make the further discussions on derivation of specific versions from the EM algorithm and using it as the semi-supervised learning.

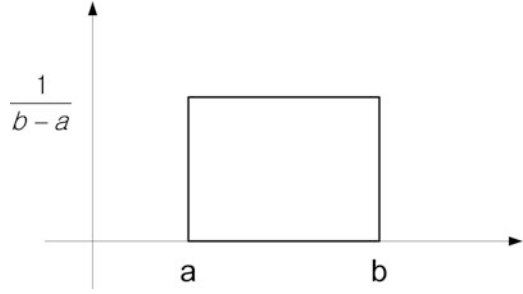
11.2 Cluster Distributions

This section is concerned with the distributions which are assumed before executing the EM algorithm. In Sect. 11.2.1, we mention the simplest clustering distribution, called uniform distribution. In Sect. 11.2.2, we describe the normal distribution which is the most popular clustering distribution by using the EM algorithm. In Sects. 11.2.3 and 11.2.4, we mention the Poisson distribution and the fuzzy distribution as the alternative ones. This section is intended to explore some cluster distributions which are assumed for specifying the EM algorithm.

11.2.1 Uniform Distribution

The uniform distribution is defined as the probability distribution which is constant within a range. It is the simplest probability distribution where events are given continuously. The start value and the end value are given as the parameters in the uniform distribution. We may consider some variants in addition. This section is intended to define the uniform distribution as the most continuous probability distribution and mention its some variants.

The continuous probability distribution is the distribution over probabilities to the continuous random variable. The random variable is one for storing the numerical value which expresses an event, and when the random variable is discrete, the distribution over probabilities is called discrete probability distribution. The finite number of probabilities which correspond to events are given in the discrete probability distribution, whereas the infinite number of probabilities are given as a continuous line or curve in the continuous probability distribution. Assumption of continuous probability distribution to each cluster in the EM algorithm is the reason of focusing on continuous probability distributions.

Fig. 11.1 Uniform distribution

Let us consider the parameters which characterize the uniform distribution as shown in Fig. 11.1. The start value, a , and the end value, b , are the parameters of the uniform distribution in the case of scalar values. $\frac{1}{b-a}$ is the constant probability between a and b , and zero is the probability out of the interval. In the case of vectors, $\mathbf{a}$ is the start vector, $\mathbf{b}$ is the end vector, and $\frac{1}{\|\mathbf{b}\| - \|\mathbf{a}\|}$ is the probability within the hypercube. The parameters of the uniform distribution should be so as shown in Eq. (11.1),

$$a \leq b, \|\mathbf{a}\| \leq \|\mathbf{b}\| \quad (11.1)$$

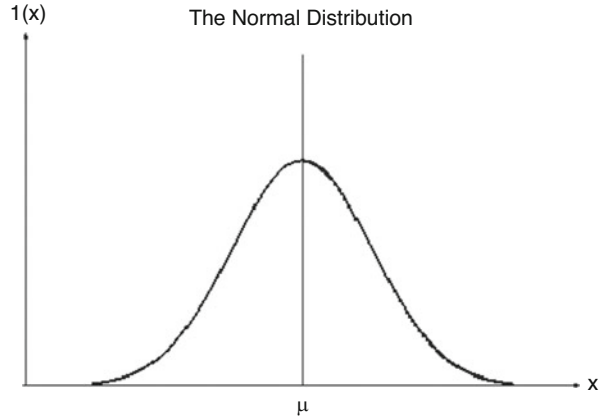
Let us consider some variants which are derived from the uniform distribution by modifying it. The variant may be the dual uniform distribution where the parameters are given as $a < b < c < d$, and the probabilities are given as Eq. (11.2),

$$\frac{1}{b-a} = \frac{1}{d-c} < \frac{1}{c-b} \quad (11.2)$$

We may consider the trapezoid distribution as an instance of the fuzzy distribution with the parameters, $a < b < c < d$. One more variant is the triangle distribution with the parameters, $a < b < c$ with the maximal probability in the point, b . Variable probabilities within an interval may be considered as the variant of the uniform distribution.

Let us make some remarks on the uniform distribution as the instance of the continuous probability distribution. In the uniform distribution, it is assumed that the probability is constant over all continuous events within an interval. The area of all probability within the interval becomes 1.0. In each cluster, the parameters of uniform distribution are estimated for clustering data items. In defining data clusters the normal distribution which will be mentioned subsequently is preferred to the uniform distribution.

Fig. 11.2 Gaussian distribution



11.2.2 Gaussian Distribution

The normal distribution or the Gaussian distribution is defined as the bell-shaped probability distribution where the probability is maximal in the average, as shown in Fig. 11.2. In the statistics, it is assumed that measures in the population follow the Gaussian distribution. The mean and the variance are the parameters of the Gaussian distribution, and the probability is estimated to a particular measure, using the Gaussian table. The Gaussian distribution is used for estimating the population mean using the sample mean and making the hypothesis test whether the sample values indicate the population ones, or not, in the statistics. This section is intended to characterize the Gaussian distribution as the most popular continuous probability distribution.

Let us mention the standard Gaussian distribution as the simplest version. The mean is zero and the variance is one as its parameters. In the case of vector, the mean vector is given as the zero vector, and the covariance matrix is given as one where its diagonal elements are ones and its off-diagonal ones are zeros. When expressing the vector values as the standard Gaussian distribution, it is assumed in the elements in the vector as independent ones. In order to estimate the probability of the value interval, we need to convert any Gaussian distribution into the standard form.

Let us mention the parameters which express the Gaussian distribution. The mean, μ , and the variance, σ , are the parameters of the scalar Gaussian distribution. The parameters of the vector Gaussian distribution are the mean vector, $\boldsymbol{\mu}$, and the covariance matrix, $\boldsymbol{\Sigma}$. When the covariance matrix is given as a diagonal matrix, the attributes in the vector are independent of each other. The assumption of the independent attributes is the popular trend in defining the clusters as Gaussian distributions for the simplicity.

Let us mention the process of estimating a value probability based on the Gaussian distribution. It is characterized by the mean and the variance as Eq. (11.3),

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma^2} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad (11.3)$$

The probability of $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]$ in the vector valued Gaussian distribution whose parameters are the mean vector, $\boldsymbol{\mu}$, and the covariance matrix, $\boldsymbol{\Sigma}$, is expressed as Eq. (11.4),

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \quad (11.4)$$

It is complicated to compute the probability of the value, using the above equation, so the Gaussian distribution is converted into the standard one, and the probability is estimated using the standard Gaussian distribution table. Refer to [1] about the statistics for getting the detail description of estimating the probability based on the Gaussian distribution.

Let us make some remarks on the Gaussian distribution which is a continuous probability distribution. The vector based Gaussian distribution is characterized by the two parameters: the mean vector and the covariance matrix. The probability of a particular vector is reversely proportional to the distance from the mean vector. The probability is decreased sharply from the mean vector in the small determinant of the covariance matrix, but it decreases slowly in the large determinant. The mixture of various continuous probability distribution converges to the Gaussian distribution by the central limit theorem.

11.2.3 Poisson Distribution

This section is concerned with the Poisson distribution which is defined as another cluster distribution. We consider the binomial distribution which defines the probability of m trials with the success among n trials, before doing the Poisson distribution. In the distribution, the total trials are assumed to be infinite and the probability of event times is defined. In the EM algorithm, the probability of each item which is its cluster membership value is estimated based on the Poisson distribution. This section is intended to describe the binomial distribution, the derivation of the Poisson distribution from it, and the parameters of the distribution.

Let us mention the binomial distribution before explaining the Poisson distribution. The number of same trials is n , and the probability of the success in each trial is p . The probability of making m trials successful is expressed into Eq. (11.5),

$$\binom{n}{m} p^m (1-p)^{n-m} \quad (11.5)$$

The probability of doing more than or equal to m trials successful is expressed into Eq. (11.6),

$$\sum_{k=m}^n \binom{n}{k} p^k (1-p)^{n-k} \quad (11.6)$$

For example, probability of coin with m heads in tossing it n times is expressed into Eq. (11.7),

$$\binom{n}{m} \left(\frac{1}{2}\right)^n \quad (11.7)$$

Let us mention the process of deriving the Poisson distribution based on the binomial distribution. The number of trials, n , is assumed to be set infinite, $n \rightarrow \infty$, the expectation is set, $np \rightarrow \mu$, as a parameter, and the number of successful cases is k . The probability distribution in the Poisson distribution is expressed in Eq. (11.8),

$$P(k) = \frac{\mu^k e^{-\mu}}{k!} \quad (11.8)$$

The probability of no successful case, $P(0)$, is the highest values 0.6 and 0.4, respectively, in $\mu = 0.5$ and $\mu = 1.0$. The higher probability of successful events is close to mean, μ , in the Poisson distribution.

Let us mention the process of estimating the probability as a clustering membership value to the given numerical vector. The Poisson distribution is applied feature by feature, μ_i is i th element of the mean vector, $\boldsymbol{\mu}$ and x_{ij} is i th element of the vector which represents a particular item, $\mathbf{x}_1$. The probability of the element, x_{ij} , is computed using Eq. (11.9),

$$P(x_{1i}) = \frac{\mu_i^{x_{1i}} e^{-\mu_i}}{[x_{1i}]!} \quad (11.9)$$

where $[x_{1i}]$ is an integer value from approximating the value. The probability of item, $\mathbf{x}_1$, is estimated by averaging the probabilities of elements as shown in Eq. (11.10),

$$P(\mathbf{x}_1) = \frac{1}{d} \sum_{i=1}^d P(x_{1i}) \quad (11.10)$$

We need to scale the attributes values which are given as normalized values or tiny scaled ones, into ones between one and ten.

Let us make some remarks on the Poisson distribution which is covered in this section. It is derived from the binomial distribution by setting the expectation as

the parameter, assuming the infinite number of trials. In the Poisson distribution, a random variable, x , is given as an integer that belongs to the discrete probability distribution. The fact that elements of a numerical vector which represents a data item are given as continuous values is the reason of modifying Eq. (11.8) into Eq. (11.9). The additional assumption in applying the Poisson distribution to EM algorithm is that the numerical vector features are independent of each other.

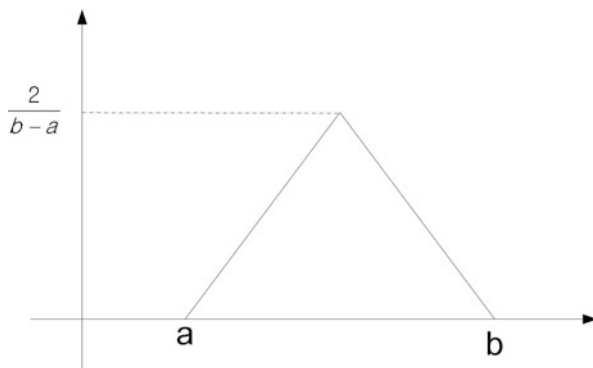
11.2.4 Fuzzy Distributions

This section is concerned with another type of distribution over a cluster, called fuzzy distribution. In the previous sections, we studied the two continuous probability distributions, the uniform distribution and the Gaussian distribution, and a discrete probability distribution, the Poisson distribution. In this section, we cover the triangle distribution and the trapezoid distribution as the fuzzy distributions. We also mention the process of computing a probability from multiple distributions over each cluster. This section is intended to describe the two typical continuous probability distributions, the triangle distribution and the trapezoid distribution.

The triangle distribution as an instance of the fuzzy distribution is illustrated in Fig. 11.3. The start value, a , and the end value, b , are given as the parameters of the triangle distribution. The probability is peak in the midpoint, $\frac{b-a}{2}$, and the probability is zero out of a and b . The probability decreases linearly in the both directions, left and right from the midpoint. The parameters of this distribution are same to those of the uniform distribution in spite of the difference between the two distributions.

The trapezoid distribution as another instance of the fuzzy distribution is illustrated in Fig. 11.4. The four parameters, the start point, a , the start point of the highest probability, b , the end point of the highest probability, c , and the end point, d , are involved in the distribution. The highest probability, $\frac{1}{d-a+b-c}$, is in the interval between b and c , the probability is increased from a to b , and the probability

Fig. 11.3 Triangle distribution



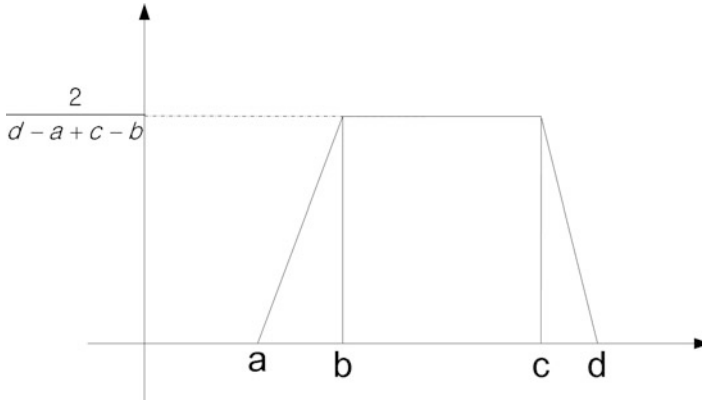


Fig. 11.4 Trapezoid distribution

is decreased from c to d . In the triangle distribution, the highest probability exists in the single point, whereas in the trapezoid distribution, the highest probability spans as an interval. The Gaussian distribution which was covered in Sect. 11.2.2 is used as a fuzzy distribution as well as the two distributions.

The hybrid distributions with their mixtures of two distribution are illustrated in Fig. 11.5. The top in Fig. 11.5, shows the mixture of the uniform distribution and the Gaussian distribution which is close to the student t distribution. In the middle of Fig. 11.5, the mixture of the triangle distribution and the Gaussian distribution which look similar as each other is shown. In the bottom of Fig. 11.5, two kinds of fuzzy distributions, the triangle distribution and the trapezoid distribution, are combined with each other. Almost infinite number of different distribution converges to the Gaussian distribution by the central limit theorem.

Let us make some remarks on the fuzzy distributions which are mentioned in this section. The fuzzy distributions are used for modeling fuzzy sets or fuzzy values, rather than for modeling data clusters. The fuzzy distributions are similar as the Gaussian distribution, and mixture of many various distributions converges to the Gaussian distribution by the central limit theorem, so the Gaussian distribution is usually adopted for implementing the EM algorithm. The fuzzy distributions are characterized in the traditional fuzzy system, depending on the prior knowledge, and in the neuro-fuzzy system, they are done by the neural networks, automatically. Because there are various versions of EM algorithm, depending on which of distribution is defined to each cluster, EM algorithm exists as a clustering frame, rather than a specific clustering algorithm.

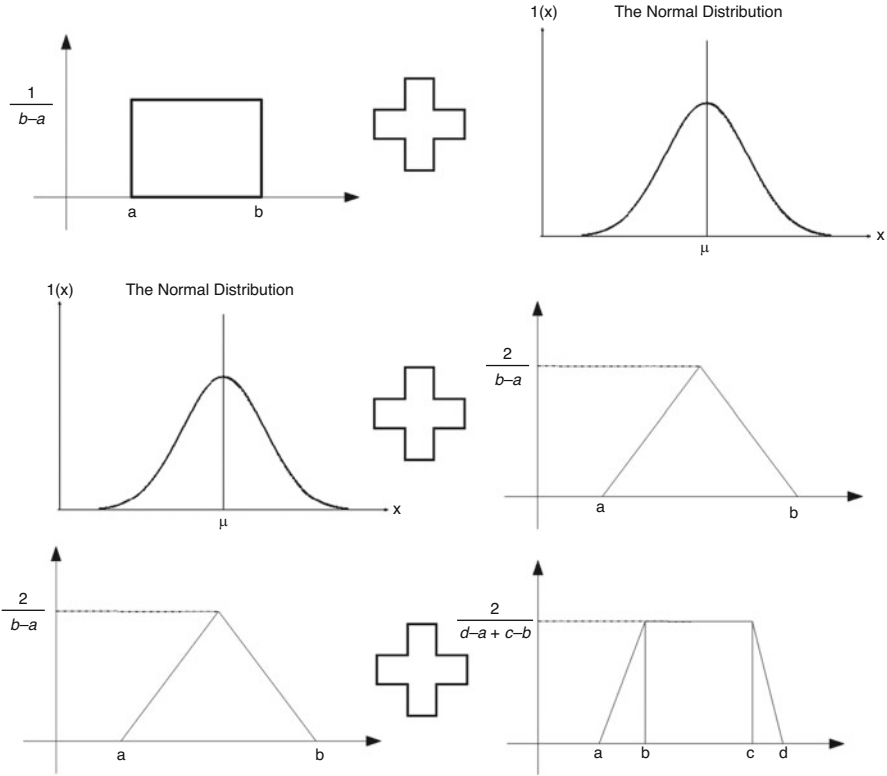


Fig. 11.5 Multiple distributions

11.3 Clustering Process

This section is concerned with the frame of clustering items by the EM algorithm. In Sect. 11.3.1, we explain the initialization for using the EM algorithm for clustering data items. In Sects. 11.3.2 and 11.3.3, we describe the two essential steps, the E-step and the M-step, respectively. In Sect. 11.3.4, we mention some issues in clustering data items by the EM algorithm. This section is intended to describe the two steps, as the frame of clustering data items.

11.3.1 Initialization

This section is concerned with the process of initializing clustering in the EM algorithm. The cluster mean vectors are initialized by selecting data items at random as many as clusters, as mentioned in Sect. 10.3.1, as the case of the k means

algorithm. The steps of initializing clusters are to decide the number of clusters, to decide the probability distribution for each cluster, and to initialize distribution parameters for each distribution. The parameters of the probability distribution which is defined to each cluster are results from initializing clusters in the EM algorithm; this initialization is called initial M step. This section focuses on the initial M step and the subsequent sections cover the subsequent M step.

It is required to decide the number of clusters in advance for using the EM algorithm for clustering data items. The number of clusters is usually decided at random or by intuition. In clustering two or three dimensional vectors, we may decide the optimal number of clusters by plotting them in the two or three dimensional space. Because it is impossible to plot more than three dimensional vectors, it is difficult to decide the optimal number of clusters. The decision of the optimal number of clusters is dependent on the prior knowledge in some applications.

The next step of preparing for using the EM algorithm for clustering data items is to decide the distribution for each cluster. The number of clusters was already decided in the previous step. For each cluster, we should decide the distribution among ones which were covered in Sect. 11.2. In general, each cluster is assumed to be a Gaussian distribution. There are various versions of EM algorithm, depending on which of the probability distribution is assumed for each cluster.

The next step of the distribution decision is to initialize the parameters which characterize the distributions. It is assumed that the training examples are unlabeled by using the EM algorithm. Assuming that each cluster is defined as its own Gaussian distribution, the mean vectors and the covariance matrices are set at random. By using the k means algorithm, it is assumed that each cluster is a Gaussian distribution with its own mean vector and its constant covariance matrix. The k means algorithm which is studied in Chap. 10 is the simplified version of the EM algorithm.

Let us consider some issues in setting up the EM algorithm. The first issue is how to decide the optimal number of clusters in advance. As the second issue, we must decide the probability distributions over the clusters; The Gaussian distribution is usually assumed for each cluster. In the case of the Gaussian distribution, we need to configure the initial parameters such as the mean vectors and the covariance matrices. We need to consider the dichotomies on clustering types: hard vs soft clustering and flat vs hierarchical clustering.

11.3.2 *E-Step*

This section is concerned with the E-step of the EM algorithm. The E-step where E means estimation is defined as the process of estimating cluster membership values for each item. Each cluster is assigned as a Gaussian distribution which is characterized by its mean vector and its covariance matrix, and the likelihoods of each item to the clusters are computed as its cluster membership values. Only mean

Fig. 11.6 E step in K means algorithm

	Cluster 1	Cluster 2		Cluster K
Item 1	0	1		0
Item 2	1	0		0
....				
Item N	0	0		1

vector is used for characterizing the Gaussian distribution in viewing the k means algorithm as an EM algorithm. This section is intended to describe the E-step in the specific view of the k means algorithm and the abstract view.

Let us mention the E-step of the k means algorithm before mentioning the E-step of the EM algorithm, and the results from the E-step of the k means algorithm are illustrated in Fig. 11.6. The data clustering is assumed as the process of segmenting a group of N items into K clusters; each column corresponds to a cluster, and each row corresponds to a data item. In defining a membership value, 1.0 indicates that it belongs to the cluster completely, and 0.0 does that it does not belong to the cluster at all. Each row which indicates a data item has only one 1.0; each data item belongs to only one cluster. Figure 11.6 illustrates the results by applying the k means algorithm to the hard clustering.

Let us consider the E-step in the general version of the EM algorithm. The E-step in the k means algorithm is described as a specific process, whereas one in the general version will be described as a frame. A particular probability distribution is assumed for each cluster, and for each item, its membership values of the clusters are estimated as the probability of the item belonging to the cluster, as shown in Eq. (11.11),

$$\mu_{C_i}(\mathbf{x}) = P(C_i|\mathbf{x}) \quad (11.11)$$

The membership value is approximated by the likelihood of item to the cluster as shown in Eq. (11.12),

$$P(C_i|\mathbf{x}) \approx P(\mathbf{x}|C_i) \quad (11.12)$$

according to the Bayes rule, under the assumption of identical portions of the categories, $P(C_1) = P(C_2) = \dots = P(C|C|)$. The likelihoods of the data item to the clusters are results from the E-step.

The E-step is explained under the assumption that all clusters are given as their own Gaussian distribution. Each Gaussian distribution is characterized as the mean vector, μ_i , and the covariance matrix, Σ_i . The likelihood of the item, $\mathbf{x}$, to the cluster, C_i , is computed by Eq. (11.13),

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i)\right) \quad (11.13)$$

The similarity as the mean vector is given as a normalized value between zero and one in considering only mean vectors, ignoring the covariance matrices. The covariance matrix is defined for each calculation as considering the correlations among attributes.

Let us make some remarks on the E-step in the general version of the EM algorithm. It is required to define the probability distribution over each cluster for using the EM algorithm. The likelihood of each item to a cluster is given as its membership value of the cluster. The E-step is to construct the item–cluster matrix whose elements are membership values. After the E-step, the parameters of the probability distributions over the clusters are updated by the membership values.

11.3.3 *M-Step*

This section is concerned with the M-step of the EM algorithm. In Sect. 13.3.2, we studied the E-step which is the step of estimating the cluster membership values or the likelihoods to the clusters for each item. The M-step which is covered in this section is the step of updating the parameters of the probability distribution over clusters for maximizing the likelihoods. The maximization of the likelihoods of items to the clusters is the reason of calling this step M-step. This section is intended to describe the M-step as the second main step of the EM algorithm.

Let us mention the M-step of the k means algorithm which was described in Chap. 10. The M-step is defined above as the process of estimating cluster membership values for each item. For each item, its membership values are one or zeros by using the k means algorithm for the hard clustering. One is assigned to the cluster whose mean vector is most similar as the item, and zeros are assigned to the others. Each item is arranged into its own cluster as the M-step in the k means algorithm.

Let us explain the M-step of the EM algorithm as its general form. It is assumed that the clusters have their own probability distributions among ones which were covered in Sect. 11.2, and the cluster membership values are estimated in the E-step for each item. For each cluster, the probability distribution parameters are computed, based on the cluster membership values which are assigned to each item. The initialization of the parameters in starting the EM algorithm is called initial M-step. The M-step is to update the probability distributions over the clusters.

Let us explain the M-step more specifically, assuming the Gaussian distribution for each cluster. The parameters of the Gaussian distribution are the mean vector and the covariance matrix. The mean vectors are computed by the process that is described in Sect. 10.3.3, and the covariance matrix is computed by finding the variances of the elements and correlations among them. Each cluster is characterized with the mean vector and the covariance matrix in the M-step. The fuzzy k means algorithm which is covered in Sect. 10.3.3 is viewed as the EM algorithm where each cluster is assumed as the Gaussian distribution and only mean vectors are considered for the simplicity.

Let us make some remarks on the M-step of the EM algorithm in clustering data items. Each cluster is assumed as a probability distribution which was described in Sect. 11.2. The M-step is the process of estimating the parameters of the probability distribution over each cluster. The Gaussian distribution is adopted as the probability distribution over each cluster in clustering data items by the EM algorithm. For each cluster, multiple Gaussian distributions may be defined in some variants of EM algorithm.

11.3.4 Issues

This section is concerned with some issues in using the EM algorithm for clustering data items. It is required to decide the number of clusters and define the probability distribution for each cluster for using the EM algorithm. The issue is that it is impossible to find the optimal number of clusters and the optimal parameters of the probability distributions, in advance. The data items are clustered based on the EM algorithm, depending strongly on the initial parameters of the probability distribution. This section is intended to point out some issues in clustering data items by the EM algorithm and discuss the solutions to them.

The decision of the optimal number of clusters is pointed out as the issue in using the EM algorithm as the approach to the data clustering. There is no way of knowing the optimal number of clusters, in advance, except plotting two dimensional or three dimensional vectors in the visible spaces. The EM algorithm is used under the arbitrary decision of the number of clusters. The clustering index was initially proposed in 2006 by Jo as the evaluation metric of the clustering results and was utilized for tuning parameters by Jo [2, 3]. The solution to the issue is to cluster data items by the EM algorithm with different numbers of clusters and select the clustering results with its best clustering index.

Let us consider which of the distribution is defined for each cluster as one more issue in using the EM algorithm. We previously mentioned the EM algorithm as not specific algorithm but a frame of clustering data items. In general, the Gaussian distribution which is characterized by its mean vector and its covariance matrix is defined for each cluster. The fuzzy k means algorithm is an instance of the EM algorithm where the Gaussian distribution is defined for each cluster and only its mean vector is counted. By using the EM algorithm, multiple Gaussian distributions may be considered for each cluster.

Let us consider the quality of the training examples in clustering data items. The training examples never have actually identical qualities; some training examples should be removed as noises, or different weights should be assigned to the individual ones. In the E step, the cluster membership values are estimated for each item and the parameters of the probability distribution are estimated in the M step, considering the weights of the training examples. Higher weights are assigned to the training examples inside each cluster, and lower weights are assigned to ones in

the boundary of each one, conceptually. It is possible to compute the weights of the training examples through the outlier detection.

Let us consider the EM algorithm by applying it to the hierarchical clustering. In Chap. 10, we pointed out the issue in using the k means algorithm for such kind of clustering. It is required to decide the number of nested cluster to every cluster for using the EM algorithm for performing the hierarchical clustering. The AHC algorithm and the divisive clustering algorithm are usually used for the hierarchical clustering. It is required to define the probability distributions even in the nested clusters for using the EM algorithm.

11.4 Semi-Supervised Learning: Text Classification

This section is concerned with the process of applying the EM algorithm to the text classification as the semi-supervised learning. In Sect. 11.4.1, we describe the semi-supervised learning before applying it to the text classification. In Sect. 11.4.2, we explain process of initializing the probability distributions over training examples. In Sect. 11.4.3, we describe the E-step in applying the EM algorithm to the task. In Sect. 11.4.4, we mention the M-step in doing so.

11.4.1 *Semi-Supervised Learning*

This section is concerned with the semi-supervised learning as the case of applying the EM algorithm, together with the Naive Bayes. The EM algorithm was entirely described in Sect. 11.3, as a clustering frame, and the semi-supervised learning was explained conceptually in Sect. 1.3. The semi-supervised learning is intended to improve the classification performance and the regression performance by utilizing unlabeled examples. The semi-supervised learning is viewed as a specific type of the supervised learning which uses both labeled examples and unlabeled ones. This section is intended to study the roles of the EM algorithm and the Naive Bayes for implementing the semi-supervised learning.

The idea of the semi-supervised learning is motivated by utilizing unlabeled examples for improving the classification performance. In reality, the labeled examples are expensive, but the unlabeled ones are very cheap for collecting them as the training examples. In the semi-supervised learning, the machine learning algorithm is trained using both the labeled examples and the unlabeled examples. The EM algorithm and the Naive Bayes are combined with each for implementing the semi-supervised learning. It will be studied in detail in Chap. 14.

Let us consider the role of the EM algorithm in implementing the text classification system by the semi-supervised learning. Because the task is given as a classification, the number of clusters is initially decided. The Gaussian distribution

is defined for each cluster, and the mean vectors and the covariance matrices are computed from the labeled examples. Both the labeled examples and the unlabeled ones are clustered by iterating the E-step and the M-step. The role of the EM algorithm is to assign the cluster membership values to the unlabeled examples.

Let us mention the role of the Naive Bayes for implementing so. The cluster membership values are optimized for each training example, by the EM algorithm. The likelihoods of the individual attribute values are computed by the Naive Bayes. A novice item is classified by the likelihoods which are computed by it. The role of the Naive Bayes is to set the parameters for classifying data items.

Let us make some remarks on the semi-supervised learning which is implemented by combining the EM algorithm and the Naive Bayes with each other. We previously studied the two algorithms in Chap. 6 and the current chapter. This section is intended to present the case of applying the EM algorithm which is covered in this chapter to the real problem. The EM algorithm is used for implementing the semi-supervised learning by assigning the category membership values to the originally unlabeled examples. Each cluster is defined as a Gaussian distribution, in this case.

11.4.2 Initialization

This section is concerned with the initialization for doing the semi-supervised learning for the text classification. In Sect. 11.4.1, we explained the semi-supervised learning and the roles of the EM algorithm and the Naive Bayes in the conceptual view. We need the preliminary tasks, such as the preparation of the sample texts, the definition of the distributions over the categories, and the encoding of texts into numerical vectors, for doing the text classification by the semi-supervised learning. The distribution over each category is assumed to a Gaussian distribution which is characterized by its mean vector and its covariance matrix. This section is intended to study the three preliminary tasks for the text categorization through the semi-supervised learning.

Let us discuss the preparation of the sample texts for training the EM algorithm and the Naive Bayes. The categories are predefined as a flat list or a hierarchical structure. Both kinds of texts, the labeled texts and the unlabeled texts, are prepared as the sample texts. It is assumed that the labeled texts are not easy to obtain, but the unlabeled ones are easy to do. The unlabeled texts are utilized for improving the performance in the text classification which is implemented by the semi-supervised learning.

The texts are encoded into the numerical vectors for performing the text classification as the preprocessing. The corpus is indexed into words as feature candidates, and some among them are selected as features. For each text, the TF-IDF (Term Frequency and Inverse Document Frequency) weight is assigned to each feature as a feature value. The TF-IDF weight is computed by Eq. (11.14),

$$w_{ij} = \log_2 tf_{ij} - \log_2 \frac{df_i}{N} \quad (11.14)$$

where tf_{ij} is the frequency of the word, i , in the text, j , df_i is the number of texts which include the word, i in the corpus, and N is the total number of texts in the corpus. Refer to [2] for studying the process in more detail.

We need to define the Gaussian distribution for each category, in order to use the EM algorithm. Its mean vector and its covariance matrix are computed using the labeled examples. The mean vector and the covariance matrix become the parameter which characterizes the Gaussian distribution. If the only mean vector is considered, the data items are clustered by the k means algorithm as the specific version of the EM algorithm. The theory on the central limit theorem is the reason of defining the Gaussian distribution for each category.

Let us make some remarks on the initialization for implementing the text classification using the EM algorithm and the Naive Bayes, as the semi-supervised learning. The two sets of the training examples are prepared; one is the set of the labeled examples and the other is the set of the unlabeled ones. The texts are encoded into numerical vectors and the process of doing it is described in detail in [3]. The initial distributions over the categories are defined using the labeled examples. The additional categories which are not available or very sparse in the labeled examples should be considered in defining the initial distributions.

11.4.3 Likelihood Estimation

This section is concerned with the likelihood estimation as the role of the Naive Bayes in the text classification. The unlabeled examples are clustered based on the clusters which are clustered by the labeled examples, by the EM algorithm. The role of the Naive Bayes is to compute the likelihoods of individual attribute values to the categories from the item-cluster matrix which is built by the EM algorithm. Another role of the Naive Bayes is to classify novice texts which are given subsequently. This section is intended to describe the process of estimating the likelihoods from the item-cluster matrix.

It is necessary to make some assumptions before estimating the likelihoods from the item-cluster matrix. Each text is encoded into a binary vector whose elements are zeros or ones. The features in encoding a text into a numerical vector are given as words; any other types of features, such as posting features and grammatical ones, are excluded. The zero indicates the absence of the word which corresponds to a feature in the text and the one does its presence. For simplicity, it is assumed that there are only binary values in the numerical vector which represents it.

The cluster-item matrix which consists of the cluster membership values is generated from clustering the labeled examples and the unlabeled examples by the EM algorithm. For each category, the Gaussian distribution is defined, using the labeled examples. The cluster membership values are updated continually for

each item by the E-step and the M-step. The cluster-item matrix is expressed as Eq. (11.15),

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1|C|} \\ c_{21} & c_{22} & \dots & c_{2|C|} \\ \dots & \dots & \dots & \dots \\ c_{N1} & c_{N2} & \dots & c_{N|C|} \end{bmatrix} \quad (11.15)$$

where each row corresponds to a training example and each column does to a cluster. The cluster membership values are used as the weights for computing the likelihoods in using the Naive Bayes.

Let us mention the process of the likelihoods by the Naive Bayes as its learning process. The item-cluster matrix is generated by the EM algorithm, using the labeled examples and the unlabeled examples. The likelihood of individual attribute value, $a_i = v_i$, to the category, c_k , $P(a_i = v_i | c_k)$ is computed by Eq. (11.16),

$$P(a_i = v_i | c_k) = \frac{\sum_{i=1 \wedge a_i=v_i}^N c_{ik}}{\sum_{i=1}^N c_{ik}} \quad (11.16)$$

The likelihood is the rate of the summation of the category membership values of the example with the attribute value, $a_i = v_i$, to the summation of the all category membership values. The training examples are labeled with the membership values of the categories, rather than with an exclusive category.

Let us make some remarks on the likelihood estimations as the role of the Naive Bayes which is mentioned in this section. The approach with the combination of the Naive Bayes with the EM algorithm to the text classification belongs to the semi-supervised learning. The unlabeled examples are added to the training examples, assuming no empty category in the labeled ones. The role of the EM algorithm is to optimize the item-cluster matrix based on the initial version from the labeled examples, using the unlabeled examples. The role of the Naive Bayes is to compute the likelihoods from the optimized item-cluster matrix.

11.4.4 Parameter Estimation

This section is concerned with the process of estimating the parameters of the distribution as the role of the EM algorithm. We studied the process of estimating the likelihoods as the role of the Naive Bayes. In the semi-supervised learning which is implemented by the EM algorithm and the Naive Bayes, each category is assumed as a Gaussian distribution and its involved parameters are estimated for each category. It is possible to assume each category as other distributions such as the triangle distribution and the uniform distribution. This section is intended to describe the process of estimating the parameters for each category.

Let us mention of the process of estimating the mean vector for each category from the item-category matrix. The matrix which consists of the category memberships of each item is expressed by $N \times |c|$ matrix as shown in Eq. (11.15), and the training set is notated by $Tr = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. The mean vector in the category, c_i is computed by Eq. (11.17),

$$\boldsymbol{\mu}_j = \frac{1}{N} \sum_{k=1}^N c_{ki} \mathbf{x}_k \quad (11.17)$$

Equation (11.17) is transformed into Eq. (11.18) by product of the j th column vector in the matrix, $\mathbf{C}$ indicating the memberships of the training example of the category, c_j , $\mathbf{col}_j(\mathbf{C})$ and the matrix which represents the training set,

$$\boldsymbol{\mu}_i = \frac{1}{N} \mathbf{Tr} \cdot \mathbf{col}_j(\mathbf{C}) \quad (11.18)$$

where the matrix, $\mathbf{Tr}$, is shown in Eq. (11.19),

$$\mathbf{Tr} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ x_{N1} & x_{N2} & \dots & x_{Nd} \end{bmatrix} \quad (11.19)$$

The predefined categories are notated as $c_1, c_2, \dots, c_{|C|}$, by applying the Naive Bayes and the EM algorithm to the text classification.

After computing the mean vectors by the above process, the $d \times d$ covariance matrix is computed for each category. The Gaussian distribution which is given to each category is characterized with its mean vector and its covariance matrix. The mean vector for the i th category is given as $\boldsymbol{\mu}_i = [\mu_{i1} \ \mu_{i2} \ \dots \ \mu_{id}]$ and each element of the covariance matrix, $\mathbf{Cov}_i$ is computed by Eq. (11.20),

$$var_{pr}^i = \frac{1}{N} \sum_{k=1}^N N c_{ki} (x_{kp} - \mu_{ip})(x_{kr} - \mu_{ir}) \quad (11.20)$$

In the category, c_i , the mean vector, $\boldsymbol{\mu}_i = [\mu_{i1} \ \mu_{i2} \ \dots \ \mu_{id}]$, and the covariance matrix which is given in Eq. (11.21) are defined,

$$\mathbf{Cov}_i = \begin{bmatrix} var_{11}^i & var_{12}^i & \dots & var_{1d}^i \\ var_{21}^i & var_{22}^i & \dots & var_{2d}^i \\ \dots & \dots & \dots & \dots \\ var_{d1}^i & var_{d2}^i & \dots & var_{dd}^i \end{bmatrix} \quad (11.21)$$

The cluster membership values are updated, based on the mean vector and the covariance matrix.

We may consider other distributions which are alternative to the Gaussian distribution. For each category, the uniform distribution may be considered. In Sect. 11.2.4, we mentioned the triangular distribution which is a fuzzy distribution. The Poisson distribution was mentioned for characterizing each category, in spite of the discrete probability distribution. Depending on the application area, multiple different distributions may be defined for each category.

We mentioned some distributions which are defined for each category, and adopt the Gaussian distribution for characterizing each cluster, in using the EM algorithm. The most simplified Gaussian distribution is one with the uniform covariance matrix and the k means algorithm is the EM algorithm which adopt the most simplified. It is assumed that a Gaussian distribution is defined for each cluster and the EM algorithm may be expanded into the advanced version by considering multiple Gaussian distributions in each cluster. Various versions of EM algorithm are developed, depending on the scheme of characterizing each cluster and the scheme of computing cluster membership values for each data item. As the further study, we consider the deep EM algorithm by attaching the multiple level input encoding and/or the multiple level output encoding.

11.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We mentioned the distributions which are defined to each cluster, including the Gaussian distribution. We studied the EM algorithm with respect to its initialization, its clustering process, and its issues. The EM algorithm may be applied together with the Naive Bayes to the text classification, as the semi-supervised learning. In this chapter, we studied the cluster distributions, the clustering frame by the EM algorithm, and its application to the text classification.

Let us consider the case of allowing multiple distributions for each cluster. By using the EM algorithm, we usually assume a single Gaussian distribution for each cluster. The distribution over each cluster is really an irregular form rather than a hyperplane or a bell shape. It is possible to reflect the irregularity of cluster by defining multiple distribution for each cluster. In this case, we compute cluster membership values for each item by averaging or voting probabilities from distributions.

The cluster-item matrix is defined by using the EM algorithm for clustering data items. It is the matrix where each column corresponds to a cluster, each row does to a data item, and each element indicates a cluster membership value. Cluster membership values are differently assigned to each item, depending on the EM algorithm version. In this case, multiple cluster-item matrices as clustering results are defined. In addition, the final cluster-item matrix should be decided by using the multiple matrices.

The initialization of the EM algorithm is to decide the number of clusters and to define probabilistic distribution for each cluster. The results from clustering

data items by the EM algorithm depend on how to initialize clusters. We may accommodate multiple different versions of clustering results; this is called multiple viewed clustering. We consider selecting best one by evaluating the versions or integrating the multiple versions into a single version, in this case. We may present the multiple versions and the integrated one at same time for users.

Let us consider integrating multiple versions of clustering results into a single version. Depending on how to define cluster distributions, we expect multiple versions by using the EM algorithm. Even if some probability distributions are defined over the clusters, different clustering results are caused by defining different initial parameters. We may consider multiple EM algorithms with different initial parameters and different probability distributions to the data clustering. We mention the combination of multiple clustering algorithms with each other in Chap. 14.

References

1. M. Beaver, *Introduction to Probability and Statistics* (PWS-Kent, Aurora, 1991)
2. T. Jo, *The Implementation of Dynamic Document Organization using Text Categorization and Text Clustering*, PhD Dissertation (University of Ottawa, Ottawa, 2006)
3. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, New York, 2018)

Chapter 12

Advanced Clustering



12.1 Introduction

In this chapter, we consider the clustering with the parameter tuning as the advanced clustering methods. The clustering index is defined as the metric for evaluating clustering results. The data items are clustered with the interactive evaluation of clustering results for tuning the parameters. The clustering governance as the additional task to the data clustering is to maintain clusters for keeping the clustering quality. This section is intended to describe the clustering index and the parameter tuning, briefly, as the additional tasks which are derived from the data clustering.

Let us mention the clustering index as the clustering evaluation metric. It was initially proposed for evaluating the clustering results by Jo in 2006 [3]. The clustering index was used for evaluating clustering results, subsequently and continually [8]. The two metrics, the intra-cluster similarity and the inter-cluster similarity, are computed to the clustering results. Both are integrated in the type of combining the precision and the recall into the F1 measure.

Using the clustering index, the parameters of clustering algorithms may be tuned. The clustering index is initially intended to evaluate the clustering results by Jo in 2006 [3]. When using the clustering index for tuning the parameters, the clustering results are found with the optimal parameters. For example, the number of clusters in the k means algorithm may be optimized automatically. In Sect. 12.3, we will mention some clustering algorithms with the parameter tuning.

Let us consider the additional tasks for maintaining the constructed clusters, called cluster governance. A cluster may be divided into more than one cluster as the reaction to the continual addition of items. More than one cluster may be merged into one cluster to the continual deletion of items. A symbolic name may be assigned to each cluster, especially in the text clustering, and it is initially mentioned by Jo in 2006 [3]. The clustering shifting may be considered as adjusting the cluster boundary, depending on the distribution over examples.

The goal of this chapter is to understand the advanced clustering algorithms with their parameter tuning. We will understand the clustering index as an evaluation metric for clustering results. We need to understand existing clustering algorithms where the clustering index is installed for tuning their parameters. We will also understand the additional tasks to the data clustering for maintaining clustering results to added data items and deleted ones. We will make the further discussions on what is studied in this chapter for installing the parameter tuning into other clustering algorithms and developing advanced algorithms for doing the cluster governance.

12.2 Cluster Index

This section is concerned with the clustering index that was initially proposed by Jo in 2006 as the evaluation metric of clustering results [3]. In Sect. 12.2.1, we explain how to compute the clustering index, assuming that labeled examples are used for evaluating the clustering results. In Sects. 12.2.2 and 12.2.3, we cover the process of evaluating results of both kinds of clustering, the fuzzy clustering and the crisp clustering. In Sect. 12.2.4, we consider the evaluation of results from the hierarchical clustering using the clustering index. This section is intended to define the clustering index as the clustering evaluation metric and explain the process of evaluating the clustering results using it.

12.2.1 Computation Process

The clustering index is defined for evaluating the clustering results by integrating the intra-cluster similarity and the inter-cluster similarity. The intra-cluster similarity is the average over similarities among items within a cluster and should be maximized for making the desirable clustering. The inter-cluster similarity is the average over similarities among clusters and should be minimized for doing it. The clustering index is constructed by integrating the intra-cluster similarity and the inverse inter-cluster similarity. This section is intended to describe the two metrics, the intra-cluster similarity and the inter-cluster similarity, and integrate them into the clustering index.

The intra-cluster similarity, which is the cohesion of each cluster, is defined as the similarities among items within a cluster. The similarity between two items, $\mathbf{x}_i$ and $\mathbf{x}_j$, is notated by $\text{sim}(\mathbf{x}_i, \mathbf{x}_j)$, and a cluster is notated as a set of items, $C_r = \{\mathbf{x}_{r1}, \mathbf{x}_{r2}, \dots, \mathbf{x}_{rk}\}$. The intra-cluster similarity of cluster, C_r , by averaging over similarities of all possible pairs of items, is shown in Eq. (12.1),

$$intra - sim(C_r) = \frac{2}{k(k-1)} \sum_{i < j} sim(\mathbf{x}_i, \mathbf{x}_j) \quad (12.1)$$

When the clustering results are noted by a set of clusters, $C = \{C_1, C_2, \dots, C_{|C|}\}$, the intra-cluster similarity over the clustering results is expressed as Eq. (12.2),

$$intra - sim(C) = \frac{1}{|C|} \sum_{i=1}^{|C|} intra - sim(C_i) \quad (12.2)$$

The direction of clustering data items is to maximize the intra-cluster similarity.

The inter-cluster similarity, which is the average discrimination among clusters, is defined as the similarities among clusters. The two clusters are notated by C_i and C_j and expressed into the two sets of input vectors, $C_i = \{\mathbf{x}_{i1}, \mathbf{x}_{i2}, \dots, \mathbf{x}_{i|C_i|}\}$ and $C_j = \{\mathbf{x}_{j1}, \mathbf{x}_{j2}, \dots, \mathbf{x}_{j|C_j|}\}$. The inter-cluster similarity between C_i and C_j is computed by Eq. (12.3),

$$inter - sim(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{k,m}^{|C_i||C_j|} sim(\mathbf{x}_{ik}, \mathbf{x}_{jm}) \quad (12.3)$$

When the clustering results are noted by a set of clusters, $C = \{C_1, C_2, \dots, C_{|C|}\}$, the inter-cluster similarity over the clustering results is computed by Eq. (12.4),

$$inter - sim(C) = \frac{2}{|C|(|C| - 1)} \sum_{i < j} inter - sim(C_i, C_j) \quad (12.4)$$

Note that the inter-cluster similarity is not applicable to only single cluster.

We computed the two metrics by the above process: the inter-cluster similarity and the intra-cluster similarity. The cluster results are given as a cluster set, $C = \{C_1, C_2, \dots, C_{|C|}\}$, and the intra-cluster similarity and the inter-cluster similarity are notated by $intra - sim(C)$ and $inter - sim(C)$. Because $inter - sim(C)$ is always given as a normalized value between zero and one, the discrimination of the clustering results, C , is expressed into Eq. (12.5),

$$discriminate(C) = 1.0 - inter - sim(C) \quad (12.5)$$

The clustering index is computed by Eq. (12.6),

$$CI(C) = \frac{2 \cdot intra - sim(C) \cdot discriminate(C)}{intra - sim(C) + discriminate(C)} \quad (12.6)$$

using $intra - sim(C)$ and $discriminate(C)$. The clustering index of the clustering results, C , $CI(C)$, integrates the two metrics, $intra - sim(C)$ and $discriminate(C)$, like the F1 measure.

Let us make some remarks on the clustering index that is the metric for evaluating the clustering results. Various metrics were previously proposed for evaluating them. The clustering index was initially proposed by Jo in 2006 [3]. It has been used until recent works [4, 6]. The clustering index is used for tuning the parameters of clustering algorithms as well as evaluating the clustering results [5].

12.2.2 Hard Clustering Evaluation

This section is concerned with the process of evaluating the results from the hard clustering, using the clustering index. The hard clustering is the clustering type where no overlapping between clusters is allowed. There is no item that belongs to more than one cluster in this clustering type. The exclusive clusters are results from the hard clustering; the intersection between two clusters is always an empty set. This section is intended to mention the process of evaluating the hard clustering results using the clustering index, which is mentioned in Sect. 12.2.1.

The labeled examples are used for evaluating clustering results. The labels are blinded in executing clustering algorithms. The similarity between two examples is computed by their labels after clustering data, using Eq. (12.7),

$$sim(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 & \text{if } y_i = y_j \\ 0 & \text{otherwise} \end{cases} \quad (12.7)$$

If the target label of two examples is identical to each other, the similarity between them becomes one, and otherwise, it becomes zero, as shown in Eq. (12.7). The similarity is allowed as a continuous value, in the case of the fuzzy clustering or the hierarchical clustering.

Let us mention the process of computing the clustering index for evaluating the clustering results. The results from clustering items are $C = \{C_1, C_2, \dots, C_{|C|}\}$, and the intra-cluster similarity for each cluster is computed by Eq. (12.1). The intra-cluster similarity over the clustering results is computed by Eq. (12.2). For each pair of clusters, the inter-cluster similarity is computed by Eq. (12.3), and the inter-cluster similarity to the clustering results is computed by Eq. (12.4). It is required to use labeled examples for evaluating the clustering results, using Eq. (12.7).

Let us demonstrate the process of computing the clustering index from simple clustering results. They are given as follows:

$$\begin{aligned} C_1 &= \{+, +, +, 1\} \\ C_2 &= \{-, -, +, -\} \\ C_3 &= \{+, +, -, -\} \end{aligned}$$

The intra-cluster similarities of the three clusters are computed as follows:

$$intra - sim(C_1) = 0.5$$

$$intra - sim(C_2) = 0.5$$

$$intra - sim(C_3) = 0.33$$

The intra-cluster similarity on the clustering results is computed by averaging over the similarities as follows:

$$intra - sim(C) = 0.4444$$

The inter-cluster similarity is computed for each cluster pair as follows:

$$inter - sim(C_1, C_2) = 0.4375$$

$$inter - sim(C_1, C_3) = 0.5$$

$$inter - sim(C_2, C_3) = 0.5$$

The inter-cluster similarity on the results is computed by averaging them as follows:

$$inter - sim(C) = 0.4791$$

The discrimination among clusters is computed by subtracting the inter-cluster similarity from 1.0, as follows:

$$discriminate(C) = 0.5209$$

The clustering index is computed by Eq. (12.6) and becomes 0.4793.

Let us make some remarks on the evaluation of the hard clustering results using the clustering index. It is assumed that the labeled examples are used for evaluating the results and their target labels are hidden during clustering data items. In evaluating the results, they are opened, and the similarity between examples is defined in Eq. (12.7). The clustering algorithms are usually evaluated in the hard clustering which is given as a toy problem. The excellent performance of a clustering algorithm in the hard clustering is never guaranteed in the soft clustering or the hierarchical clustering.

12.2.3 Fuzzy Clustering Evaluation

This section is concerned with the process of evaluating the fuzzy clustering results using the clustering index. The fuzzy clustering is the clustering type where each item is allowed to belong to more than one cluster. There are the two views of the fuzzy clustering; it is viewed into clusters that are overlapped with other clusters, graphically, and it is done into an item–cluster matrix where each element is given

as a membership value of each item in each cluster. The former view is adopted for evaluating the fuzzy clustering results in this section. This section is intended to describe the process of evaluating the fuzzy clustering with the view of the overlapped clustering, using the clustering index.

The decomposition of the fuzzy clustering into binary clustering results is described in [5]. It is assumed that more than two clusters with their overlaps are given as the results. For each cluster, the items that belong to it are set to one group, and the others are set to the other. The clustering index is computed for the two exclusive groups for each cluster. The average over the clustering indices, as many as clusters, is the final clustering index.

Let us compute the clustering index for each cluster pair from decomposing the fuzzy clustering results. The two clusters in the pair is notated as C_i and the complement, $R_i = (C_i)^R$. The intra-cluster similarity, $intra-sim(C_i)$, is computed to the given cluster, C_i , and the inter-cluster similarity of the nominated cluster and its complement, $inter-sim(C_i, R_i)$. The clustering index of the cluster pair is computed by Eq. (12.8),

$$CI_i = \frac{2 \cdot intra-sim(C) \cdot (1 - inter-sim(C, R_i))}{intra-sim(C) + (1 - inter-sim(C, R_i))} \quad (12.8)$$

The m clustering indices, $CI_1, CI_2, \dots, CI_m$, are computed, in the m overlapping clusters, $C = \{CI_1, CI_2, \dots, CI_m\}$ as the fuzzy clustering results.

Let us mention the scheme of integrating the clustering indices of the cluster pairs which are computed above. It is assumed that the clustering indices are given as $CI_1, CI_2, \dots, CI_m$. The clustering index is computed to the fuzzy clustering results, which consists of m clusters, by averaging the above clustering indices, as shown in Eq. (12.9),

$$CI(C) = \frac{1}{m} \sum_{i=1}^m CI_i \quad (12.9)$$

The above equation is modified into Eq. (12.10), by adapting the cluster sizes, $|C_1|, |C_2|, \dots, |C_m|$,

$$CI(C) = \frac{\sum_{i=1}^m |CI_i| CI_i}{\sum_{i=1}^m |CI_i|} \quad (12.10)$$

The small clusters may be removed as outliers in evaluating the fuzzy clustering results.

Let us make some remarks on the process of evaluating the fuzzy clustering results using the clustering index. The higher inter-cluster similarities in evaluating the fuzzy clustering results than in doing the hard clustering results by same items between two clusters are the cause of underestimating the clustering index. It was proposed that the fuzzy clustering results should be divided into binary clusters as

many as clusters as the solution to the problem. Each binary cluster consists of the group of items in the corresponding cluster and the group of the others. Averaging over the clustering indices of the binary clustering results is the clustering index over the entire fuzzy clustering results.

12.2.4 Hierarchical Clustering Evaluation

This section is concerned with the process of evaluating the hierarchical clustering results, using the clustering index. In the previous sections, we studied the process of computing the clustering index from the hard clustering results and the fuzzy clustering results. A hierarchical structure of item clusters is the results from clustering items hierarchically, and we study how to evaluate the clustering performance in this section. Evaluation of the hierarchical clustering results proceeds level by level, and the clustering indices of the levels are averaged. This section is intended to describe the scheme of evaluating the hierarchical clustering results, using the clustering index.

The hierarchical clustering results are given as a true structure of unnamed clusters. More than one nested clusters exists in a particular cluster in this clustering type. Examples that are labeled with categories in the lowest level are used for evaluating hierarchical clustering results. We consider the case of the consistence of categories in the higher level, but the inconsistency of those in the lower level. Actually, clustering algorithms are evaluated in the flat clustering, and the best one is applied in the hierarchical clustering.

Let us mention the process of computing the clustering index to the hierarchical clustering results. The clusters in a particular level, l , are notated by $C_l = \{C_{l1}, C_{l2}, \dots, C_{l|l|}\}$, where $|l|$ is the total number of clusters in level, l . The hierarchical clustering is assumed to have exclusive clusters, and the clustering index in the clusters in level l is computed by the process that is mentioned in Sect. 12.2.2. The clustering index to the entire clustering results is computed by the weighted average over the clustering indices of the levels, as shown in Eq. (12.11),

$$CI(C) = \sum_{i=1}^L w_i CI_i \quad (12.11)$$

where L is the number of levels in the hierarchical clustering results from the root level to the terminal level. Zero is set to the root level, and the weights are set as less in the higher level and more in the lower level, $w_1 \leq w_2 \leq \dots \leq w_L$.

Let us mention the process of evaluating the hierarchical clustering results, using the clustering index. The hierarchical clustering results may be decomposed into several flat clustering results, level by level. For each of flat clustering results in each level, the clustering index is computed. The clustering index to the entire results is computed by the weighted average over the clustering indices of levels.

The clustering results in higher levels may be overestimated, and the ones in lower levels are underestimated.

We need to decompose the hierarchical clustering results into flat clustering ones, for evaluating results, easily. The hierarchical clustering results are expressed as a tree whose root node is in level 0. The clusters in the level 1 are extracted as the flat list, and the nested clusters from each cluster are extracted, continually. The nested clusters within a particular cluster are its child nodes in expressing the hierarchical cluster results as a tree. The issue for evaluating the hierarchical clustering results is how to assign weights to each flat cluster list.

12.3 Parameter Tuning

This section is concerned with the parameter tuning that is installed in the clustering algorithms. In Sect. 12.3.1, we describe the process of computing the clustering index using the unlabeled examples. In Sects. 12.3.2 and 12.3.3, we mention some simple clustering algorithms and the k means algorithm where the parameter tuning is installed. In Sect. 12.3.4, we explain the application of the evolutionary computation to the clustering task. This section is intended to introduce the computation of the clustering index from unlabeled examples and explain the clustering algorithms with the parameter tuning.

12.3.1 Clustering Index to Unlabeled Items

This section is concerned with the process of computing the clustering index from unlabeled data items. In Sect. 12.2.1, we described the process of computing it from labeled examples for evaluating clustering results. The similarity metrics among data items, which are given as numerical vectors, such as the cosine similarity and the inverse Euclidean distance, become the basis for computing the clustering index. It is used for optimizing the parameters of clustering algorithms and deciding the termination of iterating cluster updates. This section is intended to describe the computation of the clustering index from unlabeled items.

We need to define the similarity metric among unlabeled examples for computing the clustering index. It is assumed that data items are given as d dimensional numerical vectors. The cosine similarity, which was mentioned in Chap. 4, is used as the similarity metric. The cosine similarity variants or the inverse Euclidean distance may be set as a similarity metric. Recently, the cosine similarity variant that considers the feature similarity is proposed for computing the similarity between words or texts [7].

Once the similarity metric is decided, the clustering index may be computed from the unlabeled examples. The current clustering results are given as a set of clusters, $C = \{C_1, C_2, \dots, C_{|C|}\}$, the target labels are not available, and the

cosine similarity or the inverse Euclidean distance is defined as a similarity metric. The intra-cluster similarity is computed by Eq. (12.1) and (12.2). The inter-cluster similarity is computed by (12.3) and (12.4). The process of computing the clustering index is same to the process that was described in Sect. 12.2.2, except the fact that the target labels are not available.

The two metrics are computed from the unlabeled examples, above, and they need to be integrated in a single metric, which is called clustering index. Assuming that the inter-cluster similarity is given as a normalized value, we define the discrimination among clusters by Eq. (12.5). The two measures are integrated by Eq. (12.6). The integration of the intra-cluster similarity and the discrimination into the clustering index is like the case of integrating the precision and the recall into the F1 measure in evaluating the information retrieval systems. The clustering index that is the quantitative degree of the current clustering results is used for tuning the parameters in the progress of clustering data items.

Let us compare the computation of the clustering index from the unlabeled examples with that from the labeled ones. The clustering index that is computed from the unlabeled examples is used for tuning parameters, whereas one that is computed from the labeled examples is used for evaluating the clustering results. In computing the clustering index from the unlabeled examples, the categories are not defined, whereas in computing it from the labeled examples, the categories are defined. The clustering index is based on the consistence with the target labels in computing it from the labeled examples, whereas it is based on the cosine similarity or the Euclidean distance, in computing it from the unlabeled examples. In the both cases, the intra-cluster similarity and the discrimination over the entire results are integrated with each other in the style of the F1 measure.

12.3.2 Simple Clustering Algorithms

This section is concerned with installing the parameter tuning into simple clustering algorithms. We will review the evolutionary clustering algorithm that is expanded from the evolutionary binary clustering that is mentioned in Sect. 9.3.2. The parameter tuning that is based on the clustering index, which is described in Sect. 12.3.1, is installed into the AHC algorithm and the divisive algorithm. In installing the parameter tuning, its advantage is the automatic optimization of the parameter for better clustering quality, but its disadvantage is the higher complexity of clustering data items. This section is intended to explain the evolutionary clustering algorithm, the AHC algorithm, and the divisive algorithm with the parameter tuning.

Let us mention the evolutionary clustering algorithm as a typical parameter tuning-based clustering algorithm. We mentioned the evolutionary binary clustering algorithm that segments a group into two subgroups using the simple genetic algorithm in Sect. 9.3.2. It is expanded into the evolutionary clustering algorithm that segments a group into more than two subgroups. Each solution represents into a bit string that consists of items, each of which indicates a clustering index from

zero to the number of clusters minus 1. The clustering index, which is described in Sect. 12.3.1, is used as the fitness value for evaluating an individual solution.

The parameter tuning is installed into the AHC algorithm. It is described in detail in Sect. 9.2. The process of computing the clustering index from the current clustering results is added after computing the cluster similarities and merging the clusters with their maximum similarity, in each iteration. The previous clustering index is memorized, and when the previous one is higher than the current one, the iteration is terminated. The AHC algorithm installed with the parameter tuning belongs to the hill climbing; there is possibility of falling into a local maxima of the clustering index, as the pay-off.

Let us mention the divisive clustering algorithm where the parameter turning is installed. It is mentioned entirely in Sect. 9.3. The clustering index is computed after dividing a group into two clusters. The previous clustering index is compared with the current one, like the AHC algorithm with the parameter tuning. It is applied to the process of arranging items into one of the two groups as well as the termination condition.

The parameter tuning is installed into the linear online clustering algorithm, which is covered in Sect. 9.4. It was characterized by Jo through empirical validations on clustering algorithms as the fast but less reliable one [3]. It is expected to improve the clustering performance by installing the parameter tuning. Whether each item is arranged into one of the existing clusters or create, a new cluster is decided by computing the clustering index of the two cases. In installing the parameter tuning, the clustering speed is sacrificed as the payment for improving the clustering performance.

12.3.3 *K Means Algorithm*

This section is concerned with the k means algorithm where the parameter tuning is installed. In Chap. 10, we already studied the k means algorithm that is used most popularly for clustering data items. The two parameters, the number of clusters and the initial mean vectors, are optimized by tuning them. The clustering indices of multiple k means algorithms with their different parameters are observed in parallel for selecting one among them. This section is intended to explain the process of clustering data items using the modified version of the k means algorithm.

It is required to decide initially the number of clusters for using the k means algorithm for clustering data items. The fact that the number of clusters is decided in advance arbitrary is the reality in using the k means algorithm. If the raw data is encoded into two or three dimensional vectors, it is possible to decide the desirable number of clusters by plotting them in the two or three dimensional space. When each item is given as more than three dimensional vector, it is impossible to decide the number of clusters. Here, we propose the clustering index that is computed from the current clustering results should be used for finding the desirable number of clusters.

Let us mention the process of clustering data items using the variant of the k means algorithm. The mean vectors are initialized by selecting k data items at random. The data items are clustered into the k subgroups by the k means algorithm. The clustering index is computed from the current results. It is used for updating the number of clusters, dynamically.

Let us consider the two cases, the $k - 1$ means algorithm and the $k + 1$ algorithm, after executing the k means algorithm. The data items are clustered into the $k - 1$ clusters and the $k + 1$ clusters by the two algorithms. The clustering indices of the two results are computed by the process that is described in Sect. 12.3.1. The three cases, the $k - 1$ clusters, the k clusters, and the $k + 1$ clusters, are compared with each other, and the case with the maximal clustering index is selected. In the case of maximal clustering index in clustering data items into k subgroups, the iteration is terminated.

Let us consider the combination of the k means algorithm with the evolutionary algorithm. The clustering index of the current clustering results indicates the fitness value in using the evolutionary algorithm. The number of clusters and the initial cluster prototypes become the external parameters of the k means algorithm. The external parameters are optimized using the evolutionary algorithm. The external parameters are represented into bit strings in using the genetic algorithm or numerical vectors in using the evolutionary strategy or the evolutionary programming.

12.3.4 Evolutionary Clustering

This section is concerned with the process of clustering the data items using the evolutionary computation. The clustering index, which is described in Sects. 12.1 and 12.2, is used as the fitness value of the clustering results. They are encoded into the genotypes that are given as a bit string or a numerical vector, and the recombination operations are applied to them. The clustering results are generated at random or by recombination of existing ones, and the fitness of each one is evaluated, as the process of the evolutionary clustering. This section is intended to describe the evolutionary clustering of data items, in detail.

Let us mention the process of encoding the clustering results into a genotype. They are encoded into a string that consists of integers, assuming data items into M clusters. An individual number of the strings indicate a cluster identifier that is given as a number, and the string length corresponds to the total number of data items. The recombinant operations, such as the cross over and the mutation, are applied to an integer string, as well as a bit string. The case of using the evolutionary computation to the binary clustering is expanded into that of doing it to the general clustering.

The scheme of evaluating the fitness for each iteration is needed in applying the evolutionary computation to any problem. The given problem is assumed to be a flat clustering, and the item-cluster list is given as the phenotype of the solution. We studied the process of computing the clustering index from the clustering results, in Sect. 12.3.1, so the fitness values are evaluated by doing so. The clustering index is

used as the fitness value of each solution, in applying the evolutionary computation to the data clustering. Many evaluation metrics of the data clustering have been defined until now, and it is possible to use other evaluation metrics for this type of the clustering algorithm [1].

Encoding the clustering results into their genotypes as integer strings and computing the clustering index for each solution as its fitness value is the preparation of evolving the current solution population into one in the next generation. The initial solution population is constructed at random, and off-springs are generated by the recombination of the existing ones in the current population. Each solution, which is an existing one or an off-spring, is evaluated with its fitness value, and the ones with its lower fitness values are removed. After the removal, the remaining solutions become the members of the population in the next generation. As the generation goes, the members in the population have better fitness values.

Let us mention some specific evolutionary algorithms, depending on the encoding scheme and the recombinant operations. The genetic algorithm is an evolutionary computation instance where each solution is encoded into a bit string and the cross over is used as a main recombinant operation. The genetic programming is one where each solution is encoded into a tree, and the only cross over is used. The evolutionary strategy is one where each solution is encoded into a numerical vector, and the mutation is used. The evolutionary programming is mentioned as a variant of the evolutionary strategy with the different mutation schemes.

12.4 Clustering Governance

This section is concerned with the tasks that are involved in governing the clusters of data items, separated from clustering them. In Sect. 12.4.1, we mention the assignment of symbolic names to clusters after finishing the data clustering. In Sect. 12.4.2, we describe the process of maintaining clusters to the continual addition and deletion of new items and existing ones. In Sect. 12.4.3, we present another type of data clustering, called multiple viewed data clustering. In Sect. 12.4.4, we consider the integration of clustering results by several different approaches.

12.4.1 Cluster Naming

The cluster naming is defined as the process of assigning a symbolic name that is relevant to the cluster content to each cluster. Identifying each cluster with a number does not belong to the cluster naming. The cluster naming was initially mentioned by Jo in 2006, and its principles are presented for clustering texts [3]. Less than or equal to three words is assigned to each cluster, and the label should be relevant to the cluster contents. This section is intended to describe the cluster naming principles, the cluster naming process, and the cluster naming results.

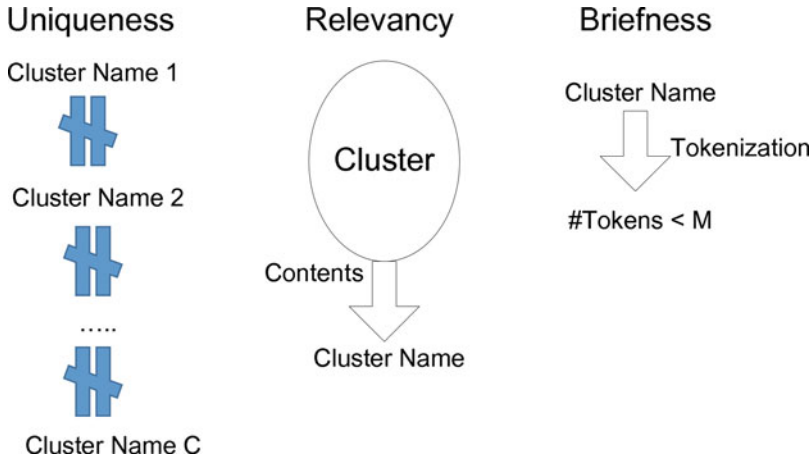


Fig. 12.1 Principles of cluster naming

```

nameClusterList(List clusterList){
  for each cluster in clusterList
    clusterName = cluster.getMaxWeightWord();
    isRedundant = clusterList.isRedundant(clusterName);
    while(isRedundant){
      clusterName = cluster.getNextWeightWord();
      isRedundant = clusterList.isRedundant(clusterName);
    }
    cluster.putClusterName(clusterName);
  }
}

```

Fig. 12.2 Cluster naming process

Some principles of naming the clusters for enabling the browsing are illustrated in Fig. 12.1. All of the cluster names should be unique; no redundant names of clusters exist. It should be possible to guess contents by the cluster names; the cluster names should be relevant semantically to the cluster contents. The cluster names should be brief; the number of words does not exceed three in naming the clusters. The definition of the principles of naming the clusters is intended to enable the clusters to be browsed.

The process of naming text clusters symbolically is illustrated in Fig. 12.2. The texts in the cluster are indexed into a list of words, and the ones with maximal weights are selected among them. It checks whether each name candidate is redundant to any one among other cluster names, and if so the word with next maximal weight is selected. Checking and selecting are iterated until the redundancy with any name of other clusters is avoided. This cluster naming process is applicable only to the text clustering.

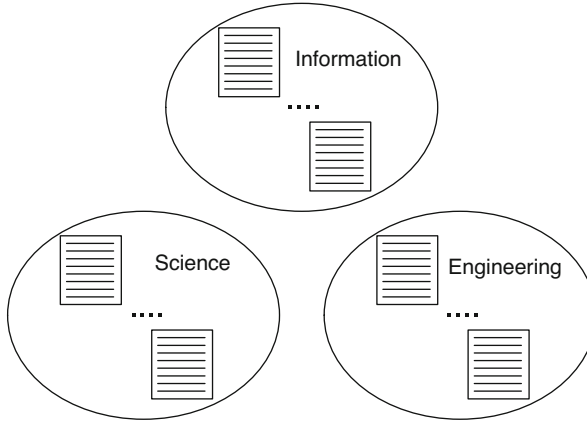


Fig. 12.3 Results from clustering + cluster naming

The results from clustering texts and naming the clusters are illustrated in Fig. 12.3. Texts are encoded into numerical vectors and clustered by a clustering algorithm. By the algorithm that is illustrated in Fig. 12.2, a symbolic name is assigned to each cluster. Both the clustering and the cluster naming are intended to enable the browsing of items. We need to govern the results that are presented in Fig. 12.3, to the addition of more texts and the deletion of some.

Let us make some remarks on the cluster naming as a task which is separated from the data clustering. A flat list or a hierarchical tree of unnamed clusters is the results from clustering data items; it is insufficient for browsing the data items. The cluster naming is needed as an additional task for improving the results enough to do the browsing. The scheme of naming the clusters, which is mentioned in this section, is applicable only to the text clustering. The captions or the scripts, which are given as texts, are required for applying this scheme in other cluster areas.

12.4.2 *Cluster Maintenance*

The cluster maintenance, which is covered in this section, is defined as the process of maintaining the clusters to the addition of more data items and the deletion of existing ones. The cluster maintenance was initially mentioned as the task, which is separated from the data clustering by Jo in his PhD dissertation in 2006 [3]. The cluster partition, the cluster merge, and the cluster prototype update are necessary for maintaining the clusters. We need the cluster naming that was mentioned in Sect. 12.4.1 for maintaining the clusters. This section is intended to describe the necessities and the tasks that are involved in maintaining the clusters.

Let us consider the necessary of maintaining the existing clusters as the additional task to the data clustering. There are always the addition of new data

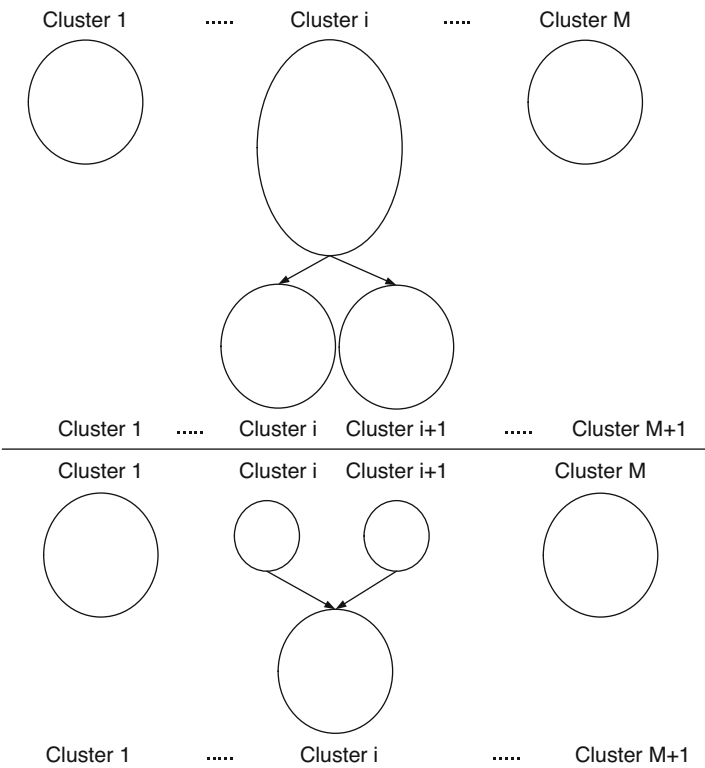


Fig. 12.4 Actions of cluster maintenance

items and the deletion of existing ones in the information systems. As time goes, we need to create new topics or delete existing ones. The desired organization of data items is defined subjectively, depending on users or administrators. Because the big data is characterized the high 4Vs: high volume, high velocity, high variety, and high veracity, we need to maintain the existing clusters much more in the big data age.

The actions for maintaining the clusters are illustrated in Fig. 12.4. The top in Fig. 12.4 is the cluster partition, which partitions a cluster into two clusters as the reaction to the continual addition of more items. The bottom in Fig. 12.4 is the cluster merge, which merges two clusters into a cluster as the reaction to the continual deletion of existing ones. When the added items are distant from all clusters, more cluster may be created. We may consider reorganizing the items entirely by clustering all again.

The hard organization through the entire clustering and the soft organization through the cluster governance are illustrated in Figs. 12.5 and 12.6. The hard organization is the process of organizing the data items, entirely with regardless of the previous organization, and it is implemented by clustering items, entirely. The

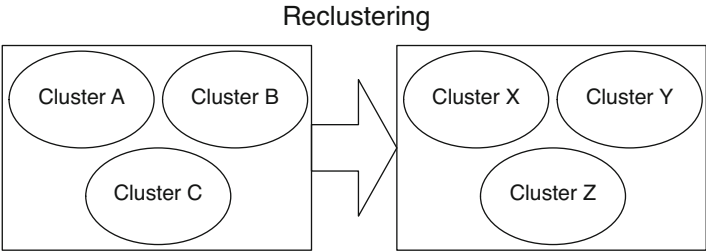


Fig. 12.5 Hard reorganization

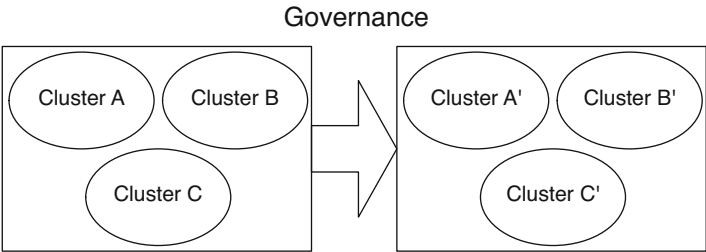


Fig. 12.6 Soft reorganization

soft organization is the process of updating the existing organization of data items, and it is implemented by the actions that are involved in the cluster governance. In the hard organization, the organization with its better quality may be expected, but too much time is taken for clustering data items. The periodical evaluation of organizing the data items is needed for judging whether the hard organization or the soft organization is used.

Let us make some remarks on the maintenance of the clusters that are results from clustering the data items. In the traditional machine learning, we focus on how to cluster data items, but the situation after clustering the data items is not considered. If the clusters of the data items are fixed without the addition and the deletion of data items, the cluster governance is not needed. The contents of the clusters may change by the addition, the deletion, and the update, in the reality; the transactions happen much more frequently in the case of processing the big data. We need to regard, at least, the cluster governance as the important task as the data clustering.

12.4.3 Multiple Viewed Clustering

This section is concerned with the special data clustering type and is called multiple viewed clustering. The multiple viewed clustering and the multiple viewed classification were initially mentioned by Jo, in 2018 [5]. The idea of this clustering type is to accommodate multiple versions of clustering results. Under this idea, the

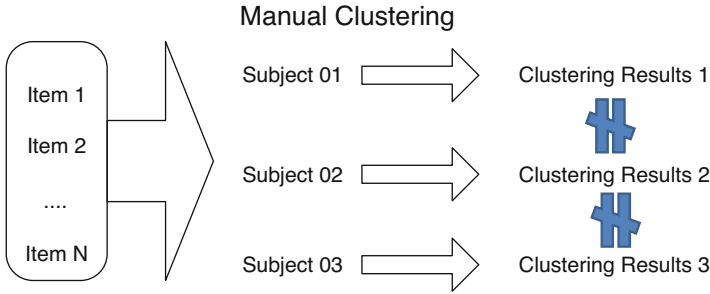


Fig. 12.7 Subjectivity of manual clustering

multiple versions of data organization are maintained at same time. This section is intended to describe the multiple viewed clustering in the conceptual and functional view.

The subjectivity in clustering data items manually is illustrated in Fig. 12.7. A group of various items is initially Given, and it is assumed that they are clustered manually by several subjectives. The results from clustering items manually are different depending on a subjective. The different number of clusters and the different members in each cluster become the output from the different subjective views. The consideration of accommodating the different results by the subjective views becomes the motivation for the multiple viewed clustering.

The dependencies of the clustering results on the approach, the parameters, and the similarity metric are illustrated in Fig. 12.8. The results in using the cosine similarity or the Euclidean distance are different from each other, even in using a same approach. Different results from clustering data items are expected from using different approaches. Changing the external parameters in using the clustering algorithm is the cause of different results. The different clustering results that are caused by the manual clustering by different subjects and the automatic clustering, which is shown in Fig. 12.8, are motivations of idea of the multiple viewed clustering.

The multiple viewed classification is derived by accepting the multiple versions of clustering results as shown in Fig. 12.9. The different subjects, the different clustering algorithms, and the different configurations within a same clustering algorithm become the causes of different clustering results. The data clustering is viewed as the automation of the preliminary tasks for the data classification: the category predefinition and the sample allocation. The multiple versions of the classification frame are obtained by accommodating the multiple clustering results. The multiple viewed classification is mentioned in detail in [5].

Let us make some remarks on one more clustering type, which is called multiple viewed clustering. It is intended to accommodate various versions of clustering results by the different subjects or the different clustering algorithms. In the data classification, we need to accommodate the multiple versions of predefined categories, called multiple viewed classification. The choice of the multiple viewed

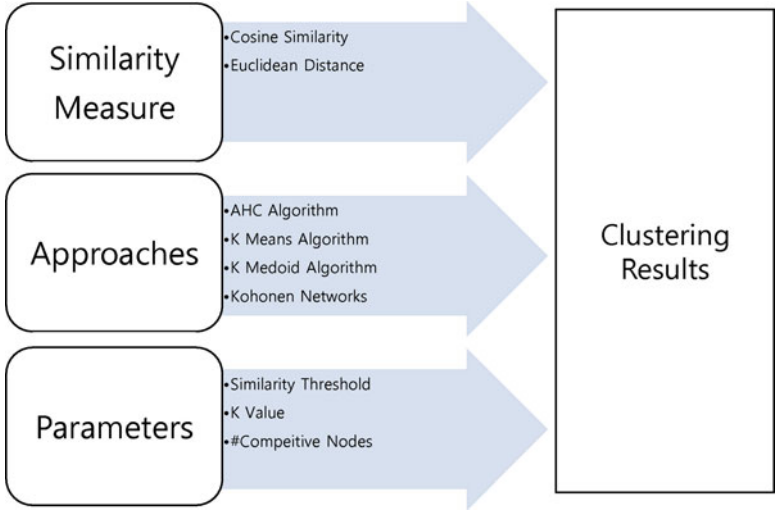


Fig. 12.8 Different clustering results on different approaches

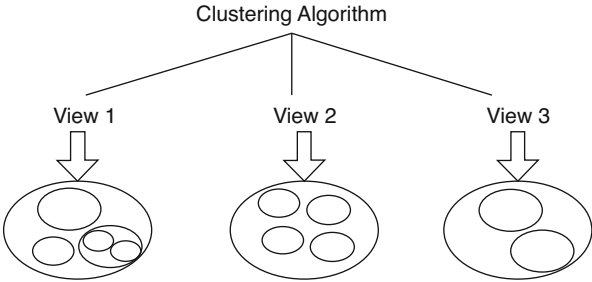


Fig. 12.9 Derivation of multiple viewed clustering

classification or clustering and a single viewed one is the dichotomy whether we accommodate the multiple versions, or not. In the future study, we need to integrate the multiple versions of the clustering results into a single version.

12.4.4 *Clustering Results Integration*

This section is concerned with the integration of multiple clustering results into a single version. The multiple viewed clustering was mentioned as a clustering type with the intention of accepting the multiple versions of the clustering results. We need to integration the multiple versions of clustering results into a single version for users. For simplicity, the integration of multiple clustering results is restricted to

```

integrateTwoClusterResults(List clusterList1, List clusterList2){
    do until isEmpty(clusterList1) or isEmpty(clusterList2){
        clusterPair = findClusterPairWithMaxIntersection(clusterList1,clusterList2)
        //clusterPair ← cluster1, cluster2
        cluster.add(clusterPair.getIntersection());
        restItemList = clusterPair.getIntersectionComplement();
        clusterList1.remove(clusterPair); //remove cluster 1 from clusterList1
        clusterList2.remove(clusterPair); //remove cluster 2 from clusterList2
        integratedClusterList.add(cluster);
    }
    if(isNotEmpty(clusterList1)) restItemList.add(clusterList1.getAllItems());
    if(isNotEmpty(clusterList2)) restItemList.add(clusterList2.getAllItems());
    for each item in restItemList{
        integratedClusterList.arrangeItem(item);
    }
    return integratedClusterList;
}

```

Fig. 12.10 Hard clustering integration

the case where they belong to the identical clustering type. This section is intended to mention the three types of clustering as the integration targets.

The process of integrating the two hard clustering results into one is illustrated as a pseudo code in Fig. 12.10. The two clusters from the two results with the maximal intersection cardinality are selected as a pair. The two clusters are merged into one cluster and deleted in the two clustering results. The process is iterated until either of two clusters becomes empty, and if so, the remaining clusters in either of them are added to the integrated results. The two clustering results are integrated by gathering the similar clusters from the two results.

The process of integrating the two fuzzy clustering results into one is illustrated in Fig. 12.11. The process of integrating them is same to that of doing the hard clustering results except the union or the intersection to the integrated clusters. In the hard clustering, the intersection of two clusters with the maximal intersection is taken into the integrated cluster in merging them, whereas in the fuzzy clustering, the union of the two clusters, instead of the intersection, is taken into the integrated one. The hard clustering is kept in results from integrating the hard clustering results. If the heard clustering results are integrated with the fuzzy clustering results, the process that is presented in Fig. 12.11 is adopted.

The process of integrating the two hierarchical clustering results into a single version is illustrated in Fig. 12.12. In Fig. 12.12, the white circle indicates the cluster with nested ones and the black circle indicates the final cluster. Only clusters with no nested ones, which are marked as black circles, are involved in the integration, and each of hierarchical cluster results are converted into flatted ones by listing the only final clusters. The two hierarchical clustering results are integrated into one flat clustering results by involving the only final clusters. The hierarchical clusters are constructed by merging the final clusters in the bottom-up direction.

```

integrateTwoClusterResults(List clusterList1, List clusterList2){
  do until isEmpty(clusterList1) or isEmpty(clusterList2){
    clusterPair = findClusterPairWithMaxIntersection(clusterList1,clusterList2)
    //clusterPair ← cluster1, cluster2
    cluster.add(clusterPair.getUnion());
    clusterList1.remove(clusterPair); //remove cluster 1 from clusterList1
    clusterList2.remove(clusterPair); //remove cluster 2 from clusterList2
    integratedClusterList.add(cluster);
  }
  if(isNotEmpty(clusterList1)) restItem.add(clusterList1.getAllItems());
  if(isNotEmpty(clusterList2)) restItem.add(clusterList2.getAllItems());
  for each item in restItem{
    integratedClusterList.arrangeItem(item);
  }
  return integratedClusterList;
}

```

Fig. 12.11 Fuzzy clustering integration

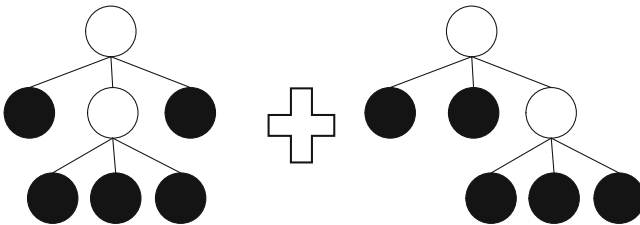


Fig. 12.12 Hierarchical clustering integration

Let us make some remarks on integrating multiple clustering results into one. The integration of the clustering results is used for combining multiple clustering algorithms. The ensemble learning was mentioned as a learning paradigm where multiple machine learning algorithms are combined for more reliable classification and clustering. The ensemble learning is studied for reinforcing mainly the supervised learning by combining multiple existing ones. It will be mentioned in the unsupervised learning, as well as the supervised learning in Chap. 14.

12.5 Summary and Further Discussions

This part provides the understanding of the advanced machine learning algorithms as the advanced part. The ensemble learning will be considered for making the strong machine learning algorithm by combining multiple weak machine learning algorithms. The semi-supervised learning that is used for the classification and the regression is the learning paradigm using the unlabeled examples as well as

labeled ones. The additional learning paradigms to the supervised learning and the unsupervised learning are the temporal learning for analyzing temporal sequence which is used for the speech recognition and the signal processing and the reinforced learning which is used for implementing an autonomous robot. This part is intended to describe the various kinds of advanced learning as the advanced course about machine learning algorithms.

This part consists of the four chapters. In Chap. 13, we mention various types of ensemble learning as the combination schemes of multiple supervised learning algorithms. In Chap. 14, we describe the some advanced learning algorithms: semi-supervised learning and ensemble unsupervised learning. Chapter 15 covers the temporal learning by the HMM (Hidden Markov Model) and its application to the text topic analysis. In Chap. 16, we mention the reinforcement learning with its application to the classification and the regression, in order to understand it easily.

Let us mention the Apriori algorithm as the approach to the data association. Most frequent items are generated from the item set which are given as the input. All possible pairs are generated from the selected items, maximal frequent item pairs are selected based on the support, and association rules are extracted from the selected item pairs based on the confidence. Selecting item sets that add one more item and deciding association rules are iterated by incrementing set cardinality by one. The frequent item set whose cardinality is n is also the frequent item with $n - 1$ cardinality according to the principle in the Apriori algorithm [2].

Let us consider implementing the data association by the clustering algorithms and the classification algorithms. The implementation of the data association was performed by extracting frequent items sets and doing association rules for each frequent item set. In this scheme of implementing the data association, the frequent item sets are constructed by clustering individual items, and each permutation of each cluster is classified into association rule or not. It is required to gather sample permutations that are labeled with association rule or non-association rule for implementing it with the scheme. The Apriori algorithm or its variants are used for implementing the data association in the traditional scheme.

Let us consider encoding an association rule into a numerical vector for selecting a permutation by a classification algorithm. We mentioned above the process of clustering data items and generating association rule from each cluster. The association rule is assumed as the conditional part that consists of a single item and the causal part that consists of more than one item, and the similarities and the confidences are features for encoding an association rule into a numerical vector. The association rules that are labeled with interesting or non-interesting are collected as the training examples. The association rules that are classified into interesting are generated as the output.

References

1. J. Gonzalo, J. Artiles, F. Verdejo, A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval* **12**(4), 461–486 (2009)
2. J. Han, M. Kamber, J. Pei, *Data Mining: Concept and Techniques* (Morgan Kaufmann, 2012)
3. T. Jo, The Implementation of Dynamic Document Organization using Text Categorization and Text Clustering, Ph.D. Dissertation of University of Ottawa, 2006
4. T. Jo, Modification of clustering algorithms for text clustering. *Int. J. Comput. Sci. Software Technol.* **3**(1), 21–33 (2010)
5. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, 2018)
6. T. Jo, Clustering texts using feature similarity based AHC algorithm. *J. Intell. Fuzzy Syst.* **35**, 5993–6003 (2018)
7. T. Jo, Text classification using feature similarity based K nearest neighbor. *AS Med. Sci.* **3**(4), 13–21 (2019)
8. T. Jo, M. Lee, in *The Evaluation Measure of Text Clustering for the Variable Number of Clusters*. *Lecture Notes in Computer Science*, vol. 4492, pp. 871–879 (2007)

Part IV

Advanced Topics

Chapter 13

Ensemble Learning



13.1 Introduction

The ensemble learning is defined as the learning style of combining multiple weak machine learning algorithms into a strong one. We consider the three schemes of combining multiple machine learning algorithms: voting, expert gate, and cascading. The meta-learning as the state of the art of ensemble learning is the learning process of coordinator in combining them. We consider various types of partitions such as the training set partition and the dimension partition, for doing the more efficient ensemble learning. This section is intended to describe the combination schemes, meta-learning, and the partition schemes of understanding the ensemble learning.

Let us mention some schemes of combining machine learning algorithms. The voting is to make the final answer, referring ones of committee members by the coordinator. The expert gate is to do it by determining an expert among the committee members to the input vector. The cascading is to find the answer by progress serially from the simplest approach to the most sophisticated one. We may consider the hybrid combination by mixing the three schemes.

Let us consider the learning of the coordinator, called meta-learning, for more advanced combinations of machine learning algorithms. The meta-learning is to learn examples for more sophisticated learning. The coordinator in the three combination schemes need to learn examples for playing more reliability its own role. We need to define the input and output and collect training examples differently, depending on the given combination scheme. More learning strategies will be explained in detail in Sect. 13.3.

Let us consider the four partition aspects for combining multiple machine learning algorithms. A training set may be partitioned into subsets each of which is allocated to each machine learning algorithm. When the input dimension is huge, an attribute set may be partitioned into subsets each of which is considered in each machine learning algorithm. We may use multiple machine learning algorithms with their simple architectures instead of a single machine learning algorithm with its

complicated one. Through the parallel processing or the distributed processing, resource which is necessary for executing machine learning algorithms may be partitioned.

The goal of this chapter is to understand the ensemble learning as the combination of weak learning algorithms into a strong one. We will understand some combination schemes of existing machine learning algorithms. We need to understand the meta-learning which is applicable to the coordinator for providing the adaptability. We will also understand the partitions of training examples, input dimensions, and architectures for implementing more efficient ensemble learning. We will make further discussions on what is studied in this chapter for providing more advanced combination schemes of learning algorithms and organizing machine learning algorithms for processing big data.

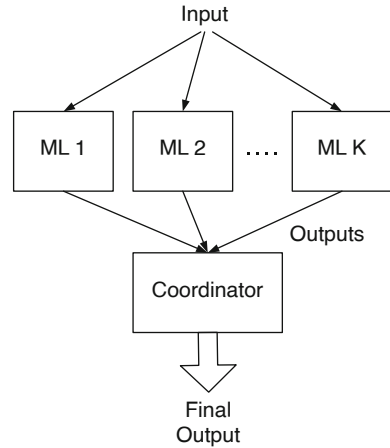
13.2 Combination Schemes

This section is concerned with the combination schemes of multiple machine learning algorithms. In Sect. 13.2.1, we mention voting as the most popular combination scheme. In Sects. 13.2.2 and 13.2.3, we describe the expert gate and the cascading as other combination schemes. In Sect. 13.2.4, we mention the cellular models which are suitable for the big data processing. This section is intended to present the four combination schemes of multiple machine learning algorithms.

13.2.1 Voting

The voting is defined as the majority or the average of outputs of multiple classifiers for generating a final output as shown in Fig. 13.1. The M machine learning algorithms are given as committee members and the coordinator is given for voting or averaging the outputs of committee members, in Fig. 13.1. When a novice input is given, it is classified by the committee members and the final output is made by the coordinator. We will mention some variants which are derived from the voting, in addition. This section is intended to describe the organization of machine learning algorithms, the classification process, and some variants.

Let us view the organization of machine learning algorithms in the combination scheme, called voting. Whether they are homogenous algorithms with their different parameters or heterogeneous ones, the M machine learning algorithms are given currently. The committee members classify each item directly, and the coordinator makes the final decision by collecting outputs from them. The role of the committee members is to classify each item, and the role of coordinator is to make the final output based on the outputs. In this combination scheme, the committee members stand in the front part, and the coordinator does in the tail part.

Fig. 13.1 Voting

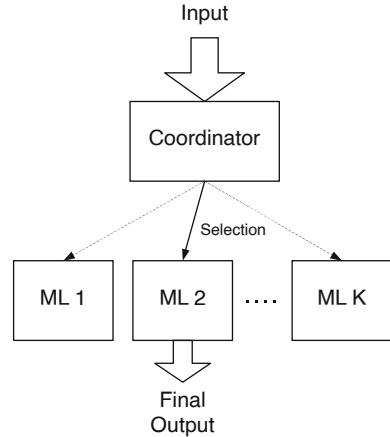
Let us explain the process of classifying an item by voting the machine learning algorithms, under the assumption that the committee members learn the training examples. A novice example is given as the input and the committee members generate their own outputs, $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_M$. The vector, $[\hat{y}_1 \ \hat{y}_2 \ \dots \ \hat{y}_M]$, which consists of outputs of committee members, is given as the input to the coordinator. The final output, $\hat{y}_f$, is generated by voting the M outputs in the classification task. In the regression task, the final output is averaged over the M outputs.

Some variants may be derived from the combination scheme, voting, which is illustrated in Fig. 13.1. The committee members may be discriminated by their weights rather than their equal portions. The meta-learning where the input vector is given as the output values of the committee members is performed to the coordinator. The input vector may be partitioned into sub-input vectors, and each committee member learns its own sub-input. The training set is partitioned into subsets with overlapping or no overlapping, and each committee member learns its own subset.

Let us make some remarks on the voting which is a combination scheme of multiple machine learning algorithms. The coordinator locates in the tail in the combination scheme; after the committee members classify the item, it makes the final decision. It votes the outputs of committee members or average over them. The voting or the averaging may be replaced by the learning process in the coordinator. This kind of learning is performed by the coordinator and called meta-learning.

13.2.2 Expert Gates

This section is concerned with the second scheme of multiple machine learning algorithms, called expert gates as shown in Fig. 13.2. The voting, which is mentioned in Sect. 13.2.1, is the combination scheme where the coordinator collects

Fig. 13.2 Expert gates

output values of the committee members. The expert gates which are mentioned in this section are one where the coordinator nominates an expert among the committee members to the given input data. In the voting, all of the committee members are involved in classifying a data item, whereas in the expert gates, only one committee member is involved in doing that. This section is intended to describe the second combination scheme of multiple machine learning algorithms, called expert gates.

This combination consists of M machine learning algorithms as committee members and the coordinator, like the case of voting. The multiple machine learning algorithms which classify items are called committee members in the voting, whereas are called experts in the expert gates. Each expert classifies data items with its expert area in the input space, and when the input vector is given, an expert is nominated by the coordinator. In the voting, all of the committee members classify the data item at same time, whereas in the expert gate, only one which is nominated by the coordinator does it. The coordinator locates in the tail part in the voting, whereas it locates in the front in the expert gate.

Let us mention the process of classifying a data item by the combination of multiple machine learning algorithms, called expert gates. It is assumed that the coordinator for selecting an expert and the experts for classifying are available. A particular item is given as the input vector, and an expert is nominated by the coordinator. The item is classified into a particular category by the expert. The expert is the decision maker in this scheme, whereas the coordinator is the decision maker in the previous scheme.

Let us consider some variants which are derived from the expert gate. There are various schemes of allocating the training examples to the experts. More than one expert may be selected by the coordinator. The fuzzy selection of experts may be performed by assigning different weights to experts. It is possible to organize experts hierarchically.

Let us make some remarks on the expert gate as the scheme of combining multiple machine learning algorithms. In this combination scheme, the coordinator

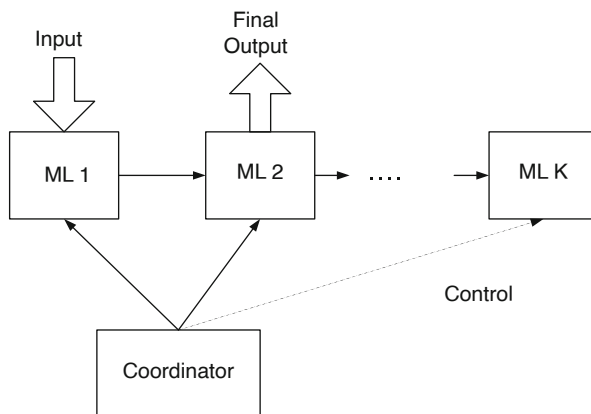


Fig. 13.3 Cascading

is given as a classifier which assigns one of the experts to each item. The training set is partitioned into subsets as many as experts and the expert whose partition is most similar to the novice item is nominated. The training set may be partitioned with the overlapping and more than one expert may be nominated. The training examples are clustered into clusters as many as experts, ignoring their target labels, and the expert where its cluster is most similar as the novice item is selected.

13.2.3 Cascading

The third scheme of combining multiple machine learning algorithms with each other is illustrated in Fig. 13.3. In the cascading, machine learning algorithms are combined serially as shown in Fig. 13.3, whereas in the voting and the expert gate, they are combined parallel as shown in Figs. 13.2 and 13.3. The coordinator in the cascading is given as the controller which decides whether a novice example transferred to the next classifier, or not. The simplest machine learning algorithm locates in the front stage, and the most complicated one does in the tail stage, in organizing machine learning algorithm. This section is intended to describe the third scheme of combining multiple machine learning algorithms, called cascading.

Let us mention how to organize machine learning algorithms in the cascading. The machine learning algorithms are arranged into a serial form. The most left machine learning algorithm is simplest among them, and the most right one is most advanced one. The coordinator which is next to the serial line is the controller which decides whether an item is transferred to the next one, or not. Organizing different kinds of machine learning algorithms is the popular trend in this scheme.

Let us mention the process of classifying a novice item in the combination which is presented in Fig. 13.3. It is assumed that the first machine learning is given as the

simplest one, and the last one is given as the most advanced one. A novice item is classified by a machine learning algorithm, and the coordinator decides whether it is transferred to one in the next state, or the classified category is taken as the final one. We need to define the certainty of the output value for making the decision in the coordinator. In the categorical score which is given as a normalized value between zero and one, the value around 0.5 is set as a uncertain value, and one which is close to 0.0 or 1.0 is set as a certain value.

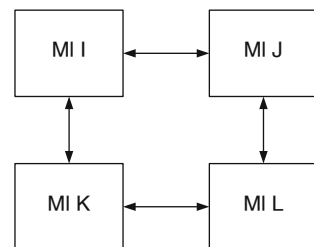
Let us mention variants which are derived from the combination scheme of the machine learning algorithms, called cascading. The output values from the initial classifier to the current one are voting as a final one, instead of taking the output value of the current one as the final one. We may consider the multiple cascading where values from the cascading lines are voted, as several serial lines of machine learning algorithms. Cascading may be given as a tree, instead of a linear line, where a branch of subsequent machine learning algorithms is selected, depending on the input vector. The Adaboost becomes the scheme where dynamic weights are assigned to machine learning algorithms, depending on their learnability to the input vector.

Let us make some remarks on the scheme of combining multiple machine learning algorithms as a serial line, called cascading. The voting and the expert gate belong to the parallel combination, whereas the cascading belongs to the serial combination. The simplest machine learning algorithm and the most advanced one are arranged in the front and the tail of the serial line, respectively. An item is classified by the simplest one, and depending on the certainty, it decides whether the classified label from the current classifier or the novice item is transferred to the next one. The label that is classified by only one among the committee members is the final decision in the standard version.

13.2.4 Cellular Model

This section is concerned with the cellular model of machine learning algorithms, which is illustrated in Fig. 13.4. We studied the three combination schemes of machine learning algorithms: voting, expert gate, and cascading. It is assumed that simple machine learning algorithms are given as a group and their cooperation is

Fig. 13.4 Cellular models



studied in this section. The relationships among machine learning algorithms are more free in the cellular model than those in the three combination schemes. This section is intended to describe the organization of machine learning algorithms.

Let us mention some simple machine learning algorithms before doing the process of classifying an item in this combination of machine learning algorithms. The 1-NN which was mentioned in Sect. 5.2.4 and the KNN which was mentioned in Sect. 5.3 belong to the simple machine learning algorithms. The Bayes classifier where a Gaussian distribution is assumed for each category belongs to the simple machine learning algorithm. Even if it is slightly complicated than the Bayes classifier and the KNN, the decision tree is considered as a simple machine learning algorithm. The Perceptron in Sect. 8.2.1 may be regarded as a simple linear classifier.

Let us mention the process of classifying an item in the organization of machine learning algorithms. It is assumed that extremely many labeled training examples are given to train individual machine learning algorithms, and only some are allocated to each machine learning algorithm, at random. A novice item is classified by a machine learning algorithm, its labels into which it is classified by other machine learning algorithms are collected, and the final decision is made by referring the labels. The cellular model of machine learning algorithms is suitable for learning the training examples which are given as big data.

A machine learning algorithm cooperates with others, concerned with the data classification. It is assumed that the training set is too big to learn all of them, so each machine learning algorithm learns its own subset. Because all of machine learning algorithms are simple, they are organized easily in parallel. The voting or the expert gate is selected or both of them are mixed with each other. In the cellular model, individual machine learning algorithms which learn their own subsets are given as independent ones, and cooperate with others in classifying data items.

Let us make some remarks on the cellular mode which is described in this section. It is intended for the powerful performance from the cooperation of simple machine learning algorithms, based on the collective intelligence. Each machine learning algorithm is assumed to be an independent entity which learns training examples and classifies items, not under the coordinator. Training examples are traded between two machine learning algorithms and they learn them incrementally. The labels into which an item is classified by other machine learning algorithms are referred for deciding the final output.

13.3 Meta-learning

This section is concerned with the meta-learning which is executed in the coordinator which organizes the machine learning algorithms. In Sect. 13.3.1, we explain the meta-learning in the combination of multiple machine learning algorithms, called voting. In Sects. 13.3.2 and 13.3.3, we describe the meta-learning of the coordinator in the expert gate and the cascading. In Sect. 13.3.4, we mention the meta-learning

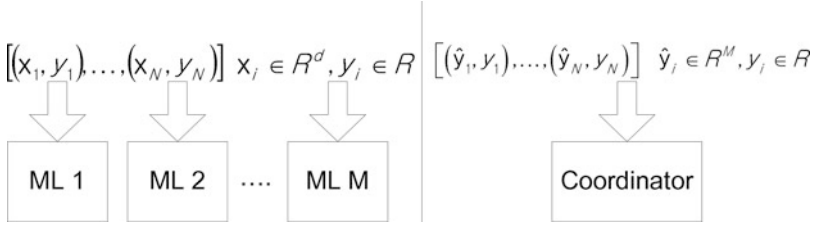


Fig. 13.5 Training examples for voting

in the cellular model which is installed in each machine learning algorithm. This section is intended to describe the meta-learning which is used for organizing the machine learning algorithms.

13.3.1 Voting

This section is concerned with the meta-learning which is applicable to the coordinator in voting the machine learning algorithms. The voting is the combination scheme where the coordinator makes the final decision by collecting outputs of committee members as described in Sect. 13.2.1. In the meta-learning, the outputs from the committee members are given as an input vector, and the final output which is decided by the coordinator is generated as the output. The committee members are trained with the training examples and ones for the meta-learning are derived by generalizing the committee members. This section is intended to describe the meta-learning for the coordinator in the voting.

The training examples for training the machine learning algorithms in the committee and the coordinator are illustrated in Fig. 13.5. It is assumed that the training examples which are given as the set, $Tr = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ where $x_i \in R^d$ and $y_i \in R$ are used for the regression task. The input vector and the output are defined to the coordinator, respectively, as $\hat{y}_i = [\hat{y}_{i1} \ \hat{y}_{i2} \ \dots \ \hat{y}_{iM}]$ which consists of the output values of the committee members and, in the case of M machine learning algorithms in the committee. The committee members are trained with the original training examples, whereas the coordinator is trained with the vectors, each of which consists of M output values. After training the committee members, the training examples which are used for training the coordinator are collected.

The learning process of the committee members and the coordinator is illustrated in Fig. 13.6. The training set, $Tr = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, is initial given and used for training the individual committee members. The training set, $Tr' = \{(\hat{y}_1, y_1), (\hat{y}_2, y_2), \dots, (\hat{y}_N, y_N)\}$, where $\hat{y}_i \in R^m$ and $y_i \in R$, is gathered by generalizing the committee members using the training examples. The coordinator is trained with the training set, $Tr' = \{(\hat{y}_1, y_1), (\hat{y}_2, y_2), \dots, (\hat{y}_N, y_N)\}$. The input

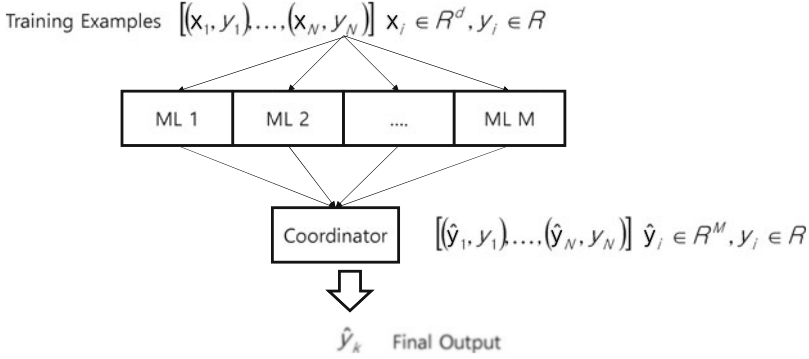


Fig. 13.6 Meta-learning process in voting

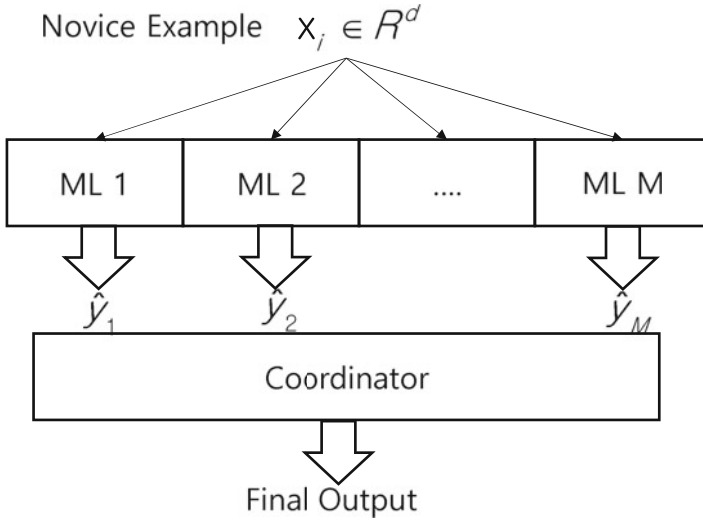


Fig. 13.7 Generalization in voting

vector of the coordinator is given as m dimensional vector whose dimensionality corresponds to the number of committee members, whereas the input vector of the committee members is given as a d dimensional vector, as an original vector.

The process of classifying a data item in the voting with the meta-learning is illustrated in Fig. 13.7. It is assumed that the coordinator learns the training examples each of which consists of answers of the committee members and its target label, and a novice input vector is given. The novice input vector is regressed by the committee members, and the coordinator collects the answers. The coordinator decides the final output by classifying the input vector which consists of the outputs which are generated by the committee members. In this case, the meta-learning is applied to the regression task; it may be applied to the classification task.

Let us make some remarks on the meta-learning which is applicable to the coordinator in the voting. The coordinator which makes the final answer based on the answers of the committee members is assumed to be a machine learning algorithm. Setting the number of machine learning algorithms in the committee less than the dimension of the input vector may become the method of reducing the dimension. Setting the number of committee members greater than the input dimension may become the method of mapping vectors into ones in another space. The role of meta-learning in the voting is to make the final decision based on the decisions of the committee members.

13.3.2 Expert Gates

This section is concerned with the meta-learning which is applicable to the coordinator in the expert gate. It was studied in Sect. 13.2.2, as a scheme of combining multiple machine learning algorithms. It is assumed that the coordinator which decides the expert among machine learning algorithms to the input vector is a machine learning algorithm, and the coordinator is trained with the input vectors for generating machine learning algorithm identifier as the output. A same input vector is given to both the machine learning algorithms and the coordinator, but their outputs are different. This section is intended to describe the meta-learning to the coordinator in the expert gate.

The training examples for training the experts are illustrated in the left of Fig. 13.8, and ones for training the coordinator are illustrated in the right of Fig. 13.8. Each training example which is given for the experts consists of the original input vector and its original target value. The input vector of each training example which is given for the coordinator is same to that of the expert, but its target label indicates an expert identifier. The role of the coordinator is to nominate a machine learning algorithm in the expert group. The task of the coordinator is viewed into the process of classifying each item into an expert identifier.

The process of learning the training examples in the coordinator and the experts is illustrated in Fig. 13.9. The experts are trained with the original training examples, and the answers are collected from them. Each input vector is associated with

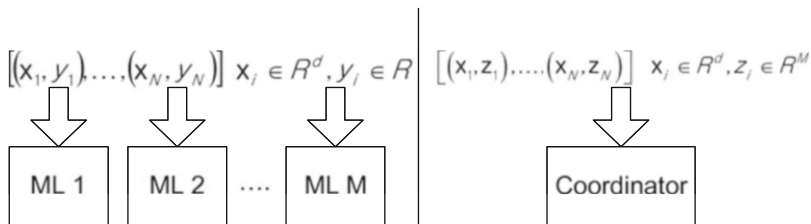


Fig. 13.8 Training examples for expert gates

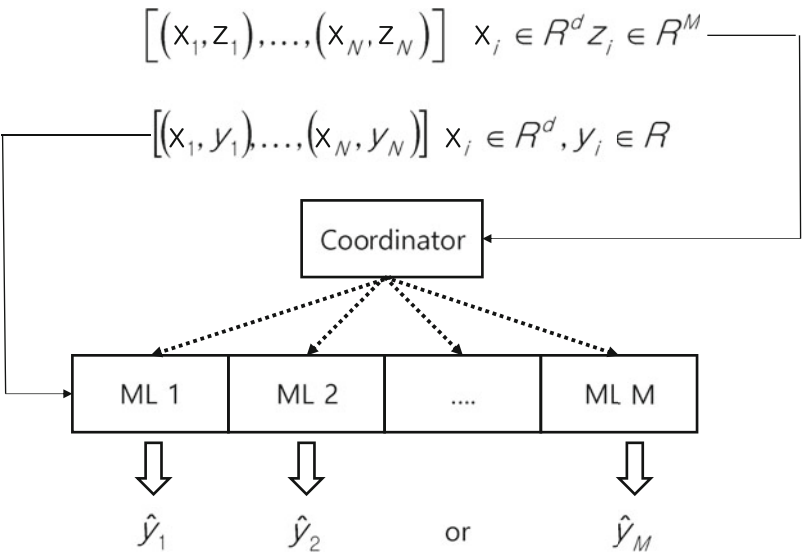


Fig. 13.9 Meta-learning process in expert gates

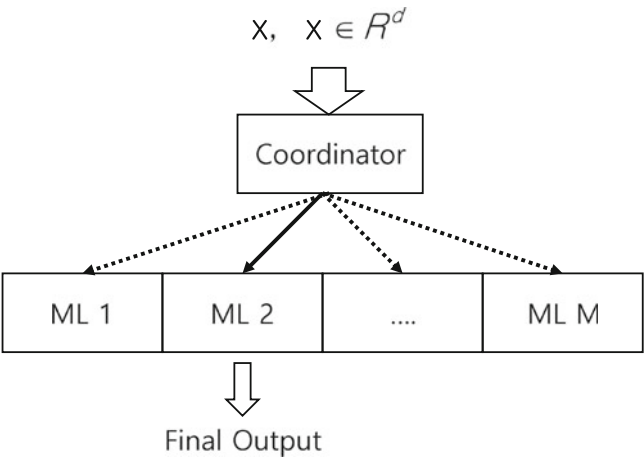


Fig. 13.10 Generalization in expert gates

the expert identifier which classifies it with its minimum error. Another set of the training examples which is used for training the coordinator is constructed. The coordinator nominates an expert who provides the most accurate answer for each novice item.

The generalization in the expert gate of machine learning algorithms is illustrated in Fig. 13.10. The machine learning algorithms which are given as experts are trained and the coordinator is also trained with the expert identifiers which are

given as the target outputs. A novice item is given as the input, and is classified into an expert identifier by the coordinator. The classification or the regression is performed by the expert which is nominated by the coordinator. The input vector is given identically to both the coordinator and the expert in this combination scheme.

Let us make some remarks on the meta-learning for the coordinator in the expert gate. The output of the meta-learning to the coordinator and the specific learning to the committee members is same to each other in the voting, whereas in the expert gates, the input of both is same to each other. The meta-learning follows the specific learning of the committee members and the experts in both the voting and the expert gate. The adaptability for deciding the final answers based on the outputs of the committee members is the intention of the meta-learning in the voting, whereas the adaptability for selecting an expert based on the input vector is the intention of the meta-learning in the expert gates. The meta-learning in this combination scheme is viewed into the classification of each item into one among the experts.

13.3.3 Cascading

This section is concerned with the meta-learning which is applicable to the coordinator in the cascading. We studied the meta-learning in the voting and the expert gate, respectively, in Sects. 13.3.1 and 13.3.2. The meta-learning in the cascading is viewed into the binary classification, which each item is classified into pass to the next classifier or current decision. The answer of from the current machine learning algorithm, or both of the input vector and the current answer is given as the input of the meta-learning. This section is intended to describe the meta-learning of the coordinator in the cascading.

The training examples which are used for the machine learning algorithms in the serial structure and the coordinator are illustrated in Fig. 13.11. The input is given as a d dimensional vector, and the output is given as a real value for the machine learning algorithms in the serial structure. Both the input and the output of the machine learning algorithms are given as the input for the coordinator; the input of coordinator becomes $d + 1$ dimensional vector in this case. The output

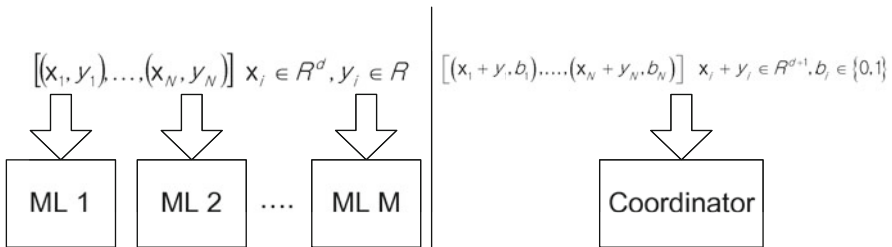


Fig. 13.11 Training examples for cascading

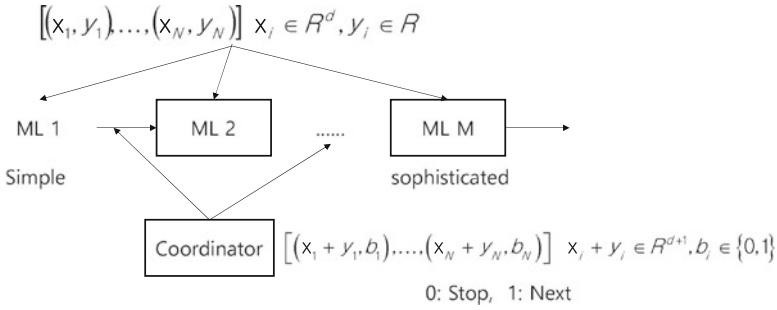


Fig. 13.12 Meta-learning process in cascading

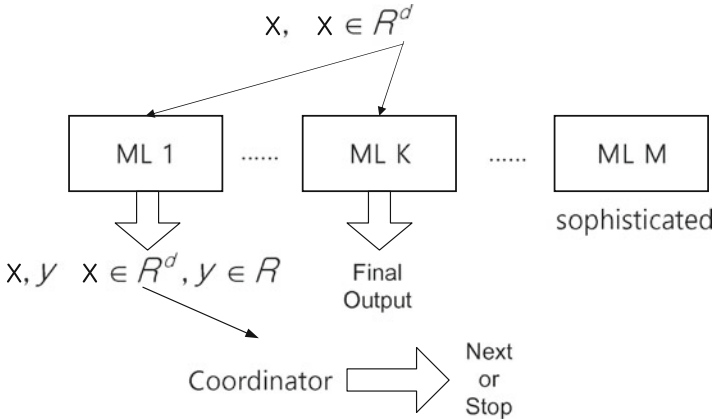


Fig. 13.13 Generalization in cascading

of coordinator is the binary value, 0 which indicates transfer to the next machine learning algorithm, or 1 which indicates the output of the current machine learning algorithm as the final one. The role of the coordinator is to classify each item which augmented with its desirable output value into the decision by the current output value or the transit to the next classifier.

The training examples are illustrated for training both the machine learning algorithms and the coordinator, in Fig. 13.12. The machine learning algorithms which are combined serially are trained with the original training examples. The positive examples which are classified clearly and the negative examples as the others are collected from each classifier. Ones which are classified by the last classifier belong to the negative class. The collected examples whose dimension is $d + 1$ are used for training the coordinator.

The generalization phase of the coordinator in the cascading is illustrated in Fig. 13.13. The novice input vector is given in a machine learning in the serial line, and the output is generated by it. The input vector and its output are generated by the current machine learning algorithm and are given as the input to the coordinator. The

input which is augmented with the computed output is classified into the decision of the current output as the final output or transfer of the input vector to the next machine learning algorithm. This combination scheme of the machine learning algorithms is called cascading; this becomes the basis for implementing the boosting algorithm.

Let us make some remarks on the meta-learning which is given for the coordinator in the cascading. The output value from the current classifier influences on the decision whether the input is transferred to the next classifier or the current output is decided as the final one. The k th classifier provides the desirable answer, the input vector and the output value which is provided by the previous classifier is labeled as the negative class which indicates the transit to the next one, and the input vector and the output value which is provided by the current one is labeled with the positive class which indicates the final decision. In this case, the $k - 1$ negative examples and only one positive example are generated; much more negative examples than the positive example are generated in this combination. The certainty of the uncertainty of the output value is the facto for deciding either of the two cases.

13.3.4 Cellular Model

This section is concerned with the co-learning as a kind of the semi-supervised learning or the meta-learning. It is usually applicable to two machine learning algorithms. The meta-learning between two machine learning algorithms is for exchanging the training examples with each other. Each machine learning algorithm consists of the dual learning systems: the meta-learning for deciding whether a training example is accepted or rejected and the main learning for classifying a novice item. This section is intended to describe the meta-learning in each machine learning algorithm.

The training examples for the both learning types are illustrated in Fig. 13.14. Each machine learning has its dual learning systems: one for the classification or

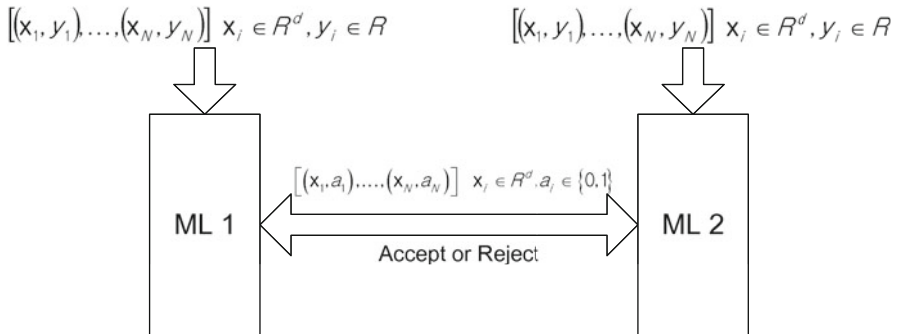


Fig. 13.14 Training examples for co learning

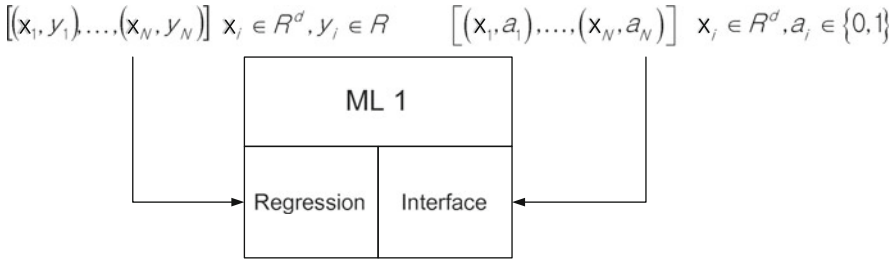


Fig. 13.15 Meta-learning process in co learning

the regression and the other for deciding the acceptance of a training example. The meta-learning in the combination is viewed as the binary classification where each training example which is provided by the opposite is classified into acceptance or rejection. The training examples which are classified into acceptance are used for training the machine learning algorithm additionally. The meta-learning is intended for exchanging the training examples between two machine learning algorithms, mutually.

The dual learning systems in each machine learning algorithm are illustrated in Fig. 13.15. The learning system in the left part is for doing its own task, regression, and one in the right part is for deciding whether it accepts or rejects a training example from another. After learning its own training examples in the left part, the label “acceptance” is assigned to ones with more training error, and the label “rejection” is assigned to the rest. The training examples which are labeled with acceptance indicate the area which need more training examples from other machine learning algorithms, and ones which are labeled with rejection does the area where no more training examples are needed. Labeling the training examples so is intended to supply more training examples from others for the area with its relative poor performance.

Let us mention the generalization of the meta-learning in the right part of the machine learning algorithm as shown in Fig. 13.16. The classification or the regression of each item in the left part is its own task of the machine learning algorithm. The role of the right part is to judge whether the external training examples are accepted or rejected. Ones which are labeled with acceptance are added to the existing ones, and they are learned in the left part, additionally. In this case, the incremental learning which is the process of learning only added ones is needed.

Let us remark on the co-learning between the two machine learning algorithms. It is the free interaction between them as unorganized machine learning algorithms. The co-learning is intended to deal with the training examples which are given as big data; it is the data collection with the four properties: extremely high volume, high velocity, high veracity, and high variety. In the environment of the big data, many machine learning algorithms need to be created and they learn some training examples, exchanging their training examples with others. Each classifier has its

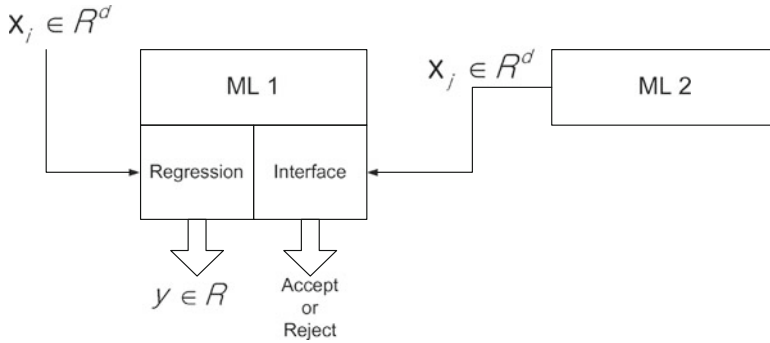


Fig. 13.16 Generalization in co learning

dual learning systems; one is for doing its own task and the other is for exchanging training examples with others.

13.4 Partition

This section is concerned with the views of partitioning a classification task for implementing the ensemble learning. In Sect. 13.4.1, we mention the partition of the training set into several subsets. In Sect. 13.4.2, we consider the partition of the attribute set as a set of features into subsets. In Sect. 13.4.3, we mention the partition of a complicated machine learning algorithm into several simple ones. In Sect. 13.4.4, we explain the implementation of ensemble learning in the parallel or distributed environment.

13.4.1 Training Set Partition

This section is concerned with the partition of the training set into subsets in implementing the multiple learning system. The schemes of combining multiple machine learning algorithms were entirely studied in the previous sections. In using multiple machine learning algorithms, it is more efficient to train each of them with only a subset, rather than the entire set. There are various schemes of partitioning the training set into subsets, depending on the view. This section is intended to study the dichotomies of setting the types of the partition schemes.

The exclusiveness or the overlapping are considered as a dichotomy of partitioning the training set, as shown in Fig. 13.17. It is assumed that the training set, Tr , is partitioned into the k subsets, $SubTr_1, SubTr_2, \dots, SubTr_k$. The exclusive partition is characterized mathematically as Eq. (13.1),

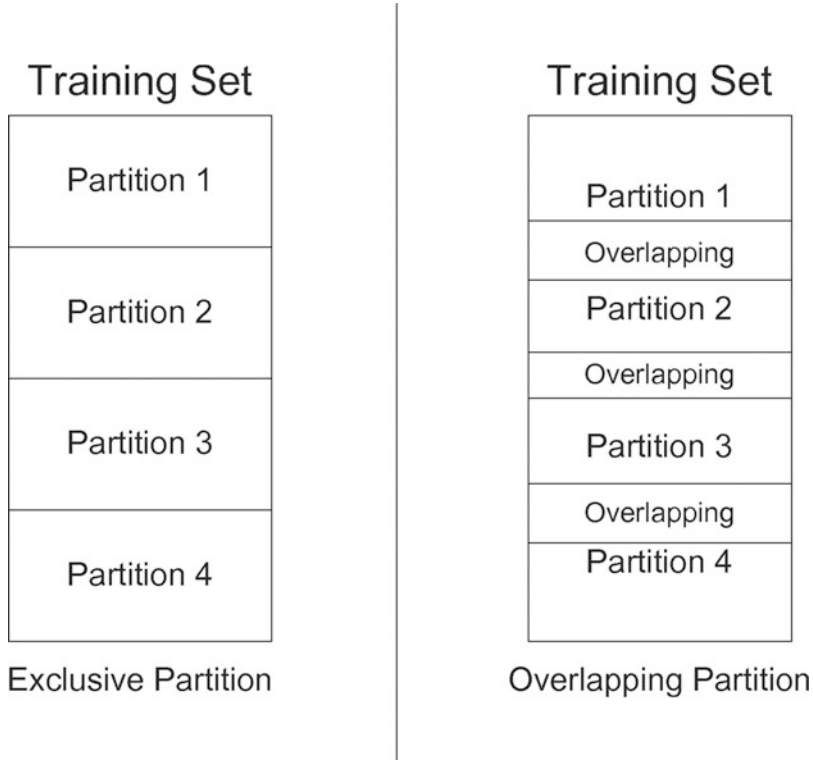


Fig. 13.17 Exclusive vs overlapping partition

$$\forall_{i,j} \text{SubTr}_i \cap \text{SubTr}_j = \emptyset \quad (13.1)$$

and the overlapping is done as Eq. (13.2),

$$\exists_{i,j} \text{SubTr}_i \cap \text{SubTr}_j \neq \emptyset \quad (13.2)$$

The degree between the exclusiveness and the overlapping is decided by the total cardinality of the intersection of all possible subsets, as shown in Eq. (13.3),

$$\sum_{i < j}^k |\text{SubTr}_i \cap \text{SubTr}_j| \quad (13.3)$$

The exclusive partition, the strong overlapping, and the weak overlapping are adopted, respectively, in the expert gate, the voting, and the cascading.

The choice of the random selection or the special selection becomes another type of dichotomy. Some may be selected among the training examples at random for building a subset. The clusters of the training examples with regardless of

their categories are built, depending on the similarities among them, and each cluster becomes a subset, in the special selection. The dichotomy in partitioning the training set into subsets is whether the similarities among the training examples are considered or not. Because clustering data items is performed in the special selection, it has more reliability of training subsets but takes higher computation cost.

Let us consider weighting the training examples, rather than partitioning the training set into several subsets. We mentioned the partition of the training set and the assignment of each subset to each machine learning algorithm. We may assign different weights as many as the machine learning algorithms in the committee to each training example. Each machine learning algorithm in the committee learns proportionally to the weights of the training examples. In implementing the Adaboosting of the machine learning algorithm, the weights are decided automatically.

Let us make some remarks on the partition of the training set into subsets which are used for training each machine learning algorithm. All of the training examples may be used for training each one, or only subset may be used for doing so. Once it is decided to use only subset for training each one, we need to decide whether the training set is partitioned into subsets, exclusively or with overlapping. In the special partition, the training examples are clustered with regardless of their target labels. The weights are assigned to the training examples, depending on their learnability, in training each machine learning algorithm.

13.4.2 Attribute Set Partition

This section is concerned with the partition of the attribute set for implementing the combination of machine learning algorithms. In Sect. 13.4.1, we studied the partition of the training set which is called horizontal partition. The attribute set of the training examples is partitioned into multiple subsets in this section. If many training examples each of which has many attributes are initially available, it is possible to partition both the training set and the attribute set. This section is intended to describe the schemes of partitioning the attribute set.

Let us consider the partition of the attribute set into subsets by selecting some attributes, randomly. The attribute set is notated by $A = \{a_1, a_2, \dots, a_d\}$, and its subsets are notated by $SubA_1, SubA_2, \dots, SubA_m$ $|SubA_i| < d$. The dimension, d , of the input data is reduced into the size of the attribute subset for each machine learning algorithm in the ensemble learning. The overlapping partition is characterized as $\exists_{i,j} SubA_i \cap SubA_j \neq \emptyset$, whereas the exclusive partition is characterized as $\forall_{i,j} SubA_i \cap SubA_j = \emptyset$, and the uniform partition is characterized as $|SubA_1| = |SubA_2| = \dots = |SubA_m|$, whereas the variable partition is characterized as $\exists_{i,j} |SubA_i| \neq |SubA_j|$, as the partition policies. In future, we consider optimizing the attribute subsets for maximizing the classification performance.

It is possible to represent the attribute values into numerical vectors. The training set is expressed as a matrix where each row indicates an individual training example, and each column is an attribute. The row vector becomes an individual training example, and a column vector becomes values within an attribute. An individual training example becomes a feature for representing the attribute into a numerical vector. If they are represented so, it is possible to compute the similarities among the attributes.

The attribute set is partitioned into subsets by clustering the attributes by their similarities. We mentioned above the process of representing the attributes into numerical vectors, so it is possible to compute the similarities among attributes using the cosine similarity or the inverse Euclidean distance. The attributes are clustered by a clustering algorithm whose essential operation is the similarity metric. Each cluster of attributes is a subset of attributes which is assigned to each machine learning algorithm. The similarity metric which considers the similarities among attributes is proposed, recently [1].

Let us make some remarks on the partition of the attribute set into subsets in applying the multiple machine learning algorithms to a given problem. When the training set is viewed as a matrix where each row is a training examples and each column is an attribute, the partition of the training set into subsets is the horizontal partition, whereas the partition of the attribute set into subsets is the vertical one. The number of subsets in partitioning the attributes is 2^d . The attribute set is usually partitioned into subsets with the constant size and the exclusiveness for the easiness and the simplicity. We may consider the process of optimizing the partition of the attribute set with the trial and error.

13.4.3 *Architecture Partition*

This section is concerned with the partition of the complicated architecture of a single machine learning algorithm into simple architectures of multiple machine learning algorithms. In using the multiple machine learning algorithms for the classification and the regression, the training set and the attribute set are partitioned into subsets in the previous sections. The idea of the architecture partition is to use the multiple machine learning algorithms with their simple architectures, instead of a single machine learning algorithm with its complicated architecture. The multiple Perceptrons may be used, instead of a single MLP (Multiple Layer Perceptron), for example. This section is intended to present the cases of using multiple machine learning algorithms with their simple architectures.

The random forest is the typical case of partitioning the architecture into sub-architectures in constructing the decision tree. We studied the random forest where each decision tree is constructed by a training subset, in Sect. 7.4.3. The architecture of each tree is simpler than that of the decision tree which is constructed by all of the training examples. In constructing the random forest, the decision tree is not

manually decomposed into subtrees, in fact. The random forest is the results from partitioning manually the training set into subsets.

Let us consider the multiple Perceptrons which replace the single MLP. Solving the exclusive OR problem which is not solved by a single Perceptron by multiple Perceptrons is the motivation for inventing the MLP. Inventing the MLP by Rumelhard in 1982 opened the revival of the research on the neural networks. The idea of the multiple Perceptrons shows the decomposition of the three layered architecture into several two layer architectures. The output values from the multiple Perceptrons are like ones of the hidden nodes in a single MLP.

Let us consider the multiple Naive Bayes models as the replacement of a single Bayesian Networks. The choice of a single complicated machine learning algorithm and multiple simple ones is the dichotomy in designing a classification system or a regression system. The training set or the attribute set is partitioned into subsets, and for each Naive Bayes, the likelihoods of attribute values are computed using its own subset of the training examples. One of the schemes which is mentioned in Sect. 13.1 is selected for deciding the final output. The multiple steps are assigned in the input encoding and the output decoding are assigned to a single simple machine learning algorithm in the deep learning.

Let us make some remarks on the partition of a complicated architecture into multiple simple architectures of the machine learning algorithm. Simple architecture decision trees are constructed by partitioning a training set into subset, in the random forest. The architecture is partitioned automatically by partition manually the training set. It is possible to construct random forest by partitioning the attribute set into subsets, called vertical partition. In this section, the direction of solving the complicated problem by multiple simple approaches, rather than a single complicated one, is presented.

13.4.4 Parallel and Distributed Learning

This section is concerned with the implementation of multiple machine learning algorithms in the parallel and distributed computing. They were constructed in 1990s for making the high performance. We need to implement multiple machine learning algorithms as a parallel algorithm or a distributed algorithm. The machine learning algorithms exist as independent ones in this environment, and they are reinforced through the fee interaction of them. This section is intended to mention the parallel machine learning algorithm and the distributed one as the cellular form of machine learning algorithms.

The process of implementing the multiple machine learning algorithms in the parallel computing is illustrated in Fig. 13.18. Before this chapter, it was assumed that a single machine learning algorithm is given as a single process. Each machine learning algorithm is implemented independently of others on its own processor. The training set is partitioned into subsets and each machine learning algorithm is

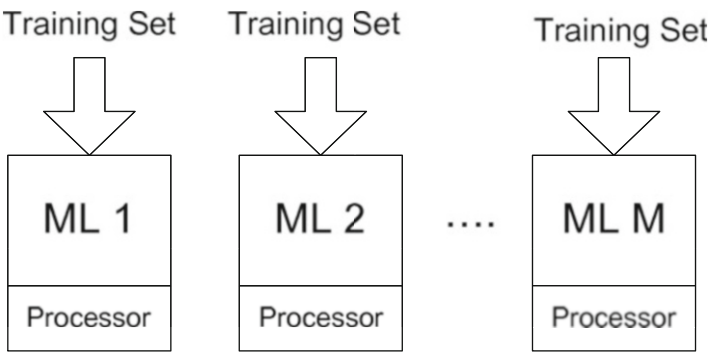


Fig. 13.18 Parallel learning

trained independently of others with its own subset. The multiple machine learning algorithms are executed independently of others in the parallel computing.

The distributed computing may be considered as another type of the high performance computing. It is the environment for doing the computing tasks in the resources which are distributed over connected systems. The use of multiple processors spans over multiple computers which are connected with each other by the Internet. The multiple machine learning algorithms are implemented independently in the distributed computing. The independent execution of multiple algorithms is the benefit from the parallel and distributed computing.

The cellular machine learning algorithms are proposed for dealing with the training examples which are given as the big data. It is defined as the collection of data items with the four characteristics: high volume, high velocity, high variety, and high veracity. It is impossible to digest this kind of the training examples by only one machine learning algorithm. In the Internet based computing, such as the cloud computing, many different kinds of the machine learning algorithms are created and each of them is trained with only small portion of the data collection. The paradigm of the machine learning algorithms is needed for dealing with the kind of data.

Let us make some remarks on the execution of the machine learning algorithms in the parallel computing or the distributed computing. The ensemble learning is easily implement in the environment and it is popular to use multiple machine learning algorithms for solving problems rather than a single machine learning algorithm. The deep learning algorithms are applied for real problems such as the image classification as the popular trend. The deep learning algorithms are implemented as fast version in the parallel computing or the distributed computing. We consider the evolution of the machine learning algorithms, following the change of the computing environment.

13.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We mentioned the three schemes of combining multiple machine learning algorithms for implementing the ensemble learning, voting, expert gate, and cascading. We described the meta-learning which is the learning of coordinator, for implementing the coordination of multiple machine learning algorithms for each combination scheme. We mentioned the partition of the training set, the attribute set, and the architecture, for simplifying each machine learning algorithms. In this chapter, we studied the three combination schemes, the meta-learning, and the partition aspects.

The combination of same type of machine learning algorithms is called homogeneous combination. We studied the three main combination schemes in this chapter. The combination of multiple machine learning algorithms which are given as SVMs for implementing a multiple classification system is the example of homogeneous combination. Multiple machine learning algorithms in same type are discriminated by their different external parameter values and subset of training examples are allocated to each machine learning algorithm. The random forest in Sect. 7.4.2 is also the example of homogenous combination of decision trees.

The heterogeneous combination is one of different types of machine learning algorithms. Above, we mentioned the homogeneous combination which is one of same type of machine learning algorithms with their different parameter values. The combination of the decision tree, the Naive Bayes, and the SVM for implementing a classification system is the typical instance of the heterogeneous combination. Using the Naive Bayes and the EM algorithm for implementing the text classification as the semi-supervised learning belong to the heterogeneous combination. It is further divided into the combination of different supervised learning algorithms, the combination of different unsupervised learning algorithms, and one of a supervised learning algorithm and an unsupervised learning algorithm.

Let us consider the division among machine learning algorithms, in case of other tasks as well as the classification or the regression. The combination of multiple machine learning algorithms for doing the task which is mentioned in this chapter is the cooperation among them. We may consider the case of using the neural networks for the classification, and using the decision tree for providing the evidence as the symbolic form. Using the Naive Bayes and the EM algorithm for implementing the semi-supervised learning is the instance of division of the multiple machine learning algorithms. In the division, one or some are involved in the classification or the regression, and the others are involved in supplementary tasks.

Let us present the direction of applying the machine learning algorithm to the big data mining. The big data is characterized as the 4Vs: big volume, high velocity, high veracity, and high variety [2]. A group of machine learning algorithms with its high diversity is used rather than a single algorithm. The training examples are discriminated by their quality; noisy ones are distinguished from sound ones, and different weights are assigned to the training examples, accordingly. We develop the machine learning algorithms which process different types of structured data.

References

1. T. Jo, Text classification using feature similarity based K nearest neighbor. AS Med. Sci. **3**(4), 13–21 (2019)
2. H. Mohanty, P. Bhuyan, D. Chenthati, *Big Data: A Primer* (Springer, Berlin, 2015)

Chapter 14

Semi-supervised Learning



14.1 Introduction

The semi-supervised learning is defined as the special type of supervised learning which uses both labeled and unlabeled training examples. The motivation of proposing the semi-supervised learning is that while labeled examples are expensive for taking them but unlabeled ones are cheap. The two sets of training examples are given in the semi-supervised learning: one is the set of originally labeled training examples and the other is the set of training examples which are labeled through the learning process. There are two schemes of implementing the semi-supervised learning: modification of unsupervised learning algorithms and combination of a supervised learning algorithm with an unsupervised one. This section is intended to describe the training set of the semi-supervised learning and its learning paradigm.

Let us mention the training examples which are used for the classification and the regression by the semi-supervised learning. In the supervised learning, only labeled examples are given as the training examples, whereas in the semi-supervised learning both labeled and unlabeled examples are given. The set of initially labeled training examples is called original training set, and the set of initially unlabeled ones is called additional training set. The two sets of training examples are used for training the learning algorithms in the semi-supervised learning. The case of using the labeled and the unlabeled example for clustering data items is called constraint clustering.

The semi-supervised learning uses labeled examples and unlabeled ones as its training examples for building the capacity of the classification or the regression. The semi-supervised learning may be viewed as the combination of a supervised learning algorithm and an unsupervised learning algorithm. The initial clusters may be constructed with the originally labeled examples, and unlabeled ones are clustered depending on their similarities with the initial clusters. Both kinds of training examples are learned by the unsupervised learning algorithm for classifying novice examples. As mentioned in Sect. 14.2, we modify the unsupervised learning

algorithms such as the Kohonen Networks and the k means algorithm into the semi-supervised versions.

Let us mention the techniques for reinforcing the supervised learning. The resampling for improving the classification performance or the regression performance is to control the distribution over the categories of training examples. Virtual training examples are generated artificially from existing training examples with various schemes, for improving the performance. The co-learning of two learning algorithms which uses the labeled examples and the unlabeled ones is the advanced scheme of the semi-supervised learning. The incremental learning is to learn training examples which are given as a data stream.

The goal of this chapter is to understand the semi-supervised learning as the hybrid paradigm of the supervised and the unsupervised. We will understand the Kohonen Networks which is initially designed as the unsupervised learning and its modification into the supervised and the semi-supervised one. We need to understand the specific semi-supervised learning algorithms which are built by combining a supervised learning algorithm and an unsupervised one to each other. We will also cover some advanced supervised which uses virtual examples and resamples the existing training examples. We will make the further discussions on what studied in this chapter for distinguishing the semi-supervised learning from the constraint clustering and modifying the unsupervised learning algorithms into their semi-supervised versions.

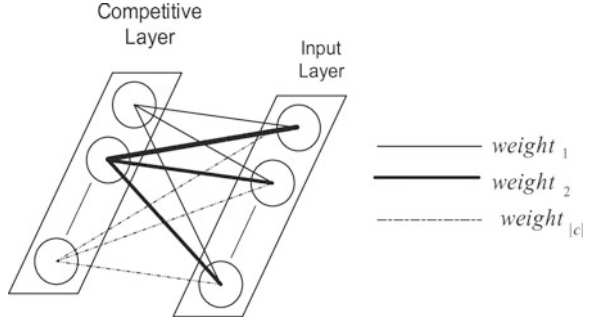
14.2 Kohonen Networks

This section is concerned with Kohonen Networks which was initially intended for clustering data items. In Sects. 14.2.1 and 14.2.2, we describe the supervised and unsupervised version of Kohonen Networks. In Sect. 14.2.3, we mention the Kohonen Networks as the semi-supervised learning algorithms. In Sect. 14.2.4, we compare the two clustering algorithms, the Kohonen Networks, and the k means algorithm with each other. This section is intended to present the three versions of Kohonen Networks and its comparisons with the k means algorithms.

14.2.1 *Initial Version*

This section is concerned with the initial version of Kohonen Networks as the unsupervised learning algorithm. There are two layers which are given in the architecture of the Kohonen Networks as the input layer and the competitive layer. The weight vectors are given as the cluster prototypes and are updated in the learning process. The SOM (Self-Organizing Map) is the version which is expanded from the initial version, for drawing the similarities on the input patterns. This section

Fig. 14.1 Architecture of initial version



is intended to describe the architecture and the learning process of the Kohonen Networks.

The architecture of the initial version of Kohonen Networks is illustrated in Fig. 14.1. There are two layers in the architecture: the input layer which receives the input vector and the competitive layer which indicates the clusters. The weight vector which is connected to each competitive node is given as a cluster prototype vector which corresponds to it. One among the competitive nodes whose weight vector is most similar as the input vector is determined as the winner. In designing the architecture of Kohonen Networks, the number of input nodes is the input vector dimension, and the number of competitive nodes is the number of clusters.

The unsupervised learning process in the Kohonen Networks is illustrated in Fig. 14.2. The input nodes and the competitive nodes are notated, respectively, by $x_1, x_2, \dots, x_d$ and $y_1, y_2, \dots, y_c$ and the weight vectors are notated as follows:

$$w_{11}, w_{12}, \dots, w_{1d}$$

$$w_{21}, w_{22}, \dots, w_{2d}$$

$$\dots$$

$$w_{c1}, w_{c2}, \dots, w_{cd}$$

where w_{ji} is the weight between the competitive node, y_j , and the input node, x_i . The weight vectors are initialized at random. The inner product of the weight vector, $[w_{k1} \ w_{k2} \ \dots \ w_{kd}]$ and the input vector, $[x_1 \ x_2 \ \dots \ x_d]$, is computed and the competitive node whose inner product with the input vector is maximal is selected as the winner, y_{win} . 1 and 0 are assigned, respectively, to the winner and the others, and the weight vector is updated by Eq. (14.1),

$$w_{ji} \leftarrow w_{ji} + \eta y_j (x_i - w_{ji}) \quad (14.1)$$

where η is the learning rate. The above process is iterated until the weight vectors converge.

```

learnItemListByKohonenNetworks(List itemList){
  Repeat the Following until Convergence of Weights{
    For each item in itemList{
      For each competitiveNode in competitiveNodeList
        compute its net input  $netinput_k = \sum_{i=1}^d x_i w_{ki}$ 
      Select the competitive node with its maximum net input as the winner
      Competitive Nodes: The Winner  $\leftarrow 1$  and the others  $\leftarrow 0$ 
      Update the weight using the following equation
         $w_{ji} = w_{ji} + \eta v_j (x_i - w_{ji})$ 
    }
  }
}

```

Fig. 14.2 Learning process of initial version

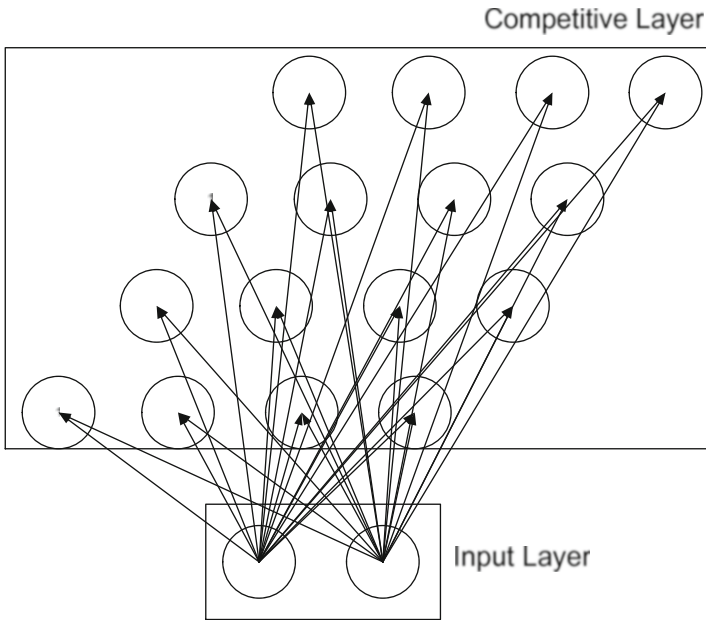


Fig. 14.3 Self-organizing map

The SOM is illustrated as a variant of the Kohonen Networks, in Fig. 14.3. Each competitive node corresponds to its own cluster in the Kohonen Networks, whereas it does to a grid of individual entries, in the SOM. In the Kohonen Networks, only single competitive node is selected as the winner, whereas the winner and its neighbors are selected as winners and the neighborhood function is defined as the degree of updating their weights in the SOM. The weight vector is updated, as expressed into Eq. (14.2),

$$w_{ji} \leftarrow w_{ji} + \eta \epsilon(y_{win}, d)(x_i - w_{ji}) \quad (14.2)$$

where $\epsilon(y_{win}, d)$ is the neighborhood function of the winner node, y_{win} , and the distance, d . After learning the training examples, the cluster boundaries over the competitive layer are defined in the SOM.

Let us make some remarks on the Kohonen Networks which were mentioned in this section. The Kohonen Networks were proposed for clustering data items by Kohonen, in 1970s [1]. They may be modified in the version for drawing a clustering map of individual items, called Self-Organizing Map. It may be modified into the supervised version for classifying items, called LVQ (Learning Vector Quantization). The Kohonen Networks and the LVQ are combined for implementing the semi-supervised version with each other.

14.2.2 Learning Vector Quantization

This section is concerned with the supervised version of the Kohonen Networks, called LVQ (Learning Vector Quantization). It is assumed that all of the training examples are labeled with one of the predefined categories. The number of competitive nodes is equal to the number of the predefined categories, and the weights which are connected to the node corresponding to the target category are updated. A novice example is classified by the inner product between the input vector and the weight vector. This section is intended to describe the LVQ which is the supervised neural networks with the competitive learning.

The design of the LVQ architecture is illustrated in Fig. 14.4. The number of nodes in the input layer is given as the dimension of the input vector, like the case in

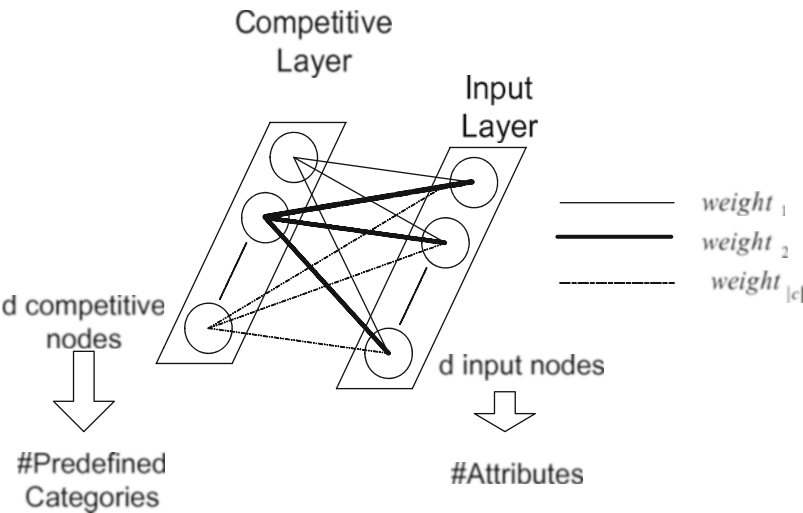


Fig. 14.4 Design of learning vector quantization

```
learnItemListByLearningVectorQuantization(List labeledItemList){
    Repeat the Following until Convergence of Weights{
        For each item in labeledItemList{
            Select the competitive node corresponding to desired Label as Winner
            Competitive Nodes: The Winner ← 1 and the others ← 0
            Update the weight using the following equation
                 $w_{ji} = w_{ji} + \eta y_j (x_i - w_{ji})$ 
        }
    }
}
```

Fig. 14.5 Learning process of learning vector quantization

Table 14.1 LVQ vs. Perceptron

	LVQ	Perceptron
Initialization	Random	Random
Learning type	Competitive learning	Gradient descent
Learning criteria	Similarity	Error
Update	Connected to winner	All weights

the supervised version. The number of nodes in the competitive layer is the number of the predefined categories. Two nodes are set in the competitive layer for the binary classification, and the nodes as many as categories are set for the multiple classification. When applying the LVQ to the regression, the number of competitive nodes is given as the number of output variables.

The learning process of the LVQ is illustrated as the pseudo code in Fig. 14.5. It is assumed that all of the training examples are labeled with one of the predefined categories. For each training example, the competitive node which corresponds to the target label is selected as the winner and the weights which are connected with it are updated. The winner selection and the weight update are iterated until the convergence where the weight vectors change very little. The winner in the initial version is the competitive node with its maximal inner product of the weight vector and the input vector, whereas in the LVQ, the winner is the competitive one which corresponds to the target label.

In Table 14.1, we present the comparisons of the LVQ and the Perceptron with respect to their learning process. In both of them, the weight vectors are initialized at random. In the LVQ, the competitive learning where the winner is selected in the competitive layer is performed, whereas, in the Perceptron, the gradient descent for decreasing the misclassification rate of the training examples is performed. What the LVQ pursues is more similarity with weights which connected from the competitive node which corresponds to the target label, whereas what the Perceptron pursues is the less misclassification rate. In the LVQ, the weights which are connected to the winner are updated, whereas all weights are updated in the Perceptron.

Let us make some remarks on the LVQ which is mentioned as a classification algorithm in this section. The LVQ is a version of Kohonen Networks which is mentioned in Sect. 14.2.1, which is modified into the supervised version. In Sect. 10.2, the KNN algorithm is changed into the unsupervised version. Note that it is possible to transit the machine learning algorithm between the unsupervised learning and

Fig. 14.6 Training examples for semi-supervised version



the supervised learning. The constraint clustering or the semi-supervised learning is applicable to the situation where both unlabeled examples and labeled ones are given as the training examples.

14.2.3 *Semi-supervised Version*

This is concerned with the semi-supervised version of Kohonen Networks. We studied the unsupervised version and the supervised version, is called LVQ. The two types of learning algorithms may be combined for using the labeled examples and the unlabeled examples in the semi-supervised version. The semi-supervised learning was already mentioned in Sect. 10.2.3, which is concerned with the semi-supervised version of KNN algorithm. This section is intended to describe the semi-supervised learning process with the Kohonen Networks.

Two sets of training examples which are used in the semi-supervised learning are presented in Fig. 14.6. It is intended to use labeled examples which are obtained easily for training the machine learning algorithms, as well as unlabeled ones. The training set is divided into the two sets: the labeled set and the unlabeled set. The labeled one is used directly for training the machine learning algorithm. The issue of the semi-supervised learning is how to assign labels to the unlabeled examples.

The learning process of the semi-supervised version of the Kohonen Networks is illustrated in Fig. 14.7. It is assumed that both the labeled examples and the unlabeled ones are given as the training examples. To each labeled training example, the competitive node which corresponds to its target label is selected as the winner, and to each unlabeled one, one whose weight vector has the highest inner product is selected. The weights which are connected to the winner are updated like the previous versions of the Kohonen Networks. The semi-supervised version is viewed as the mixture of the both versions of Kohonen Networks which are covered, respectively, in Sects. 14.2.1 and 14.2.2.

The co-learning is illustrated as another semi-supervised learning type in Fig. 14.8. The labeled examples and the unlabeled examples are given as the training examples, and the unlabeled example set is divided into the two subsets: unlabeled set 1 and unlabeled set 2. The two machine learning algorithms learn the labeled examples, the examples in unlabeled set 1 are classified by the left machine

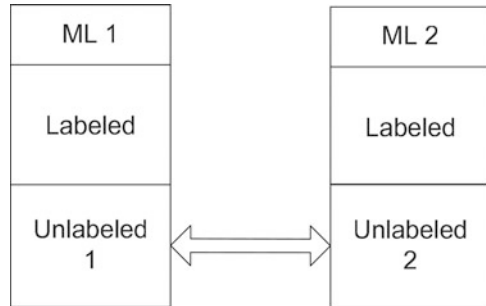

```

learnItemListByKohonenNetworks(List itemList){
  Repeat the Following until Convergence of Weights{
    For each item in itemList{
      if(isLabel(item)){
        Select the competitive node corresponding to desired Label as Winner
      }
      else{
        For each competitiveNode in competitiveNodeList
          compute its net input  $netinput_k = \sum_{j=1}^d x_j w_{kj}$ 
        Select the competitive node with its maximum net input as the winner
      }
      Competitive Nodes: The Winner  $\leftarrow 1$  and the others  $\leftarrow 0$ 
      Update the weight using the following equation
       $w_{ji} = w_{ji} + \eta y_j (x_i - w_{ji})$ 
    }
  }
}

```

Fig. 14.7 Learning process of semi-supervised version

Fig. 14.8 Co-learning version



learning algorithm, and ones in unlabeled set 2 are classified by the right machine learning algorithm. The two sets of unlabeled examples are swapped between the two machine learning algorithms; unlabeled set 2 is added to the training set of the left machine learning algorithm, and unlabeled set 1 is added to that of the right machine learning algorithm. The co-learning is the semi-supervised learning by the interaction of the two machine learning algorithms, as shown in Fig. 14.8.

Let us make some remarks on the semi-supervised version of the Kohonen Networks, which is mentioned in this Section. We already studied the semi-supervised version of the KNN in Sect. 10.2.3. The KNN is evolved from the supervised version into the semi-supervised version by means of its unsupervised version. Here, the Kohonen Networks are evolved from the unsupervised version into the semi-supervised version by means of its supervised version. The semi-supervised version of the KNN and the Kohonen Networks are applied to the constraint clustering, as well as the classification.

Table 14.2 Kohonen vs. K means

	Kohonen	K means
Initialization	Random weights	Random vectors
Cluster prototypes	Weight vectors	Mean vectors
Learning criteria	Similarity	Similarity
Update	Weights connected to winner	Averaging

14.2.4 Kohonen Networks vs. K Means Algorithm

This section is concerned with the comparisons of the two unsupervised learning algorithms, as shown in Table 14.2. We studied the k means algorithm as a typical clustering algorithm in Chap. 10, and did the three versions of Kohonen Networks in the previous section. The clustering prototypes are updated as the clustering process in both algorithms. The clustering prototypes in the k means algorithm are initialized by mean vectors, whereas they in the Kohonen Networks are initialized at random. This section is intended to explore the differences between the Kohonen Networks and the k means algorithm through Table 14.2.

The difference between the k means algorithm and the Kohonen Networks with respect to the initialization phase is illustrated in Table 14.1. The weights between the competitive layer and the input layer are initialized at random in the Kohonen Networks. The weights which are connected to each competitive node become the initial cluster representatives. In the k means algorithm, it is initialized it by selecting ones as many as clusters from data items at random. The selected data items become the initial cluster representatives in the k means algorithm.

Let us mention the difference between two clustering algorithms with respect to the cluster prototype vectors. The clustering prototype vector means the representative vector which characters its own cluster. In the k means algorithm, the clustering prototype is the cluster mean vector, whereas in the Kohonen Networks, it is the weight vector. In each cluster, its prototype vector is usually not one among cluster members in both algorithms. Cluster prototypes are updated gradually, but the updating processes are different from each other.

Let us consider the difference between the two unsupervised machine learning algorithms with respect to the process of updating cluster prototypes. In the k means algorithm, the mean vectors are given as prototypes, whereas in the Kohonen Networks, the weight vectors are given prototypes, as mentioned above. In the Kohonen Networks, the competitive node whose weight vector is most similar as the input vector is selected as the winner, and its weight vector is updated to be closer to the input vector. In the k means algorithm, the mean vectors are recalculated after arranging items into clusters. In the Kohonen Networks, cluster prototypes are updated interactively, whereas in the k means algorithm, they are one at batch.

Cluster prototype vectors converge finally in both the Kohonen Networks and the k means algorithm. The convergence to almost fixed ones happens in the Kohonen Networks by continual update of weight vectors. It happens in the k

means algorithm by iterating the update of mean vectors and the arrangement of items. The difference between previous weight vectors and current ones becomes the termination condition in executing the Kohonen Networks, whereas the difference between the previous mean vectors and current ones becomes the termination condition of executing the k means algorithm. The initial decision of the number of clusters is required for clustering data items in both clustering algorithms.

14.3 Combined Learning Algorithms

This section is concerned with the combined learning algorithms for implementing the semi-supervised learning. In Sect. 14.3.1, we mention the three combinations schemes of unsupervised learning algorithm. In Sect. 14.3.2, we mention the combinations of simple clustering algorithms which were covered in Sect. 9.2. In Sects. 14.3.3 and 14.3.4, we describe the two instances of combining the supervised learning and the unsupervised learning with each other for implementing the semi-supervised learning. This section covers the combination of unsupervised learning algorithms and one of the supervised and the unsupervised for implementing the semi-supervised learning.

14.3.1 Combination Paradigms

This section is concerned with the three combination schemes, voting, expert gate, and cascading, which were mentioned in Sect. 13.1, which are applicable to the unsupervised learning algorithms. In Chap. 13, we studied the three combination schemes of the unsupervised learning algorithms based on their output values. Because data items are initially unlabeled in the unsupervised learning, the relationship between two items becomes the basis for deciding the final clusters in combining unsupervised learning algorithms. The integration of multiple clustering results is the role of the combination of unsupervised learning algorithms. This section is intended to apply the three schemes which are mentioned in Sect. 13.1 to unsupervised learning algorithms.

The voting of the unsupervised learning algorithms is illustrated in Fig. 14.9. M unsupervised learning algorithms build their own clusters from the same data items as the committee members. All possible pairs are generated from the data items, and the coordinator decides whether each pair belongs to a same cluster or different clusters. By voting clustering results of the committee members, it decides the status of each item pair, same cluster or different clusters. It takes the quadratic complexity of the number of items for voting the clustering algorithms.

Another scheme of combining unsupervised learning algorithms, called expert gate, is illustrated in Fig. 14.10. The intra-clustering index in the case of two examples in a same cluster or the inter-cluster index in the case of them in different

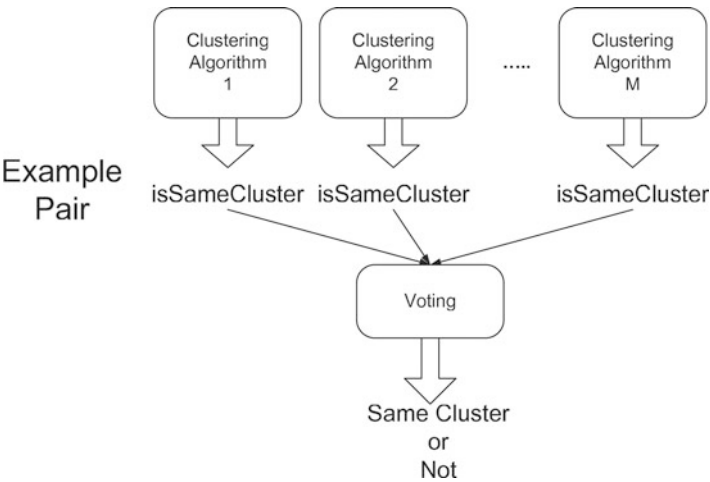


Fig. 14.9 Voting in supervised learning

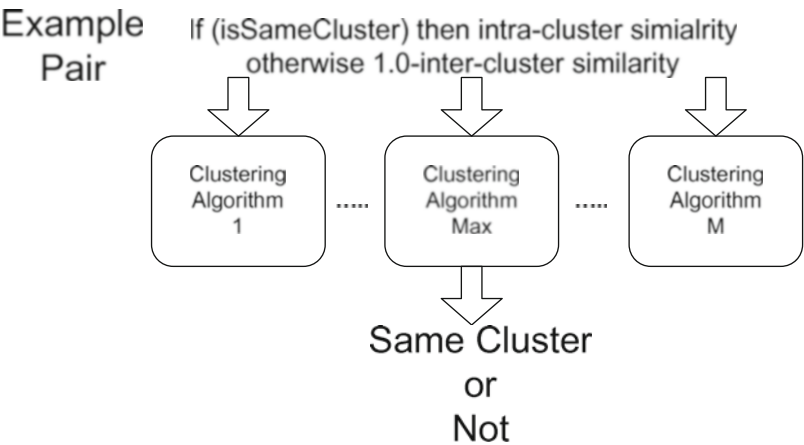


Fig. 14.10 Expert gate in supervised learning

clusters is computed, and the discrimination is defined by subtracting the inter-cluster similarity from 1.0, under the assumption that both values are given as normalized values between zero and one. An example pair is given as the input, and the clustering algorithm with the maximum intra-cluster similarity or the maximum distribution is selected as the expert. The expert decides whether the examples in the pair should be arranged into a same cluster or different clusters. In the voting, all clustering algorithms are involved in the final decision, whereas in the expert gate, a single clustering algorithm is involved.

The cascading of the unsupervised learning algorithms is illustrated in Fig. 14.11. In the cascading, the coordinator decides whether the input is transferred to the next

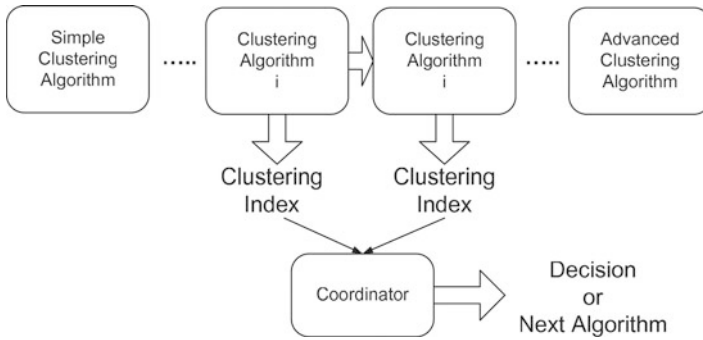


Fig. 14.11 Cascading in supervised learning

algorithm or takes the answer from the current one. The process of computing the clustering index from the unlabeled examples was described in Sect. 12.3.1, so if the clustering index is poor, it is transferred to the next algorithm. If the clustering index by the current algorithm is less than that by the previous one, the results by the previous one are adopted; this scheme looks similar as the hill-climbing method. The simple and fast clustering algorithm is arranged in the early state and the reliable and complicated one is done in the later stage.

Let us make some remarks on the cascading of the unsupervised learning algorithms. In the cascading, the simple version is arranged in the early state and the advanced version is arranged in the later stage. Each clustering algorithm decides whether two items in each pair belong to same cluster or different clusters. If both items belong to same cluster, the intra-cluster index is computed as the clustering index, and otherwise, the discrimination among clusters including them is computed as the clustering index. In this combination, the clustering algorithms are connected with each other serially.

14.3.2 Simple Learning Algorithms

This section is concerned with the combination of simple clustering algorithms. The three clustering algorithms, the AHC algorithm, the divisive clustering algorithm, and the online linear clustering algorithm, were mentioned as simple clustering algorithms. We mention the three representative specific combinations: the multiple AHC algorithms, the AHC + divisive clustering algorithm, and the AHC + online linear clustering algorithm. It is possible to combine the clustering algorithm with the classification algorithm, for implementing the semi-supervised learning. This section is intended to explain the three combined clustering algorithms with respect to the process of clustering data items.

Using more than one AHC algorithm for clustering data items is illustrated in Fig. 14.12. The several AHC algorithms are discriminated by the similarity metric,

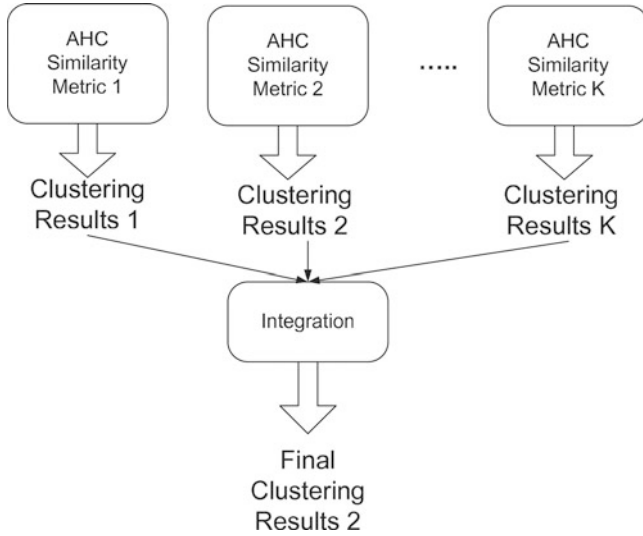


Fig. 14.12 Multiple AHC algorithms

the termination condition, and the clustering process. By executing each AHC algorithm independently, its own clustering results are generated. By combining the clustering algorithms with one among ones which are mentioned in Sect. 14.3.1, the clustering results are integrated into each other. Much higher complexity which is the quadratic complexity to the number of data items is taken for combining the clustering results.

The combination of the AHC algorithm with the divisive clustering algorithm is illustrated in Fig. 14.13. We mentioned the AHC algorithm as the bottom-up clustering algorithm and the divisive clustering algorithm as the top-down clustering algorithm, respectively, in Sects. 9.2 and 9.3. The AHC algorithm starts with singletons as many as items, and the divisive clustering algorithm starts with a single group of all items. The desired number of clusters is given as the parameter, when both algorithms meet in the desired number of clusters by their both directions, and the clustering results of both algorithms are integrated with each other. Depending on an application area, multiple clustering results are accommodated; this case is called multiple viewed clustering [2].

The combination of the AHC algorithm and the online linear clustering algorithm for clustering data items is illustrated in Fig. 14.14. The online linear clustering is characterized as the high speed but the poor performance. The N items are clustered into M subgroups by the online linear clustering algorithm, and the M clusters are converted into C clusters by the AHC algorithm, as shown in Fig. 14.14. The similarity threshold is set closely to 1.0, under the assumption of the similarity as a normalized value in the online linear clustering algorithm. The hierarchical clustering may be implemented using the two algorithms.

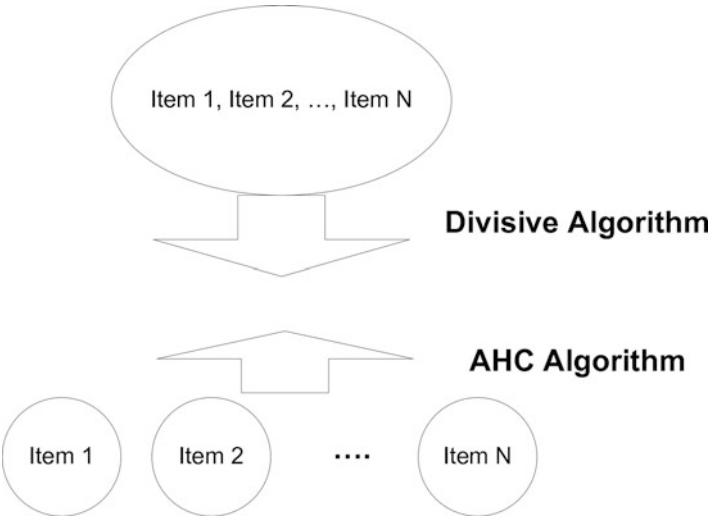


Fig. 14.13 AHC + Divisive algorithm

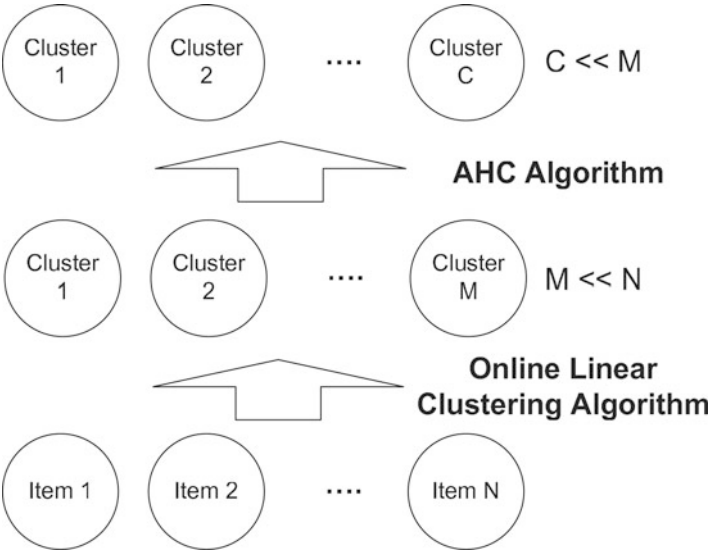


Fig. 14.14 AHC algorithm + online linear clustering

Let us review installing the parameter tuning into clustering algorithms which was covered in Sect. 12.3. It is expected to improve the clustering performance, by observing the clustering quality while clustering data items. Computing a clustering index from the current results from clustering data items depends on the similarities among data items. Automatically updating external parameters of the clustering

algorithm depends on the clustering index. Installing the parameter tuning into the clustering algorithm is intended to improve the clustering performance by sacrificing the clustering speed.

14.3.3 *K Means Algorithm + KNN Algorithm*

This section is concerned with the semi-supervised learning which is implemented by combining the supervised learning algorithm and the unsupervised one with each other. The semi-supervised learning algorithm is derived by modifying an unsupervised learning algorithm into its supervised version in Sects. 10.2 and 14.2. In this section, the semi-supervised learning algorithm is proposed by combining the two kinds of learning algorithm. As an instance, the KNN algorithm and the k means algorithm are combined with each other for implementing the semi-supervised learning. This section is intended to describe semi-supervised learning by combining the two learning algorithms.

Let us mention the training examples for training the semi-supervised learning by the combination of the k means algorithm and the KNN algorithm. The training set for the semi-supervised learning consists of the labeled examples and the unlabeled examples. The training example set, Tr , is divided into the two sets; Tr_L is the set of originally labeled examples and Tr_U is the set of originally unlabeled examples. The unlabeled examples are utilized for improving the classification performance in the previous literatures [3]. The training examples in the set, Tr_U , are labeled subsequently.

The data items are clustered by an unsupervised learning algorithm for labeling the items in the set, Tr_U . By the items, in the set, Tr_L , the clusters are defined and the initial mean vectors are decided. The items in the set, Tr_U , are arranged into clusters which are defined by the items in the set, Tr_L . Updating the mean vectors and arranging the items in the set, Tr_U , are iterated. The role of the k means algorithm is to construct all of the training examples in the both sets, Tr_L and Tr_U , as the labeled ones.

A novice item is classified by the KNN algorithm, after clustering the data items. The data items in the set, Tr_U , are clustered, following the set, Tr_L , and individual items in the set, Tr_U , are labeled through the clustering. A novice item is given and its similarities with the examples in the both sets, Tr_U and Tr_L , are computed. The most similar examples are selected as the nearest neighbors, with regardless of the both sets, and the label of the novice example is decided by voting the labels of the nearest neighbors. The process of classifying the novice item is same to the process of doing by the KNN algorithm which was mentioned in Sect. 5.3, except using the originally unlabeled examples.

Let us make some remarks on the implementation of the semi-supervised learning by combining the k means algorithm with the KNN algorithm. Both the set of unlabeled examples and the set of the labeled ones are given as the training set. The labeled examples are arranged by their labels, and the initial mean vectors

are computed from the labeled examples. The unlabeled ones are clustered by the k means algorithm, iteratively. Both kinds of the training examples are used for executing the KNN algorithm for classifying a novice item.

14.3.4 EM Algorithm + Naive Bayes

This section is concerned with another combination of the two machine learning algorithms for implementing the semi-supervised learning. We already studied the combination of the k means algorithm and the KNN algorithm with each other for doing so, in Sect. 13.3.3. The combination of the Naive Bayes and the EM algorithm with each other is another scheme of implementing the semi-supervised learning. This combination was applied of the text categorization, using the labeled texts and unlabeled ones as the training examples, in previous work [3]. This section is intended to describe the combination of the two machine learning algorithms.

Let us review the E-step under the assumption of the probability distributions over the clusters as the Gaussian distributions. The training set consists of the labeled examples and the unlabeled ones, and the probability distributions are initialized by the labeled examples. For each unlabeled item, its cluster membership values are computed based on the Gaussian distributions. The cluster-item matrix which consists of the cluster membership values as the elements is constructed by the E-step. Each labeled example has only one membership value to the target category and zero membership values to the others, whereas each unlabeled example has continuous membership values from zero to one to the all clusters.

The M-step is the process of updating the parameters of the probability distribution for each cluster. The item-cluster matrix whose element is a cluster membership value to an item is constructed in the E-step. The mean vectors and the covariance matrices are computed based on the item-cluster matrix. The final version of the item-cluster matrix is generated by iterating the E-step and the M-step. The matrix is used for computing the likelihoods of individual values in using the Naive Bayes.

The final version of the item-cluster matrix which consists of the cluster membership value is constructed by the above process from both the labeled examples and the unlabeled ones. The given problem is assumed as a binary classification where each item is classified into the positive class or the negative class, the cluster membership values, $\mu_+(\mathbf{x})$ and $\mu_-(\mathbf{x})$, are assigned to the example, $\mathbf{x}$, and the training examples which include the originally unlabeled ones are given as a set, $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. In the learning process, the likelihoods of individual values are computed by Eqs. (14.3) and (14.4),

$$P(a_i = x_i | +) = \frac{\sum_{\mathbf{x}_k \in (a_i = x_i)} \mu_+(\mathbf{x}_k)}{\sum_{k=1}^n \mu_+(\mathbf{x}_k)} \quad (14.3)$$

$$P(a_i = x_i | -) = \frac{\sum_{\mathbf{x}_k \in (a_i = x_i)} \mu_{-}(\mathbf{x}_k)}{\sum_{k=1}^n \mu_{-}(\mathbf{x}_k)} \quad (14.4)$$

The likelihoods of the novice example to the both categories are computed by the product of ones of individual values as shown in Eqs. (14.5) and (14.6),

$$P(\mathbf{x} | +) = \prod_{i=1}^d P(a_i = x_i | +) \quad (14.5)$$

$$P(\mathbf{x} | -) = \prod_{i=1}^d P(a_i = x_i | -) \quad (14.6)$$

and it is classified into the category with the maximal likelihood. The originally labeled examples have the crisp cluster membership values as zero or one, whereas the originally unlabeled ones have the continuous cluster membership values between zero and one.

Let us make some remarks on the combination of the EM algorithm and the Naive Bayes for implementing the semi-supervised learning. The probability distribution for each cluster is defined as a Gaussian distribution which is characterized by its mean vector and its covariance matrix. The given task is assumed to be a binary classification; the EM algorithm is used for the binary clustering. The likelihoods of the individual attribute values are computed based on the summation of the cluster membership values, rather than by counting the number of corresponding examples, in using the Naive Bayes. The role of the EM algorithm is to perform the constraint clustering of the labeled examples and the unlabeled ones.

14.4 Advanced Supervised Learning

This section is concerned with the advanced supervised learning for improving the classification performance or the regression one. In Sects. 14.4.1 and 14.4.2, we mention the manipulations on the training examples, such as the virtual example generation and the resampling, respectively. In Sect. 14.4.3, we describe the co-learning which is executed in a pair of machine learning algorithms. In Sect. 14.4.4, we explain the incremental learning which trains a machine learning algorithm or machine learning algorithms by adding training examples, gradually. This section is intended to explore some advanced learning for reinforcing the supervised learning algorithms.

14.4.1 Resampling

The resampling is defined as the process of adjusting the training set by adding more training examples or deleting some of them. Until now, the training set has been assumed to be a fixed set, but it is not guaranteed that the training set is complete for training machine learning algorithms. In this case, we need to make the training set which is suitable for doing so, by adjusting the current training set. The semi-supervised learning which is studied in the previous chapters and the current chapter becomes a resampling instance. This section is intended to study the schemes of adjusting the training set for improving the classification and the regression performance.

The oversampling and the undersampling for balancing the distribution over the categories are illustrated in Fig. 14.15. It is assumed that the given task is a binary classification and the distribution over the two categories is unbalanced. The oversampling means to balance the distribution by adding more training examples, and the undersampling means to do it by deleting some training examples. The simplest scheme is addition of more artificial examples by manipulating existing training examples as the oversampling and the deletion of some at random in the undersampling. The oversampling scheme and the undersampling scheme have been developed into advanced ones [4, 5].

Let us make some schemes of resampling the training examples. In 2004, the oversampling and the undersampling were proposed in using a combination of multiple decision trees for the data classification tasks, including the text classification, by Estabrooks et al. [4]. In 2004, Jo and Japkowicz proposed the resampling method which considers the small disjuncts which are caused by the class imbalance [5]. The resampling was applied for implementing the DDO (Dynamic Document Organization) system by Jo in 2006 [6]. It was discovered that the reliability and the

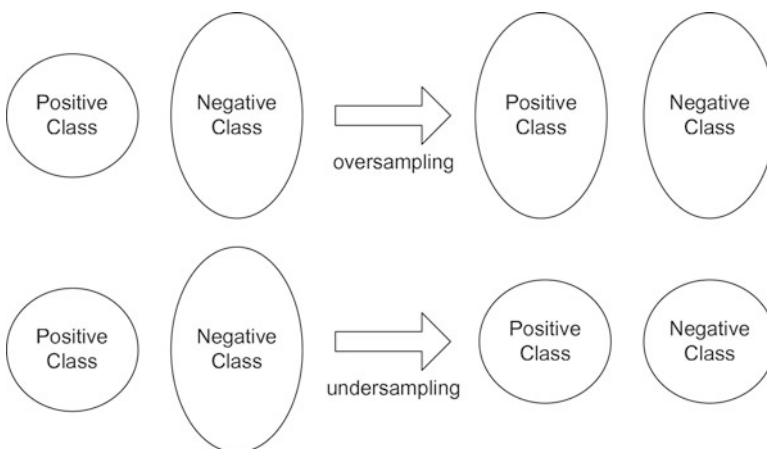


Fig. 14.15 Oversampling vs. undersampling

integrity of the original training examples are required for using the resampling by Jo [6].

The classification performance is improved as the effect of resampling the training examples. The bias toward a particular category is removed by balancing the distribution over the predefined categories. It is asserted that the classification performance is improved by removing small disjuncts by Jo and Japkowicz, in 2004 [5]. However, it failed to improve the classification performance by the resampling in implementing the DDO system [6]. Note that the resampling is effective on the training examples which are collected manually with their high reliability.

The resampling is expanded into the generation of the artificial training examples for the classification performance or the regression performance. The resampling is intended to balance the distribution over the categories by adding more training examples or deleting some. A sufficient number of training examples is required for training the machine learning algorithm, robustly; it was very important issue in 1990s. The results were successful in applying the MLP (Multiple Layer Perceptron) to the time series predictions by generating artificial measures in 2010 and 2012 [7, 8]. The semi-supervised learning which is mentioned in Sect. 14.3 is proposed for utilizing the unlabeled examples which are very cheap to obtain.

14.4.2 Virtual Training Example

The virtual training examples are defined as the training examples which are not given originally, but are generated artificially from existing ones. The resampling which was studied in Sect. 14.4.1 is intended to balance the distribution over the categories, whereas the generation of the virtual examples is done for solving the insufficient number of training examples. The virtual training examples are generated by generating the training examples based on the Gaussian distribution at random or manipulating the existing ones. The scheme should be distinguished from the semi-supervised learning where the originally unlabeled training examples are used, rather than the virtual examples. This section is intended to study the scheme of generating the artificial data for improving the generalization performance.

Because the semi-supervised learning and the generation of the virtual examples may be similar as each other with respect to their ideas, the semi-supervised learning is mentioned before discussing the virtual training examples. The labeled examples and the unlabeled ones are used for building the capacity of the classification or the regression in the semi-supervised learning. The unlabeled examples are viewed as the virtual examples, and the labeled ones are viewed as the actual ones. The motivation of the semi-supervised learning is to gather unlabeled examples very easily. The unlabeled examples belong to the actual examples, in the real world.

The process of using the virtual examples which are generated from the existing training examples as the labeled ones is illustrated in Fig. 14.16. The labeled training examples are initially given, and the input domain is defined. The unlabeled examples are generated from the input domain at random; the artificial examples are

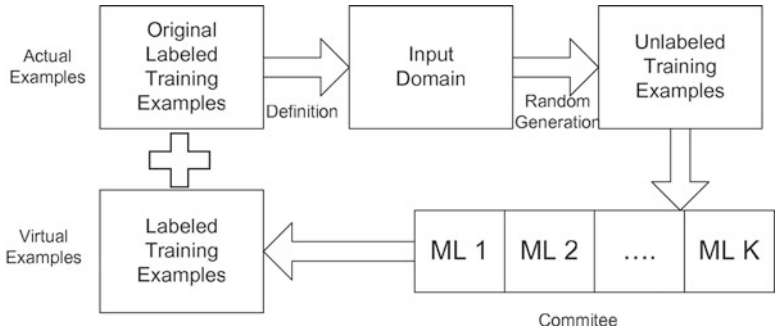


Fig. 14.16 Generation of virtual training examples

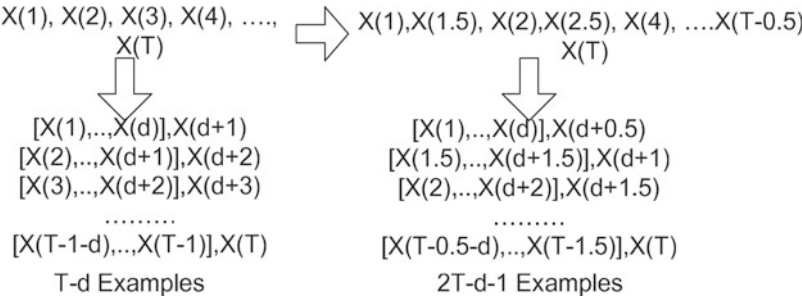


Fig. 14.17 Virtual terms in time series prediction

constructed by manipulating slightly the existing training examples. The generated training examples are classified by the machine learning algorithms which are given as the committee members, and they are added to the training examples. The results in both the classification and the regression using the artificially generated examples are successful in previous works [7–9].

The schemes of estimating mid-terms in the time series prediction were proposed by Jo in 2010s [7, 8]. The process of estimating mid-terms which are called virtual terms is illustrated in Fig. 14.17. The number of the training examples is increased almost two times as shown in Fig. 14.17. The results from using the MLP for the both kinds of time series predictions were very successful [7, 8]. The schemes are characterized as the mixture of the artificial data and the original data in each training example.

Let us make some remarks on the generation of the artificial data for improving the generalization performance. The overfitting is the phenomena where the classification or the regression works very well on the training examples, but poorly on the novice ones and happens in learning a smaller number of training examples. In 2000s, more training examples are generated artificially by manipulating the original training examples by Sassano and Jo [7–9]. The fact that the unlabeled examples are gathered easily is the motivation for proposing the semi-supervised

learning. In the comparison of the resampling, the virtual sampling, and the semi-supervised learning, the first is intended to balance the distribution over the categories, the second is intended to improve the generalization performance using the artificial training examples, and the semi-supervised learning is intended to improve the generalization performance using the unlabeled examples.

14.4.3 Co-Learning

The co-learning is defined as the type of the supervised learning by exchanging unlabeled examples between two machine learning algorithms. The semi-supervised learning is intended to utilize the unlabeled examples as the training examples for improving the generalization performance. The unlabeled example set is divided into two sets and they are allocated to the two learning algorithms. The originally unlabeled examples which are labeled by the opposite one are added to the originally labeled ones. This section is intended to describe the co-learning as the special type of the semi-supervised learning.

The partition of the training set for the co-learning is illustrated in Fig. 14.18. The labeled training examples and the unlabeled training examples are prepared for the type of semi-supervised learning, and the unlabeled set is partitioned into two subsets, additionally. The both learning algorithms are trained with the originally labeled training examples. The first machine learning algorithm, ML1, classifies the first subset of the unlabeled examples, and the second, ML2, classifies the second subset. The examples which are classified subsequently are added to the set of the originally labeled examples.

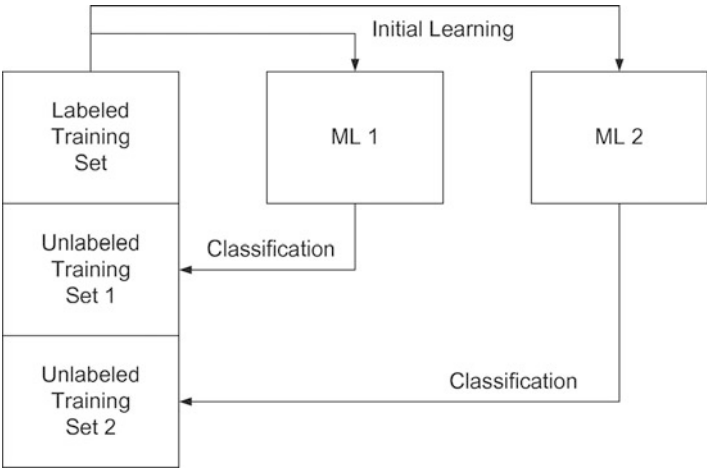


Fig. 14.18 Training example for co-learning

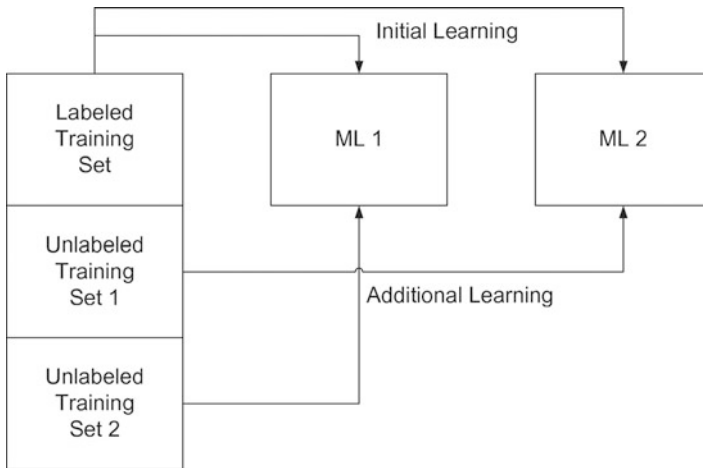


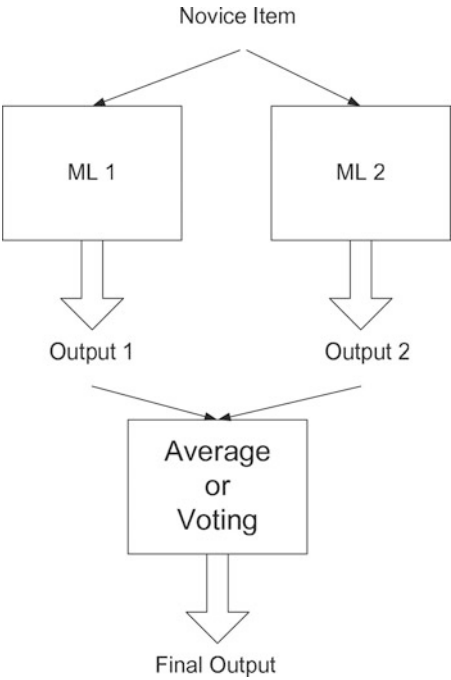
Fig. 14.19 Dual learning process

Using the training examples for the two machine learning algorithms in the co-learning is illustrated in Fig. 14.19. The both learning algorithms are initially trained with the originally labeled examples. The unlabeled examples in the first set are classified by the ML1, and ones in the second set are classified by the ML2. By swapping the two sets with each other, the first set is allocated for training the ML2, and the second set is done for training the ML1. The addition of examples which are labeled by the opposite learning algorithm to the originally labeled examples is the reason of calling this semi-supervised learning type co-learning.

The process of classifying an item by the two machine learning algorithms with the co-learning is presented in Fig. 14.20. The novice item is given as the classification target for the two machine learning algorithms. It is classified by the two machine learning algorithms; the two outputs are generated from them. The final output is decided by voting or averaging the two outputs. Averaging over the two outputs, rather than voting them is to make the final decision clearly.

Let us make some remarks on the co-learning which is the special type of the semi-supervised learning. The two machine learning algorithms are involved in the co-learning but note that more than two may be involved in the co-learning in some cases. The co-learning may be viewed as the hybrid style of the ensemble learning and the semi-supervised learning. The final output is decided by voting or averaging the outputs of the two machine learning algorithms in the view of the ensemble learning. The unlabeled examples are used in the additional ones for training the machine learning algorithms in the view of the semi-supervised learning.

Fig. 14.20 Classification process



14.4.4 Incremental Learning

This section is concerned with the incremental learning as the recent trend of the machine learning. The incremental learning is the learning paradigm which deals with the situation where the training set grows gradually. It is defined as the process of learning only additional training examples, keeping learned by the existing ones. The incremental learning is intended to avoid retraining the existing training examples, whenever more training examples are added. This section is intended to describe the incremental learning in the conceptual view.

The training set which is incremented gradually as time goes is illustrated in Fig. 14.21. Until now, we have assumed that the training is initially fixed. In the real world, the training set grows gradually by adding more training examples. It is assumed that the machine learning algorithm is trained by the initial training set and it is continually trained by the additional training set. The learning process has been developed in almost machine learning algorithms until now, under the assumption of only initial fixed set of the training examples.

The increment learning in the binary classification in the two dimensional space is visualized in Fig. 14.22. The training examples which are labeled with the positive class or the negative class are originally given as the left part of Fig. 14.22. More training examples are added into the distribution in the two dimensional space as shown in the right part of Fig. 14.22. The hyperplane from the left one to the right

Fig. 14.21 Increment of training examples

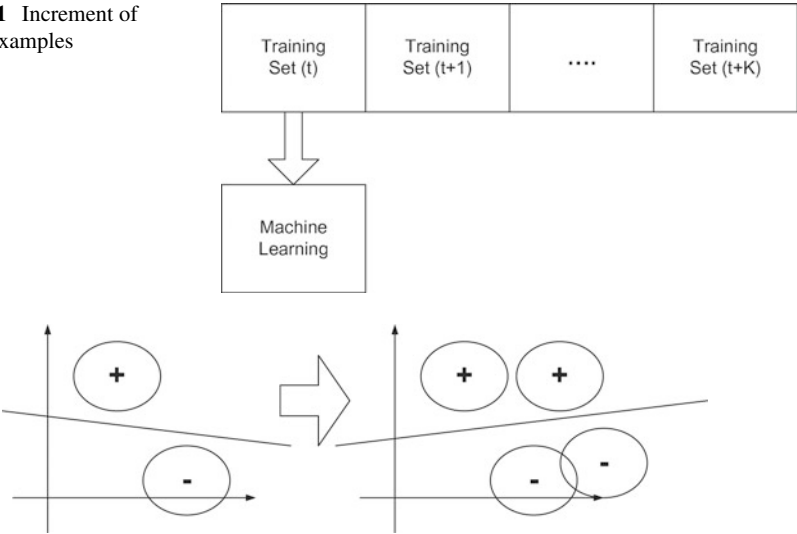


Fig. 14.22 Incremental learning process

one is updated for classify the added examples as well as the original ones without any error. The optimal hyperplane is searched by minimizing gradually the error to the added training examples, as the incremental learning.

The addition of the unlabeled examples and the incremental learning to them is illustrated in Fig. 14.23. The set of the labeled examples is initially given and the labeled ones are learned by the machine learning algorithm. After the initial training, the unlabeled examples are gathered and used as additional training examples. We need to cluster unlabeled ones, based on the labeled ones. It is not easy to implement the increment learning, covering the unlabeled examples.

Let us make some remarks on the incremental learning as an advanced learning scheme. All of the training examples are given as a stream, rather than at a batch, in the real world. We expect the gradual learning from the training examples which are given as a stream. Whenever additional training examples are accommodated, retraining the machine learning algorithms with the previous training examples should be omitted. The previous training examples should be referred, in order to prevent the misclassification which is leaded by learning the additional ones.

14.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We explored the three versions of Kohonen Networks: the unsupervised version, the supervised version, and the semi-supervised version. We studied the ensemble unsupervised learning

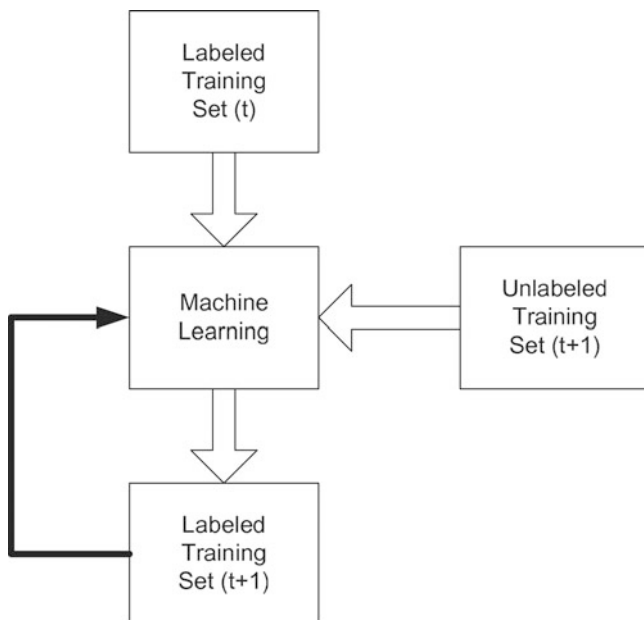


Fig. 14.23 Increment of unlabeled examples

and the semi-supervised learning by combining the supervised and the unsupervised. We studied the advanced supervised learning such as resampling, virtual example generations, co-learning, and the incremental learning. In this chapter, we studied the three versions of Kohonen Networks, the combination of learning algorithms for implementing the unsupervised and the semi-supervised, and the advanced supervised learning techniques.

Let us mention the constraint clustering to which we apply the k means algorithm in Sect. 10.4.4. We used labeled examples and unlabeled examples like the semi-supervised learning. The clusters are initialized using the labeled examples, and updated using the unlabeled ones, continually. We need to consider the possibility of existence of more clusters beyond the predefined labels. A very small number of labeled examples is really given in the constraint clustering, compared with the number of unlabeled ones.

Both the labeled examples and the unlabeled ones are used for implementing the semi-supervised learning and the constraint clustering. The unlabeled examples are obtained more easily than the labeled ones in reality. The semi-supervised learning was proposed using the unlabeled examples rather than generating virtual examples. The constraint clustering is included in the process of semi-supervised learning. We consider control or manipulate the predefined categories by adding more categories or deleting some.

Let us mention the active learning as a kind of the advanced supervised learning. The traditional learning which is called passive learning is to train the machine

learning algorithms with the training examples which are initially given. The active learning is the process of training them with the training examples which are selected actively as essential ones. Its contribution is to improve the classification or regression performance in using any machine learning algorithm to any task. In active learning, query should be generated automatically for retrieving relevant training examples.

Let us remain the discriminations of the training examples by their necessity from considering the active learning. The active learning is mentioned as learning the training examples which are relevant to the given query, and they are discriminated by their relevancy to the query. We need to discriminate the training examples by their qualities as well as their relevancies. The training examples should be measured by their quality, and weights are assigned to them, proportionally to the qualities. We need to develop the metric for evaluating the quality.

References

1. T. Kohonen, Correlation matrix memories. *IEEE Trans. Comput.* **21**, 353–359 (1972)
2. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, Berlin, 2018)
3. K. Nigam, A.K. McCallum, S. Thrun, T.M. Mitchell, Text classification from labeled and unlabeled documents using EM. *Mach. Learn.* **39**(2–3), 1–34 (2000)
4. A. Estabrooks, T. Jo, N. Japkowicz, A multiple resampling method for learning from imbalanced data sets. *Comput. Intell.* **28**(1), 18–36 (2004)
5. T. Jo, N. Japkowicz, Class imbalances versus small disjuncts. *ACM SIGKDD Explorat.* **6**(1), 40–49 (2004)
6. T. Jo, The implementation of dynamic document organization using text categorization and text clustering, PhD Dissertation of University of Ottawa, 2006
7. T. Jo, The effect of mid-term estimation on back propagation for time series prediction. *Neural Comput. Appl.* **19**(8), 1237–1250 (2010)
8. T. Jo, VTG schemes for using back propagation for multivariate time series prediction. *Appl. Soft Comput.* **13**(5), 2692–2702 (2013)
9. M. Sassano, Virtual examples for text classification with support vector machines, in *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing* (2003), pp. 208–215

Chapter 15

Temporal Learning



15.1 Introduction

The temporal learning is defined as the process of analyzing temporal relations among items through previous temporal sequences. We gather previous temporal sequences as a set of training examples. The HMM (Hidden Markov Model) is used as the learning algorithm for the temporal analysis. It is applied to the text topic analysis that is the temporal analysis of topics within a text. This section is intended to describe briefly the parameters of temporal learning and its learning and generalization.

Let us consider the parameters of temporal learning in using the HMM. The initial probabilities of states are given as a vector. The probabilities of state transitions are given as a square matrix. Conditional probabilities of states given observations are given as a general matrix. The temporal learning process is to estimate or optimize the parameters.

Let us consider the process of estimating the parameters of the temporal learning that is mentioned above. The collection of observation sequences each of which is associated with its own state sequence is given as the training set. The state probabilities and the state transition probabilities are estimated by the state frequencies and the state sequence. The conditional probabilities of states given observations are estimated by the pairs of observation sequences and their associated state sequences. The state sequence is generated to the given observation sequence by the maximal likelihood as the generalization of the temporal learning.

Let us consider the process of estimating the parameters of the temporal learning that is mentioned above. The collection of observation sequences each of which is associated with its own state sequence is given as the training set. The state probabilities and the state transition probabilities are estimated by the state frequencies and the state sequence. The conditional probabilities of states given observations are estimated by the pairs of observation sequences and their associated

state sequences. The state sequence is generated to the given observation sequence by the maximal likelihood as the generalization of the temporal learning.

Let us consider the application areas to which we apply the temporal learning. The speech recognition where character sounds are given temporally is mentioned as a typical task. The text topic analysis, which is mentioned in Sect. 15.4, is the process of extracting temporal topic sequence from a text. The time series prediction is to extract a sequence in future from the current sequence or the past one. The signal processing is the analysis of temporal sequences that represent signals.

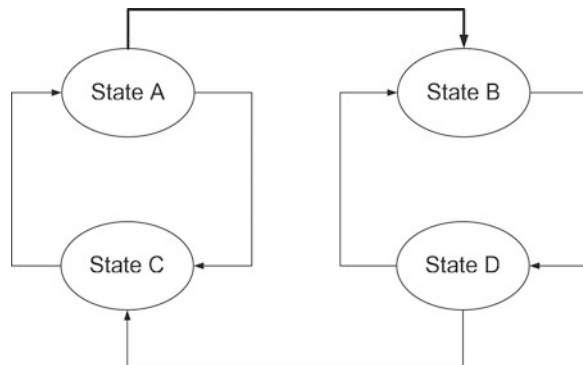
15.2 Discrete Markov Model

This section is concerned with the discrete Markov model for providing the background for understanding the Hidden Markov Model. In Sect. 15.2.1, we describe the state diagram that is given as the graphical form of the discrete Markov model. In Sects. 15.2.2 and 15.2.3, we cover the probabilities of state transitions and state sequences. In Sect. 15.2.4, we mention the application of the discrete Markov model to the task of time series prediction. This section is intended to describe in detail the discrete Markov model.

15.2.1 State Diagram

The state diagram is defined as the graphical form that shows transitions among states as illustrated in Fig. 15.1. Each node indicates a state in the state diagram. Each edge that is given as an arrow stands for a transition from a state to another state. The state diagram belongs to the special type of directed graph. This section is intended to describe the nodes and the edges of the state diagram, briefly.

Fig. 15.1 State diagram



Let us explain the nodes in the state diagram. Each node indicates a state. In Fig. 15.1, there are four states: A, B, C, and D. The first step of drawing the state diagram is to define the states. The initial state and the goal state are marked in the state diagram in the finite state automata [1].

Let us explain the edges in the state diagram. Each edge in the state diagram, which is shown in Fig. 15.1, is given as an arrow. In each edge, there are the source state in its start and the destination state in its end. An edge indicates the transition from the source state to the destination state. When the source state is identical to the destination one, the edge is given as cycle to the single state.

The state diagram is viewed as the special type of the graph. It is defined as the two sets: the vertex set and the edge set. The state set corresponds to the vertex set and the transition set does to the edge set, following the definition of the graph. Since each edge is given as an arrow and unweighted, the state diagram belongs to the directed and unweighted graph. If the transition probability may be assigned to each edge as its weight, the state diagram belongs to the directed and weighted graph.

Let us make some remarks on the state diagram that is mentioned in this section. The state diagram that is covered in this section belongs to the directed graph. The vertex set indicates a state set, and the edge set indicates a transition set in this kind of graph. The probability of a transition between states is given as an edge weight. The recursive cycle of a particular state indicates to stay in the same state.

15.2.2 State Transition Probability

This section is concerned with the state transition probabilities from the state diagram that is covered in Sect. 15.2.1. Before considering the state transition probabilities, we need to define the initial probability for each state. Each state transition probability is viewed as the conditional probability, $P(Dest|Start)$. The state transition probabilities are used for estimating the probability of state sequence. This section is intended to describe the initial state probabilities, the state transition probabilities, and estimation of state sequence probabilities.

The initial probabilities of the states are parameters of the discrete Markov model. The states are notated by $s_1, s_2, \dots, s_m$. The initial probabilities of m states are given as m dimensional vector, $\boldsymbol{\Pi} = [p_1 \ p_2 \ \dots \ p_m]$, where $p_i = P(s_i)$. The initial probability of the state, s_i , is based on the state frequency. It corresponds to the category prior to the probability learning.

The conditional probabilities that are given as a matrix are the parameters of the discrete Markov model. The conditional probability, $p(s_j|s_i)$, is the probability of transitioning from the state, s_i , to the state, s_j , and notated simply by p_{ij} . The transition probabilities are expressed as $m \times m$ matrix to the states, $s_1, s_2, \dots, s_m$, as follows:

$$\mathbf{T} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix}$$

The probability, p_{ii} , which corresponds to a diagonal element of the matrix $\mathbf{T}$, is the probability of recursive cycle of the state, s_i . The number of transition probabilities becomes m^2 to the m states.

Let us estimate the probability of a temporal state sequence. $\boldsymbol{\Pi}$ and $\mathbf{T}$ are the parameters that are defined above, and a temporal state sequence is given as $s(1), s(2), \dots, s(q)$. The probability of the state sequence, $s(1), s(2), \dots, s(q)$ is, computed by Eq. (15.1):

$$P(s(1), s(2), \dots, s(q)) = P(s(1))P(s(2)|s(1)) \dots P(s(q)|s(q-1)) \quad (15.1)$$

where $P(s(1))$ is the initial probability of the state, $s(1)$, which is obtained from $\boldsymbol{\Pi}$, and $P(s(i)|s(i-1))$ is the transition probability from the state, $s(i-1)$, to the state, $s(i)$, which is obtained from the matrix, $\mathbf{T}$. If no transition between states at least one time in the state sequence, its probability becomes zero.

Let us make some remarks on the state transition probability in the discrete Markov model. The initial probabilities of the states are given as a vector, $\boldsymbol{\Pi}$, and the transition probabilities are given as a matrix, $\mathbf{T}$. The task of the discrete Markov model is to estimate the probability of a temporal state sequence, given the parameters. We search for the temporal state sequence with its maximum probability, using the parameters. The parameters are estimated given a collection of temporal state sequences.

15.2.3 State Path Probability

This section is concerned with the learning process of the discrete Markov model for estimating parameters. They are assumed to be the initial probability vector, $\boldsymbol{\Pi}$, and the transition probability matrix, $\mathbf{T}$. The temporal state sequence is given as an input, and its probability is estimated by the parameters. The parameters are defined, depending on prior knowledge manually or automatically by analyzing a collection of temporal state sequences. This section is intended to describe the process of estimating the parameters from the temporal state sequences that are given as the training examples.

Let us mention the training set for the temporal learning by the discrete Markov model. The state set is defined as $S = \{s_1, s_2, \dots, s_m\}$. The collection of n state sequences is given as follows:

$$s_{11}, s_{12}, \dots, s_{1T}$$

$$s_{21}, s_{22}, \dots, s_{2T}$$

$$\dots$$

$$s_{n1}, s_{n2}, \dots, s_{nT}$$

where $s_{ij} \in S$. n state sequences that are presented above are used for estimating parameters that are mentioned in Sect. 15.2.2. We need an enough number of state sequences as the training examples, for computing the parameters, robustly.

Let us mention the process of estimating the parameters from the training state sequences as the learning process of the discrete Markov model. The vector of initial state probability is estimated based on the relative frequency of the first state in the state sequences by Eq. (15.2):

$$P(s_i) = \frac{\#(s_i \text{ as first state})}{n} \quad (15.2)$$

The matrix of the state transition probability, $P(s_j|s_i)$, is estimated based on the relative frequency of the training sequences, by Eq. (15.3),

$$P(s_j|s_i) = \frac{\#(s_i s_j)}{n(T-1)} \quad (15.3)$$

$T-1$ indicates the number of temporal two states from each sequence, and n is the total number of states sequences that are given as the training examples. When the number of states is very high, but the number of training state sequences is very small relatively, the distribution of the initial state probability vector and the state transition probability matrix is very sparse.

The next state is predicted given the current state, using the discrete Markov model. The state s_c is set as the current state, and $S = \{s_1, s_2, \dots, s_m\}$ is set as the state set. The probabilities, $P(s_1|s_c), P(s_2|s_c), \dots, P(s_m|s_c)$, are computed for each state, by Eq. (15.3). The state is decided as the next state to the current state by Eq. (15.4):

$$s_{\max} = \operatorname{argmax}_{s \in S} P(s|s_c) \quad (15.4)$$

This process may be applied to the time series prediction that predicts the future value, given the current value.

The next state is predicted given the previous states as well as the current state. s_p, s_c , and s_f are, respectively, the previous state, the current state, and the future space. The state conditional probabilities, $P(s_1|s_p s_c), P(s_2|s_p s_c), \dots, P(s_m|s_p s_c)$, by Eq. (15.5),

$$P(s_f|s_c s_p) = P(s_f|s_c)P(s_f|s_p)P(s_p) \quad (15.5)$$

The state conditional probability is computed in the case of more than one previous state, $s_{p1}, s_{p1}, \dots, s_{pT}$, by Eq. (15.6),

$$P(s_f | s_{p1}, \dots, s_{pT} s_c) = P(s_f | s_c) P(s_c | s_{pT}) P(s_{pT} | s_{pT-1}) \dots P(s_{p2} | s_{p1}) P(s_{p1}) \quad (15.6)$$

We will study the process of applying this model to the time series prediction in the next section.

15.2.4 Application to Time Series Prediction

This section is concerned with the process of applying the discrete Markov model to the time series prediction. In the previous sections, we studied the parameters and the learning process of the discrete Markov model. The change between adjacent temporal measures in the time series is defined as a state, and the state sequence is generated from the time series by sliding the window. The parameters of the discrete Markov model are estimated by learning state sequences, and the next state is predicted using the parameters. This section is intended to mention the application of the discrete Markov model to the time series prediction.

The first step of applying the discrete Markov model to a real problem is to define the states. In this task, the three states are defined as increment, decrement, and no change. The state set is notated by $S = \{s_{in}, s_{no}, s_{de}\}$. s_{in} is $X(t-1) < X(t)$, s_{no} is $X(t-1) = X(t)$, and s_{de} is $X(t-1) > X(t)$. In the real case, the three states are defined as follows:

$$s_{in} : X(t-1) < X(t) + \alpha$$

$$s_{no} : X(t-1) - \alpha \leq X(t) \leq X(t-1) + \alpha$$

$$s_{de} : X(t-1) - \alpha < X(t)$$

Let us mention the process of estimating the initial probability vector, $\boldsymbol{\Pi}$, and the transition probability matrix, $\boldsymbol{\Sigma}$ from the time series. It is given temporally as $X(1), X(2), \dots, X(T)$, and series are generated by d sized slide window as follows:

$$X(1), X(2), \dots, X(d)$$

$$X(2), X(3), \dots, X(d+1)$$

$$\dots$$

$$X(T-d), X(T-d+1), \dots, X(T)$$

For each d sized series, by the defined rules, $d - 1$ sized temporal state sequences are generated as follows:

$$\begin{aligned}
 &s(1), s(2), \dots, s(d-1) \\
 &s(2), s(3), \dots, s(d) \\
 &\dots \\
 &s(T-d+1), s(T-d+2), \dots, s(T)
 \end{aligned}$$

The parameters of the discrete Markov model, Π and Σ , are estimated by the process that was described in Sect. 15.2.3. The next state is estimated using the estimated parameters.

The discrete Markov model is applied for predicting the next state, given the state sequence. The parameters, Π and Σ , are estimated by the learning process that was described in Sect. 15.2.3. The state sequence is given as $s(T-d+1), s(T-d+2), \dots, s(T)$, and the state, $s(T+1)$, is predicted. The conditional probabilities, $p(s_{in}|s(T-d+1)s(T-d+2) \dots s(T))$, $p(s_{de}|s(T-d+1)s(T-d+2) \dots s(T))$, and $p(s_{no}|s(T-d+1)s(T-d+2) \dots s(T))$, are computed, and the state that corresponds to the maximum conditional probability is selected as the next state, as shown in Eq. (15.7),

$$s(T+1) = s_{\max} \quad (15.7)$$

This process is viewed into applying the discrete Markov model for classifying a state sequence into one of the three categories.

Let us make some remarks on applying the discrete Markov model to a real problem. Defining a finite number of states is the important step of applying the model. The sufficient number of state sequences is required for predicting the next state reliably by the discrete Markov model. Because the states are observed not easily in the real problems, the states are given as the hidden variables. We need to expand the discrete Markov model into the Hidden Markov Model, which is covered in Sect. 15.3, by adding observations.

15.3 Hidden Markov Model

This section is concerned with the HMM (Hidden Markov Model), which is expanded from the discrete Markov model by adding observations. In Sect. 15.3.1, we present the parameters of the HMM mathematically. In Sects. 15.3.2 and 15.3.3, we describe the process of estimating the probabilities of a state sequence and an observation sequence using the HMM. In Sect. 15.3.4, we explain the process of

estimating the parameters of HMM, as its learning process. This section is intended to describe the HMM as a temporal learning algorithm.

15.3.1 Initial Parameters

This section is concerned with the initial parameters of the HMM (Hidden Markov Model). We studied the discrete Markov model in Sect. 15.2, and the initial state probabilities and the state transition probabilities are its parameters. In the HMM, the observations are added, so the conditional probabilities of states given observations are added as the parameters. The fact that the states are hidden in the given problem is the reason for calling this model HMM. This section is intended to describe and notate mathematically the initial parameters of the HMM.

The first parameter of the HMM is the vector of the initial state probabilities. It is assumed that the states are given as elements of the set, $S = \{s_1, s_2, \dots, s_m\}$. The initial state probability, $P(s_i)$, is notated by p_i . The vector of the state probabilities is notated by $\boldsymbol{\Pi} = \{p_1, p_2, \dots, p_m\}$. The initial state probability vector was mentioned in Sect. 15.2.2, as a parameter of the discrete Markov model.

The second parameter of the HMM is the matrix of the state transition probabilities. The state transition probability, which is the conditional probability of the state, s_i , given the state s_j , is notated by p_{ij} . The second parameter is expressed as a $m \times m$ square matrix, as follows:

$$\mathbf{T} = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1m} \\ p_{21} & p_{22} & \dots & p_{2m} \\ \dots & \dots & \dots & \dots \\ p_{m1} & p_{m2} & \dots & p_{mm} \end{bmatrix}$$

In the element of the matrix $\mathbf{T}$, the row index stands for indexing the destination state, and the column index stands for indexing the source state. The state transition matrix is also the parameter of the discrete Markov model.

The state observation matrix is added as one more parameter in the HMM. The observations are defined as a set, $O = \{o_1, o_2, \dots, o_p\}$, and the assumption that the observations are visible and the states are invisible is the reason for calling this model HMM. The conditional probability of the observation, o_j given the state, s_j , $P(o_j|s_i)$, is notated by r_{ij} , and the state observation matrix where each column corresponds to an observation and each row corresponds to a state is defined as follows:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1p} \\ r_{21} & r_{22} & \dots & r_{2p} \\ \dots & \dots & \dots & \dots \\ r_{m1} & r_{m2} & \dots & r_{mp} \end{bmatrix}$$

The two matrices are given as the parameters of the HMM: one is the $m \times m$ matrix and the other is $m \times p$ matrix. The observations and the states are defined as the essential process in using the HMM to a real problem.

Let us compare the HMM with the discrete Markov model. The discrete Markov model is expanded into the HMM by adding the observations that are related with the states. The assumption that is inherent in the discrete Markov model is that the states are visible, whereas the assumption that is so in the HMM is that the states are not visible. A set of states should be defined in the discrete Markov model, whereas both sets of states and observations should be defined in the HMM. The causal relations between the states and the observations are given as a matrix in the HMM.

15.3.2 Observation Sequence Probability

This section is concerned with the process of estimating the probability of an observation sequence. It is assumed that the parameters of the HMM, the initial probability vector, the state transition matrix, and the state observation matrix, are initially set. The observation sequence is given as the input, and its probability is estimated using the parameters. Because the cost for estimating the probability in the primitive was very high, the two recursive computations, the forward computation and the backward computation, are mentioned. This section is intended to describe the process of computing the probability of an observation sequence.

Let us mention the process of computing the probability of the temporal observation sequence, $o(1), o(2), \dots, o(T)$, using the parameters that are defined in Sect. 15.3.1. The probability of the temporal observation sequence given a temporal state sequence by product of conditional probabilities of an observation given a state is as shown in Eq. (15.8):

$$P(o(1)o(2) \dots o(T) | s(1)s(2) \dots s(T)) = \prod_{i=1}^T P(o(i) | s(i)) \quad (15.8)$$

The probability of the temporal state sequence, $P(s(1)s(2) \dots s(T))$, is computed by Eq. (15.9) based on the state transition matrix:

$$P(s(1)s(2) \dots s(T)) = p_1 \prod_{i=2}^T p_{(i-1)i} \quad (15.9)$$

The joint probability of both the temporal state sequence, $s(1)s(2) \dots s(T)$, and the temporal observation sequence, $o(1), o(2), \dots, o(T)$, $P(o(1)o(2) \dots o(T), s(1)s(2) \dots s(T))$, is computed by Eq. (15.10):

$$P(o(1)o(2) \dots o(T), s(1)s(2) \dots s(T)) = p_1 \prod_{i=2}^T p_{(i-1)i} \prod_{i=1}^T P(o(i)|s(i)) \quad (15.10)$$

The probability of the temporal observation sequence, $o(1), o(2), \dots, o(T)$, $P(o(1)o(2) \dots o(T))$, is computed by Eq. (15.10):

$$P(o(1)o(2) \dots o(T)) = \sum_{\text{all possible state sequence}} P(o(1)o(2) \dots o(T), s(1)s(2) \dots s(T)) \quad (15.11)$$

Because all state sequences are considered, it takes very high complexity to compute the probability of the observation sequence, so it is computed by the iteration, which is shown in Fig. 15.2, called forward computation. The forward variables are defined as Eq. (15.12):

$$\alpha_1(i) = P(o(1), s_i) = P(o(1)|s_i)P(s_i) \quad (15.12)$$

The computation proceeds recursively, from 1 to $t + 1$, according to Fig. 15.2, by Eq. (15.13),

$$\alpha_{t+1}(i) = \left[\sum_{k=1}^m \alpha_t(k) p_{ki} \right] P(o(t+1)|s_i) \quad (15.13)$$

The probability, $P(s_i|o(t+1))$, is taken from the parameter, the observation state matrix. The probability of the observation sequence is $P(o(1)o(2) \dots o(T))$, by iterating recursively, from the first observation to the last one; this becomes the reason for calling this scheme forward computation.

The probability, $P(s_i|o(t+1))$, is taken from the parameter, the observation state matrix. The probability of the observation sequence is $P(o(1)o(2) \dots o(T))$,

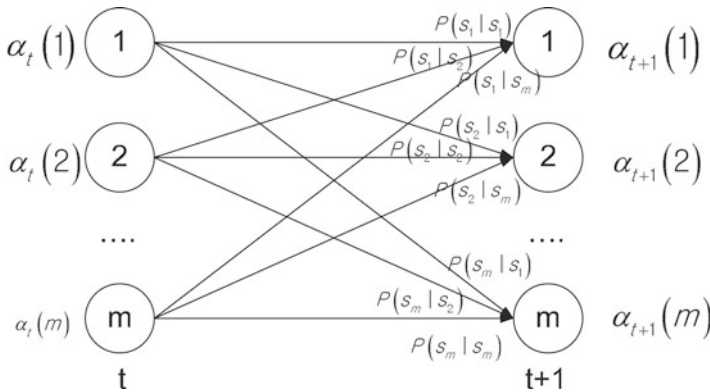


Fig. 15.2 Forward computation

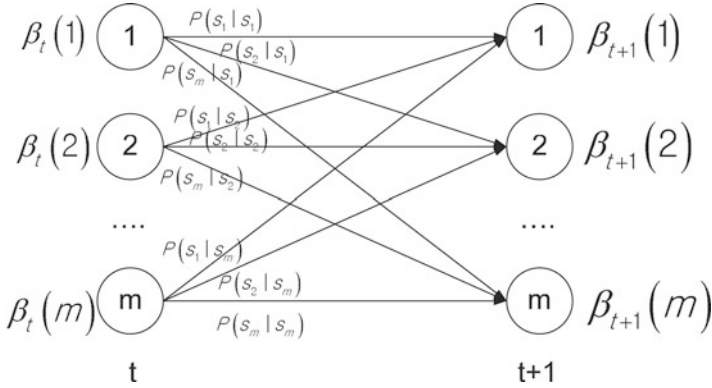


Fig. 15.3 Backward computation

by iterating recursively, from the first observation to the last one; this becomes the reason for calling this scheme forward computation. Let us mention another scheme of computing the probability of the observation sequence that is shown in Fig. 15.3. The backward variable is initialized from the last term, as $\beta_T = 1$. From $t + 1$ to 1, according to Fig. 15.3, the recursion is given as Eq. (15.14):

$$\beta_t(i) = \sum_{j=1}^m p_{ij} P(o(t+1)|s_j) \beta_{t+1}(j) \quad (15.14)$$

The results from computing the probability of the observation sequence in the backward computation or forward computation are the same to each other. The fact that the probability of the state sequence is computed recursively from the last to the first is the reason for calling this scheme backward computation.

Let us make some remarks on the process of estimating the probability of an observation sequence. It is assumed that the parameters, the initial state probabilities, the state transition probabilities, and the conditional probabilities of the observations given the states, are initially optimized. The process of computing the probability of an observation sequence given a state sequence looks simple, but because all possible state sequences are considered, it takes high complexity for computing the probability of an observation sequence. The two iterative schemes, the forward computation and the backward computation, were proposed for computing the probability. The HMM is used more frequently for finding the most probable state sequence given an observation sequence.

15.3.3 State Sequence Estimation

This section is concerned with another task using the HMM, generating the most probable state sequence, given an observation sequence. We studied the process of estimating the probability of an observation sequence, in Sect. 15.3.2. The three parameters of the HMM are initially defined, and the most probable temporal state sequence is decided, given an observation sequence. It is assumed that the observation sequence is visible externally and is initially shown from the task. This section is intended to describe the process of generating the most probable temporal state sequence, given a temporal observation sequence.

Let us compute the probability of the observation sequence, $o(1), o(2), \dots, o(t)$, and the state at time, $s(t)$, $P(o(1)o(2) \dots o(t)s(t))$. The probability is expressed by Eq. (15.15):

$$\begin{aligned}
 P(o(1)o(2) \dots o(t)s(t)) &= P(o(1)o(2) \dots o(t)|s(t))P(s(t)) \\
 &= P(o(1)o(2) \dots o(t-1)|s(t))P(o(t)|s(t))P(s(t)) \\
 &= P(o(1)o(2) \dots o(t-1)|s(t))P(s(t))P(o(t)|s(t)) \\
 &= P(o(1)o(2) \dots o(t-1)s(t))P(o(t)|s(t))
 \end{aligned} \tag{15.15}$$

where $P(o(t)|s(t))$ is obtained from the conditional probability matrix, **R**. The first term in Eq. (15.15), $P(o(1)o(2) \dots o(t-1)s(t))$ is expressed into Eq. (15.16):

$$\begin{aligned}
 P(o(1)o(2) \dots o(t-1)s(t)) &= \sum_{\text{all possible } s(t-1)} P(o(1)o(2) \dots o(t-1)s(t-1)s(t)) \\
 &= \sum_{\text{all possible } s(t-1)} P(o(1)o(2) \dots o(t-1)s(t)|s(t-1))P(s(t-1)) \\
 &= \sum_{\text{all possible } s(t-1)} P(o(1)o(2) \dots o(t-1)|s(t-1))P(s(t)|s(t-1))P(s(t-1)) \\
 &= \sum_{\text{all possible } s(t-1)} P(o(1)o(2) \dots o(t-1)s(t-1))P(s(t)|s(t-1))
 \end{aligned} \tag{15.16}$$

where $P(s(t)|s(t-1))$ is obtained from the state transition matrix, **T**. The above equation is expressed into Eq. (15.17):

$$\begin{aligned}
& P(o(1)o(2) \dots o(t-1)s(t)) \\
&= \sum_{\text{all possible } s(t-1)} P(o(1)o(2) \dots o(t-1)s(t-1))P(s(t)|s(t-1)) \\
&= \sum_{\text{all possible } s(t-1)} \sum_{\text{all possible } s(t-2)} P(o(1)o(2) \dots o(t-1) \\
&\quad |s(t-2))P(s(t-1)|s(t-2))P(s(t)|s(t-1)) \\
&= \sum_{\text{all possible } s(t-1)} \sum_{\text{all possible } s(t-2)} \dots \sum_{\text{all possible } s(1)} \\
&\quad P(o(1)s(1)) \prod_{i=2}^t P(s(i)|s(i-1)) \tag{15.17} \\
&= \sum_{\text{all possible } s(t-1)} \sum_{\text{all possible } s(t-2)} \dots \sum_{\text{all possible } s(1)} P(o(1)|s(1)) \\
&\quad P(s(1)) \prod_{i=2}^t P(s(i)|s(i-1)) \\
&= \sum_{\text{all possible } s(1)s(2) \dots s(t-1)} P(o(1)|s(1))P(s(1)) \\
&\quad \prod_{i=2}^t P(s(i)|s(i-1))
\end{aligned}$$

where $P(o(1)|s(1))$ is obtained from **T**. The two parameters, the conditional probability matrix and the state transition matrix, are involved in computing the probability, $P(o(1)o(2) \dots o(t)s(t))$.

Let us compute conditional probability of the observation sequence given a state. $P(o(t+1)o(t+2) \dots o(T)|s(t))$ is expressed into Eq. (15.18):

$$\begin{aligned}
& P(o(t+1)o(t+2) \dots o(T)|s(t)) \\
&= \sum_{s(t+1) \in S} P(o(t+1)o(t+2) \dots o(T)s(t+1)|s(t)) \\
&= \sum_{s(t+1) \in S} P(o(t+1)o(t+2) \dots o(T)|s(t+1)s(t))P(s(t+1)|s(t)) \tag{15.18}
\end{aligned}$$

Equation (15.18) is expanded into Eq. (15.19):

$$\begin{aligned}
& \sum_{s(t+1) \in S} P(o(t+1)o(t+2) \dots o(T)|s(t+1)s(t))P(s(t+1)|s(t)) \\
&= \sum_{s(t+1) \in S} P(o(t+1)|s(t)s(t+1))P(o(t+2) \dots o(T)|s(t)s(t+1))P(s(t+1)|s(t)) \quad (15.19) \\
&= \sum_{s(t+1) \in S} P(o(t+1)|s(t+1))P(o(t+2) \dots o(T)|s(t)s(t+1))P(s(t+1)|s(t))
\end{aligned}$$

Equation (15.19) is expanded into Eq. (15.20):

$$\begin{aligned}
& \sum_{s(t+1) \in S} P(o(t+1)|s(t+1))P(o(t+2) \dots o(T)|s(t)s(t+1))P(s(t+1)|s(t)) \\
&= \sum_{s(t+1) \in S} \sum_{s(t+2) \in S} \dots \sum_{s(T) \in S} \prod_{i=1}^{T-i} P(o(t+i)|s(t+i))P(s(t+i)|s(t+i-1)) \\
&= \sum_{\text{all possible } s(t+1) \dots s(T)} \prod_{i=t+1}^T P(o(i)|s(i))P(s(i)|s(i-1)) \quad (15.20)
\end{aligned}$$

The state transition matrix and the observation state matrix are involved in computing $P(o(t+1)o(t+2) \dots o(T)|s(t))$. The conditional probability of the state at a particular time, t , is $s(t)$, given the observation sequence, $P(s(t)|o(1)o(2) \dots o(T))$. The conditional probability, $P(s(t)|o(1)o(2) \dots o(T))$, is expressed by the Bayes rule into Eq. (15.21):

$$P(s(t)|o(1)o(2) \dots o(T)) = \frac{P(o(1)o(2) \dots o(T)|s(t))P(s(t))}{P(o(1)o(2) \dots o(T))} \quad (15.21)$$

Equation (15.21) is expanded into Eq. (15.22):

$$\begin{aligned}
& \frac{P(o(1)o(2) \dots o(T)|s(t))P(s(t))}{P(o(1)o(2) \dots o(T))} \\
&= \frac{P(o(1)o(2) \dots o(t)|s(t))P(o(t+1)o(t+2) \dots o(T)|s(t))P(s(t))}{P(o(1)o(2) \dots o(T))} \\
&= \frac{P(o(1)o(2) \dots o(t)s(t))P(o(t+1)o(t+2) \dots o(T)|s(t))}{P(o(1)o(2) \dots o(T))} \quad (15.22)
\end{aligned}$$

$P(o(1)o(2) \dots o(T))$ in Eq. (15.22) is computed by the forward scheme or the backward scheme that was described in Sect. 15.3.2. Because we are interested in the state with its maximal probability, rather than in the exact maximum probability value, we do not need to compute the probability, $P(o(1)o(2) \dots o(T))$, which is given as the nominator in Eq. (15.22).

Let us mention the process of finding the most probable state sequence under the observation sequence. For each state, in time t , the probability, $P(s_i(t)|o(1)o(2)\dots o(T))$, is computed. The state with the maximal probability is selected at time t , as expressed in Eq. (15.23),

$$s_{\max} = \operatorname{argmax}_{i \in S} P(s_i(t)|o(1)o(2)\dots o(T)) \quad (15.23)$$

The state sequence is generated by Eq. (15.23) from time 1 to time T as $s_{\max}(1), s_{\max}(2), \dots, s_{\max}(T)$. The temporal observation sequence, $o(1), o(2), \dots, o(T)$, is converted into the most probable temporal state sequence, $s_{\max}(1), s_{\max}(2), \dots, s_{\max}(T)$, under the fixed parameters of the HMM.

15.3.4 HMM Learning

This section is concerned with the process of updating the parameters from the training examples in the HMM. It is assumed that the parameters, the initial state probabilities, the state transition probabilities, and conditional probabilities of the observations given the states, are initially defined. Each observation sequence is associated with its own state sequence in each training example, and the observation sequences that are labeled with the state sequences are collected. The three parameters are estimated using the observation sequences. This section is intended to describe the process of estimating the parameters of the HMM.

The training examples that are used for the temporal learning by the HMM are illustrated in Fig. 15.4. The observation sequences with their fixed lengths that are associated with their own state sequences are collected externally. In each training example, the observation sequence is given as the input, and the state sequence is given as the output. The task of the temporal learning is to estimate most probable state sequence given an observation sequence. Before doing the task, the three parameters are estimated, using the training examples.

$$\begin{aligned} O_1 &= O_{11}O_{12}\dots O_{1T}, S_1 = S_{11}S_{12}\dots S_{1T} \\ O_2 &= O_{21}O_{22}\dots O_{2T}, S_2 = S_{21}S_{22}\dots S_{2T} \\ &\dots \\ O_K &= O_{K1}O_{K2}\dots O_{KT}, S_K = S_{K1}S_{K2}\dots S_{KT} \end{aligned}$$

Fig. 15.4 Example in Hidden Markov learning

Let us explain the process of estimating the initial state probabilities and the state transition probabilities from the above training examples. The initial state probabilities are computed by their relative frequencies of the states using Eq. (15.24):

$$P(s_i) = \frac{freq(init(s_i))}{N_{seq}} \quad (15.24)$$

where $freq(init(s_i))$ is the frequency of the state, s_i , which is the first state of the sequence, and N_{seq} is the total number of the state sequences. The state transition probability is computed by Eq. (15.25):

$$P(s_j|s_i) = \frac{P(s_i s_j)}{P(s_i)} = \frac{freq(s_i s_j)}{freq(s_i)} \quad (15.25)$$

where $freq(s_i s_j)$ is the frequency of both states, s_i and s_j , and $freq(s_i)$ is the frequency of the state, s_i . The vector that consists of the initial state probabilities and the matrix that consists of the state transition probabilities are constructed from the training examples that are mentioned above. Same probability may be assigned to all of the initial state probabilities, depending on the application area, for the simplicity.

The training examples in Fig. 15.3 are presented into the two matrices, **O** and **S**, as follows:

$$\mathbf{O} = \begin{bmatrix} o_{11} & o_{12} & \dots & o_{1T} \\ o_{21} & o_{22} & \dots & o_{2T} \\ \dots & \dots & \dots & \dots \\ o_{K1} & o_{K2} & \dots & o_{KT} \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \dots & s_{1T} \\ s_{21} & s_{22} & \dots & s_{2T} \\ \dots & \dots & \dots & \dots \\ s_{K1} & s_{K2} & \dots & s_{KT} \end{bmatrix}$$

and the probability of the observation, o_j , and the state, s_i , $P(o_j, s_i)$, is computed by Eq. (15.26):

$$P(o_j s_i) = \frac{\#((o_{kt} = o_j) \wedge (s_{kt} = s_i))}{T \cdot K} \quad (15.26)$$

where $\#((o_{kt} = o_j) \wedge (s_{kt} = s_i))$ is the number of elements in the both matrices where $o_{kt} = o_j$ and $s_{kt} = s_i$ for $1 \leq k \leq K$ and $1 \leq t \leq T$. The conditional probability of the observation, o_j , given the state, s_i , $P(o_j|s_i)$, is computed by dividing the probability $P(o_j, s_i)$ by the probability $P(s_i)$. In the case where $P(s_1) = P(s_2) = \dots = P(s_m)$, the conditional probability, $P(o_j|s_i)$, may be approximated by the probability, $P(o_j, s_i)$, according to Eq. (15.27):

$$P(o_j|s_i) \approx P(o_j s_i) \quad (15.27)$$

Let us make some remarks on the learning process of the HMM that is described in this section. The training examples for the temporal learning by the HMM are given as associations of the temporal observation sequence and the temporal state sequence. The three parameters of the HMM are usually estimated through the training examples; the case of setting the parameters manually without any training example is very rare. The HMM is applied to the speech recognition where the observation sequence is given as a temporal signal and the state sequence is given as a text. It is applied to the temporal text topic analysis where the observation sequence is a word sequence and the state sequence is a topic sequence, in the next section.

15.4 Text Topic Analysis

This section is concerned with the process of applying the HMM to the temporal text topic analysis. In Sect. 15.4.1, we specify the task of temporal topic analysis in the functional view. In Sect. 15.4.2, we mention the process of collecting training examples. In Sect. 15.4.3, we describe the process of estimating the HMM parameters that are mentioned in Sect. 15.3. In Sect. 15.4.4, we explain the process of generating the most probable state sequence after the learning process.

15.4.1 Task Specification

The temporal topic analysis of a text is defined as the process of assigning a temporal topic sequence to the text, as shown in Fig. 15.5. The task is needed for understanding the temporal flow of the text contents. The task is specified in the functional and conceptual view, before applying the HMM to it. In this task, each text is assumed as one with the fixed number of paragraphs, and the ordered paragraphs in the text are viewed as the observation sequence. This section is intended to explain the task, conceptually.

The observation sequence for performing the text topic analysis is illustrated in Fig. 15.6. The text is given as more than one paragraph, and it is assumed that all of the texts have their fixed number of paragraphs. A text is expressed as a temporal sequence of words, by extracting manually a representative word for each paragraph. From each text, the temporal word sequence is expressed as an

Fig. 15.5 Text topic analysis





Fig. 15.6 Observation sequences in text topic analysis

$\text{Topic}_1 \text{Topic}_2, \dots, \text{Topic}_p$

Fig. 15.7 State sequences in text topic analysis

observation sequence. Because more than one is available in each paragraph, it is possible to express a single text as multiple observation sequences.

The state sequences in the text topic analysis are illustrated in Fig. 15.7. Before presenting the state sequences, a finite number of topics is predefined in advance. Each state indicates one among the predefined categories, and the temporal topic sequence is given as a state sequence. The fact that the temporal topic sequence is invisible, compared with the word sequence that corresponds to the paragraph sequence, is the reason for defining the topic sequence as the state sequence. The temporal topic sequence to the paragraphs that are generated as the results from the task reflects the flow of text contents.

Let us mention the parameters of the HMM in applying it to the text topic analysis. The initial state probabilities are assumed to be identical ones; $1/m$ where m is the number of states is assigned to each state as its initial probability. The state transition probabilities that are topic transition probabilities are given as a $m \times m$ matrix. Each conditional probability of an observation given a state is the probability of a word given a topic; the conditional probabilities are given as $n \times m$ matrix, where n is the number of words. The parameters are estimated by learning the training examples each of which is given as an association of each word sequence with its own topic sequence.

Let us make some remarks on the text topic analysis in the functional view. The text categorization that was mentioned as a similar task is to assign topics to texts based on their contents. The text topic analysis should be distinguished from the text categorization in that the topics are given as a temporal sequence, rather than an unordered list of them. The text categorization is intended to arrange the texts by their topics, whereas the text topic analysis is intended to view the flow of the text content. In future, we consider the variable number of paragraphs in each text in doing the task.

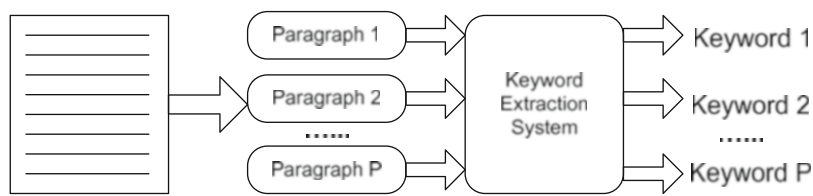


Fig. 15.8 Keyword extraction from paragraph

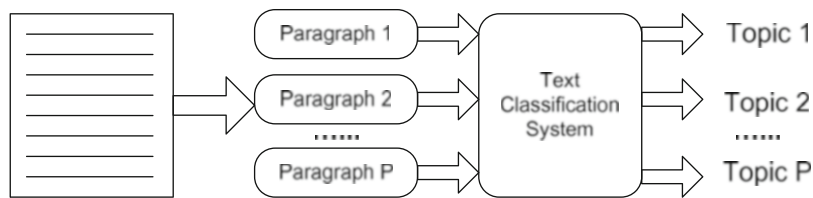


Fig. 15.9 Paragraph categorization

15.4.2 Sampling

This section is concerned with the process of collecting the training examples for the text topic analysis. Each training example for the temporal learning consists of the observation sequence and the state sequence. It is necessary to encode a text into a pair of the two sequences for applying the HMM to the task. We need the keyword extraction techniques and the text categorization techniques, respectively, for generating the observation sequence and the state sequence. This section is intended to study the process of generating the training examples from texts.

The process of extracting the observation sequence from a text is illustrated in Fig. 15.8. A text is partitioned into paragraphs by the carriage return. The most important word is extracted from each paragraph by the keyword extraction technique. The temporal sequence of the words each of which represents its own paragraph is the output of this process. The word sequence becomes the observation sequence.

The text categorization system for generating the state sequence from a text is illustrated in Fig. 15.9. It is assumed that the categories are predefined as a list, and the text is partitioned into paragraphs like the process of generating the observation sequences. Each paragraph is classified into one among the predefined categories by the text categorization system. The topics of the paragraphs are given as a temporal sequence that is the state sequence for the topic analysis. If each paragraph is classified softly into more than one category, all possible topic lists may be generated.

The process of generating the training examples is illustrated in Fig. 15.10. The keyword extraction for generating the observation sequences and the text categorization for generating the state sequences are presented in Figs. 15.8

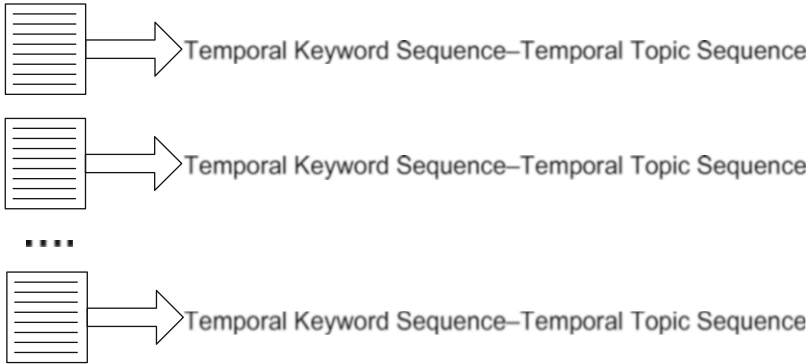


Fig. 15.10 Collection of state sequences and observation sequences

and 15.9. The pair of the observation sequence, which is given as a word sequence, and the state sequence, which is given as a topic sequence, is generated by each text. Figure 15.10 assumes the case where only one word is extracted from a paragraph, and only one topic is assigned to it.

Let us make some remarks on the process of gathering the samples from the texts for the temporal topic analysis. The temporal keyword sequences are given as the observation sequences, and the temporal topic sequences are given as the state sequence in this task. The keyword extraction module is used for extracting the observation sequences from the texts, and the text categorization module is used for extracting the state sequences. We need to consider the extraction of more than one word from each paragraph and the assignment of more than one topic to each paragraph. From the single text, we may gather more than one example, each of which is given as a pair of an observation sequence and a state sequence.

15.4.3 Learning

This section is concerned with the application of the temporal learning of the HMM to the text topic analysis. We already described the process of preparing the sample data from the texts in Sect. 15.4.2. The temporal learning is applied for collecting pairs each of which consists of an observation sequence and a state sequence, as the training examples. The HMM parameters are estimated through the learning, a novice text is encoded into an observation sequence, and the most probable state sequence is estimated. This section is intended to describe the process of applying the temporal learning to the text topic analysis.

The initial state probabilities are estimated as a vector from the training examples. The topics that are given as the states are defined as a finite set, $S = \{topic_1, topic_2, \dots, topic_m\}$. The initial state probability is computed by Eq. (15.28):

$$p(topic_i) = \frac{freq(s_1 = topic_i)}{N} \quad (15.28)$$

where N is the total number of the training examples, and $freq(s_1 = topic_i)$ is the number of the state sequences whose initial state, s_1 , is $topic_i$. The initial probabilities are computed as a m dimensional vector, $\boldsymbol{\Pi} = [p(topic_1) \ p(topic_2) \ \dots \ p(topic_m)]$. The topics are assumed as the independent ones without any relationships among them.

The state transition probabilities are estimated using the state sequences. The state transition probability in this application is the probability of the topic transition from one paragraph to its next one. The probability, $p(topic_i|topic_j)$, is computed by Eq. (15.29):

$$p(topic_i|topic_j) = \frac{freq(topic_i, topic_j)}{freq(topic_j)} \quad (15.29)$$

where $freq(topic_i, topic_j)$ is the frequency of the both topics, $topic_i$ and $topic_j$, and $freq(topic_j)$ is the frequency of the topic, $topic_j$, in the state sequence. The matrix of the topic transition probabilities is constructed as Eq. (15.30):

$$\mathbf{S} = \begin{bmatrix} p(topic_1|topic_1) & p(topic_2|topic_1) & \dots & p(topic_m|topic_1) \\ p(topic_1|topic_2) & p(topic_2|topic_2) & \dots & p(topic_m|topic_2) \\ \dots & \dots & \dots & \dots \\ p(topic_1|topic_m) & p(topic_2|topic_m) & \dots & p(topic_m|topic_m) \end{bmatrix} \quad (15.30)$$

where $p(topic_i|topic_i)$, which is given as a diagonal element in the matrix, $\mathbf{S}$, is the probability of no change of the topic in transition to the next paragraph.

Let us estimate the conditional probability matrix as the third parameter of the HMM. Each element in the matrix is the conditional probability of a state given an observation. In this application, the probability of a topic given a word, $p(topic_i|word_j)$, is defined. It is computed by Eq. (15.31):

$$p(topic_i|word_j) = \frac{\sum_{t=1}^T I((s_t = topic_i) \wedge (o_t = word_j))}{\sum_{t=1}^T I(o_t = word_j)} \quad (15.31)$$

The conditional probability may be found from a corpus, by Eq. (15.32) as the alternative way,

$$p(topic_i|word_j) = \frac{df(topic_i \wedge word_j)}{df(word_j)} \quad (15.32)$$

where $df(topic_i \wedge word_j)$ is the number of texts that include the word, $word_j$, and are relevant to the topic, $topic_i$, and $df(word_j)$ is the number of texts that include the word, $word_j$.

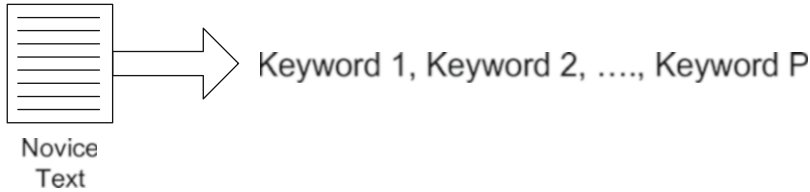


Fig. 15.11 Observation sequence from text

Let us make some remarks on the process of estimating the three parameters of the HMM from the training examples in the temporal topic analysis. The list of pairs, each of which consists of a temporal observation sequence and a temporal state sequence, is given as the training set for the temporal learning by the HMM. Each observation and state is set, respectively, as a word and a topic, in applying the HMM to the task. The initial topic probabilities are given as a vector, the topic transition probabilities are given as matrix, and the conditional probabilities of topics given words are also given as a matrix. The parameters are used for estimating most probable topic sequence to a novice word sequence from a text.

15.4.4 Topic Sequence

This section is concerned with the process of generating most probable topic sequence under the observation of a word sequence from a text. By the keyword extraction, a word sequence is observed from a text. Assuming no text classification system, most probable topic sequence is generated with the word sequence that is given as an observation sequence. In doing the text topic analysis, it is assumed that the topic in the current paragraph is dependent on ones in the previous paragraphs. This section is intended to describe the process of generating most probable topic sequence from a word sequence, using the HMM.

The process of generating the observation sequence from a novice text is illustrated in Fig. 15.11. The novice text with its fixed number of paragraphs is given as the input. Each paragraph is mapped into an important word temporally through the keyword extraction module. The temporal word sequence is generated as the observation sequence. Multiple word sequences may be considered by mapping each paragraph into more than one word.

The process of computing the probability of a topic sequence given a word sequence is illustrated in Fig. 15.12. A text is represented into a word sequence by the keyword extraction. A particular topic sequence is generated at random, and the probability of the topic sequence is computed by equation that is presented in Fig. 15.12. The task is to search for the topic sequence with its maximal probability, given a word sequence. We need to generate all possible topic sequences for finding the most probable one.

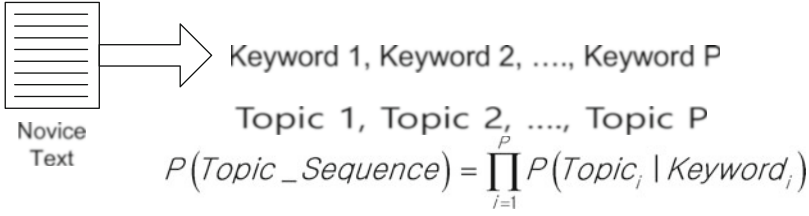


Fig. 15.12 State sequence probabilities

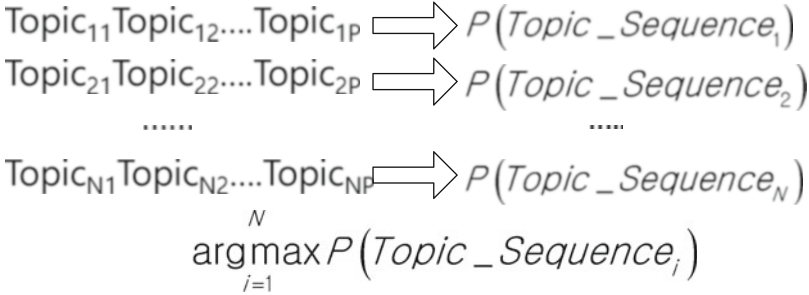


Fig. 15.13 Selection of state sequences

The process of deciding the most probable topic sequence is illustrated in Fig. 15.13. All possible temporal sequences are generated, and their probabilities are computed by the process of which is illustrated in Fig. 15.12. The topic sequence with its maximal probability is selected among them. Because of the product of the conditional probabilities of the states under the observations, we need to discover zero conditional probabilities for omitting the unnecessary computation. The iterative algorithm for searching for the most probable sequence should be developed.

Let us make some remarks on the temporal topic analysis on the text. The temporal dependency is assumed in deciding the topic in the current paragraph. The reason for applying the HMM for finding the most probable topic sequence is dependency of the topic in the current paragraph on ones in its previous paragraphs. Only text classifier may be applied to this problem in each paragraph independently. Because more than one keyword may be extracted from a single paragraph, we consider schemes of extracting observation sequences from a paragraph in the next study.

15.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We studied the discrete Markov model for modeling state transitions. The discrete Markov model was

expanded into the HMM by considering observation sequences as well as state sequences. The HMM was applied to the temporal topic analysis deciding most probable topic sequence by observing the temporal word sequence representing a text. In this chapter, we studied the discrete Markov model, the HMM, and its application to the text topic analysis.

A collection of labeled paragraphs is constructed as the corpus for applying the HMM to the temporal topic analysis. A list of predefined topics that are assigned to paragraphs is given as the state set, and a list of words that are discovered in paragraphs is given as the observation set. The paragraphs in each text is given as a temporal sequence, and the observation sequence and the state sequence are extracted from it. Because more than one word is included in each paragraph, or more than one topic is covered in it, it is possible to extract more than one observation sequence and one state sequence from a single text. In reality, much more training examples than texts are extracted in the corpus.

A text is viewed into a temporal topic sequence by performing the task that is mentioned in Sect. 15.4. We need the topic flow in the text for understanding the contents. The topic in the current paragraph is influenced by one of its previous one. We need to consider the topics of the previous paragraphs for deciding one of the current paragraph. The temporal learning is proposed for doing that rather than text classification system.

Let us consider the alternative way of estimating the HMM parameters in the temporal topic analysis. It is assumed that texts each of which consists of paragraphs temporally are initially given. The initial probability of each state is the rate of number of paragraphs that are leveled with the state to the total number of paragraphs. The conditional probability of the state given the observation is the rate of number of paragraphs that are labeled with the state and include the word that corresponds to the observation to the number of paragraphs with the corresponding word. The HMM parameters are estimated directly from the text collection without mapping them into the associations of states sequence and word sequence.

Let us mention of generating virtual texts from actual ones [2]. The virtual text refers to the text that is assembled artificially and temporally by paragraphs from different texts. In the text collection, each text is partitioned into its paragraphs, and several paragraphs that are relevant to a topic may be assembled into a text like form. The text mining techniques are needed for implementing it. The virtual text generation is described in detail in [2].

References

1. P. Linz, *An Introduction to Formal Languages and Automata* (Jones and Bartlett Publishers, Burlington, 2001)
2. T. Jo, *Text Mining: Concepts and Big Data Challenge* (Springer, New York, 2018)

Chapter 16

Reinforcement Learning



16.1 Introduction

The reinforcement learning is defined as the learning that receives the perception from the environment and generates the action based on the rewards and the penalties. In the initial stage, an action is generated randomly to a given perception, and a table is constructed based on the reward and the penalty. In the subsequent stages, the action that is associated with a similar perception or the exactly same one is selected from the table. While the supervisor provides exact answers to training examples in the supervised learning, the critics provide a reward or a penalty to the given action with some delay in the reinforcement learning. This section is intended to describe the basic concepts that are necessary for understanding the reinforcement learning.

Let us consider the process of preparing the reinforcement learning for the given problem. We define the perception that is entered from the outside through the sensors and the action that is taken properly to the given perception. The Q table where each perception is associated with its own action is initialized. The existing perception is updated with its more optimal action, and more associations of perceptions with actions are added in the Q table, through the learning. The Q table is initially fixed in the supervised learning, whereas the Q table is updated continually in the reinforcement learning.

Let us mention the process of learning the perceptions in the reinforcement learning and assume the autonomous agent with its sensors and its actuators. It receives the percepts from the outside through its sensors. It looks up the same percept or the most similar one in the Q table and extracts the action by the actuator. The reward or the penalty is taken by the critics, and the Q table is updated accordingly. Updating continually the Q table based on the reward and the penalty from taking the action is the learning process of the reinforcement learning.

Let us consider the components that are necessary for implementing the reinforcement learning. The sensors are needed for collecting percepts from the environment. The Q table is given as the collection of associations of perceptions

with their actions. The module is needed for deciding an action from a given perception by referring the Q table. The action as the response to the perception is generated by the actuator.

The goal of this chapter is to understand the reinforcement learning for implementing autonomous robots. We will understand the simple reinforcement learning algorithms that are applied to familiar tasks such as the classification and the regression. We understand the Q learning as the most popular reinforcement learning. We will also understand the advanced learning algorithms that are built by combining the reinforcement learning with the supervised learning algorithm and the unsupervised one. We will make the further discussion on what is studied in this chapter for implementing the autonomous vehicles and autonomous drones using the reinforcement learning algorithms.

16.2 Simple Reinforcement Learning

This section is concerned with the very simple environment of the reinforcement learning. In Sect. 16.2.1, we mention the situation where a simple example is given for the classification or the regression. In Sects. 16.2.2 and 16.2.3, we consider the classification and the regression to which we apply the reinforcement learning, even if the reinforcement learning is not applied to the tasks. In Sect. 16.2.4, we mention the autonomous agent that is implemented by the reinforcement learning. This section is intended to warm up understanding the reinforcement learning by mentioning the simplified one.

16.2.1 *Single Example*

This section is concerned with the case where a single example is given as a numerical vector. Let us assume the case of classifying a single example into one of the predefined categories. Let us assume another case of estimating a continuous output value based on the single example. We will mention the stochastic classification with the assumption of not fixed label of the example. This section is intended to describe the classification and the regression of the single example, for providing the background about the reinforcement learning.

Classifying a single example is illustrated as the reinforcement learning process in Fig. 16.1. A single example enters into the agent as a percept. If the corresponding example is not available in the table, the agent assigns an arbitrary label to it. If the label that is assigned to it is correct, the reward is given to the agent, and otherwise, the penalty is given. In the table, the example and its associated label are recorded, after doing it.

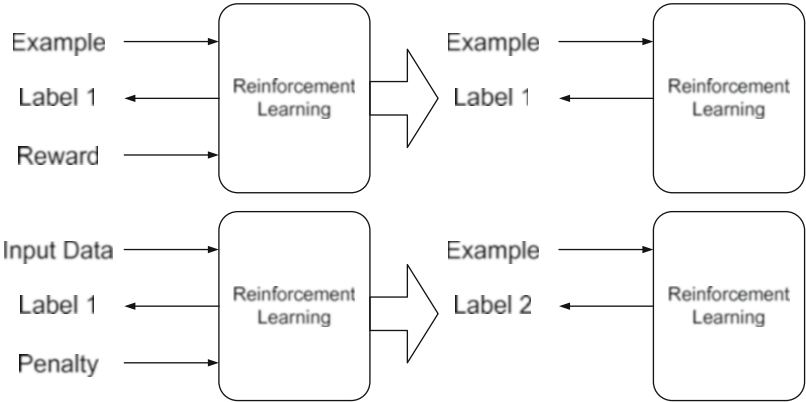


Fig. 16.1 Classification with single example

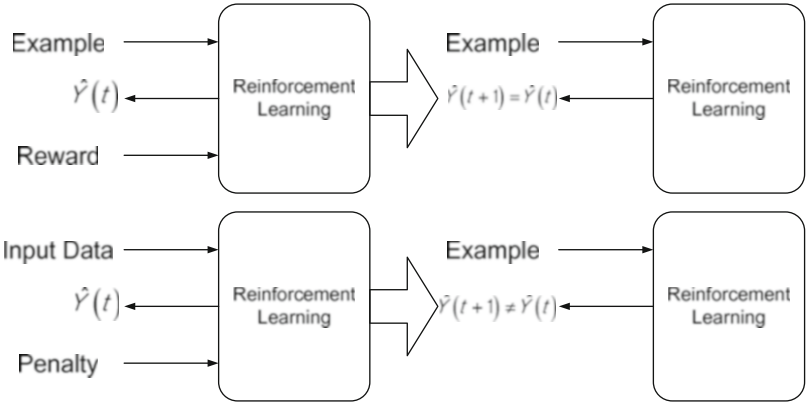


Fig. 16.2 Regression with single example

Estimating a continuous output value to the single example by the reinforcement learning is illustrated in Fig. 16.2. When the matching example is not available in the table, the output value is estimated arbitrary. If the output value matches with the target value, the reward is given. If the generated output differs from the target, the penalty is given. The example and its target output value are recorded in the table after doing it.

Let us consider the stochastic classification that is shown in Fig. 16.3, as a more complicated problem than the classification that is mentioned above. The stochastic classification is the classification task where the target label of each example is not fixed. Even if the example is classified correctly, it is not correct to classify it with the same label in the future. When the penalty is given to the classification of the item into the label from the table, one more entry should be added with the different

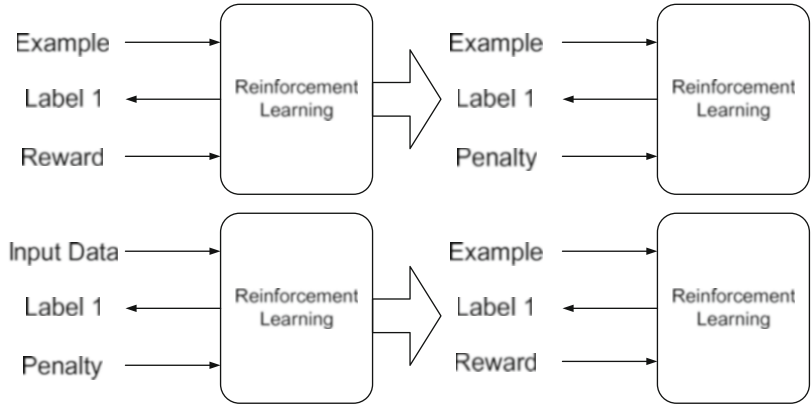


Fig. 16.3 Stochastic classification with single example

label. In this case, more weights are assigned to most recent correct label, and the item may be classified by the linear combination of weighted entries.

Let us make some remarks on the reinforcement learning to the classification and the regression. It is usually used for implementing an autonomous robot or a game agent. The percept through sensors is given as the input, and the action through the actuator is generated as the output. The reward and the penalty are given after making the action to the percept. The action is updated based on the reward and the penalty in the given table.

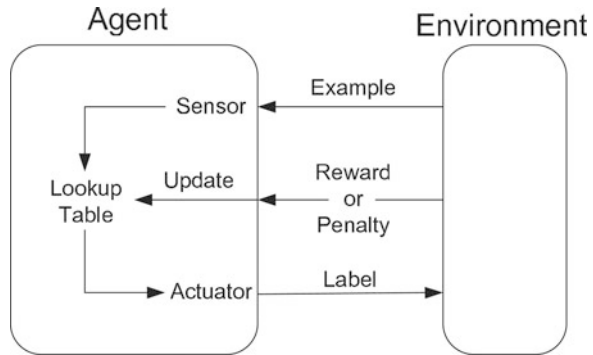
16.2.2 Classification

This section is concerned with the reinforcement learning process to the classification task. The percept is given as a data item that is the classification target, and the action is generated as the decision of its label. The look-up table is constructed using the training examples. Its matching entry or its most similar one is retrieved from the table for classifying the novice example. This section is intended to explain the reinforcement learning to the classification task.

The environment of the reinforcement learning for the classification task is illustrated in Fig. 16.4. The example as the classification target is received through the sensors. From the look-up table, a labeled example that is matching or similar to the input is retrieved. The label of the input is decided from the retrieved one, and the label is updated in the table, depending on the reward or the penalty. The time delay between the label decision and the reward or the penalty exists.

The process of constructing the look-up table as the learning process is illustrated in Fig. 16.5. The training examples are assumed to be unlabeled, and the penalty and the reward are given, depending on the classified label. The look-up table is initially

Fig. 16.4 Environment definition for classification



```

constructLookUpTable(List trainingExampleList, List categoryList){
    for trainingExample in trainingExampleList{
        categoryName = randomCategory(categoryList);
        lookUpTable.add(trainingExample, categoryName);
    }
    do until no penalty{
        for each entry in lookUpTable{
            if(isPenalty(entry)){
                classifiedCategoryName = entry.getClassifiedCategory();
                categoryName = randomOthers(categoryList, classifiedCategoryName);
                lookUpTable.update(entry, categoryName);
            }
        }
    }
}

```

Fig. 16.5 Look-up table construction in classification

constructed by assigning the category to each training example, at random, and each example is updated, depending on the reward and the penalty. The example is skipped in the reward, and its label is changed into another at random in the penalty. If the labeled training examples are given initially, the look-up table is constructed by them at a time.

The process of classifying an item by the reinforcement learning is illustrated in Fig. 16.6. The look-up table and the item are given as the input. The most similar item is retrieved from the look-up table, and the label is taken from the retrieved one. If penalty is given for classifying it, the entry in the look-up table is updated. This algorithm looks like the One Nearest Neighbor that was covered in Chap. 5.

Let us make some remarks on the reinforcement learning in the classification task that is mentioned in this section. If in the training examples, their target labels are fixed, the learning algorithm becomes the same to the One Nearest Neighbor. In the situation where the reinforcement learning algorithm is applied, the labels are not fixed. When the target label is changed, the penalty is put to the autonomous agent. The reinforcement learning is intended to the stochastic classification where the target label is changed as time goes.


```

classifyExample(Table lookUpTable, Item example){
    relevantEntry = selectRelevant(lookUpTable, example);
    categoryName = relevantEntry.getCategoryName();
    return categoryName;
}

selectRelevant(Table lookUpTable, Item example){
    for entry in loopUpTable{
        distance = computeDistance(example, entry);
        if(minDistance > distance){
            minimumIndex = currentIndex;
            minDistance = distance;
        }
    }
    return lookUpTable.getEntry(minimumIndex);
}

```

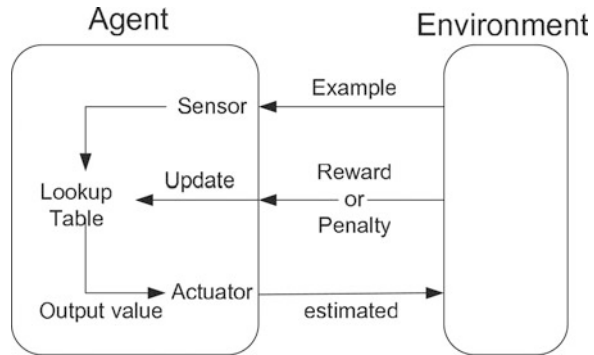
Fig. 16.6 Classification process based on One Nearest Neighbor

16.2.3 Regression

This section is concerned with the reinforcement learning for the regression task. It is the task that is expanded from the classification to which the supervised learning algorithms are applied. In the classification that is covered in Sect. 16.2.3, the action is to decide the label of a novice example, whereas the action in the regression is to estimate a continuous output value. In the classification, the penalty is the difference between the target class and the decided class, whereas in the regression, the penalty is the error between the target value and the estimated one. This section is intended to describe the reinforcement learning process in case of the regression task.

The environment of the autonomous agent for the regression is illustrated in Fig. 16.7. The percept that is given as the input vector is receipt through the sensors. An estimated output is generated as the action from the look-up table. When a penalty is given, the corresponding entry in the look-up table is updated. Because the output is given as a continuous value in the regression, the penalty is given always.

The process of constructing the look-up table for the regression task is illustrated in Fig. 16.8. The look-up table is initialized by generating a random output value between the maximum and the minimum for each entry. The output value is updated based on the penalty for each entry, by adding a random value between a negative constant and a positive constant, to the previous output value. In the case of the

Fig. 16.7 Environment definition for regression**Fig. 16.8** Look-up table construction in regression

```

constructLookUpTable(List trainingExampleList){
    for trainingExample in trainingExampleList{
        outputValue= randomBetween(min, max);
        lookUpTable.add(trainingExample, outputValue);
    }
    do until no penalty{
        for each entry in lookUpTable{
            if(isPenalty(entry)){
                outputValue = entry.getOutputValue();
                outputValue = outputValue + randomValue();
                lookUpTable.update(entry,outputValue);
            }
        }
    }
}

```

reward, the output value is not updated. The situation where the target output value is not available, but the penalty or the reward is given to each entry, is assumed.

The process of making the regression using the reinforcement learning is illustrated as the pseudo code in Fig. 16.9. The look-up table and a particular item are given as the input. The example that is most similar as the novice item is taken from the look-up table, as its most relevant one. The output value is taken from the most relevant one, and a random noise is added to it. This regression process is identical to the One Nearest Neighbor that is applied to the regression task.

Let us make some remarks on the reinforcement learning to the regression task that is mentioned in this section. It is assumed that the given task to which we apply the reinforcement learning is the regression. The reward and the penalty are set in this problem, based on the error between the target value and the estimated one. The reward may be given in the case of approximating the estimated value closely to the target value within the threshold, rather than the case of their exact matching. Until now, we have assumed that the target output is fixed.

```

estimateExample(Table lookUpTable, Item example){
    relevantEntry = selectRelevant(lookUpTable, example);
    outputValue = relevantEntry.getOutputValue () + randomNoise;
    return outputValue;
}

selectRelevant(Table lookUpTable, Item example){
    for entry in loopUpTable{
        distance = computeDistance(example, entry);
        if(minDistance > distance){
            minimumIndex = currentIndex;
            minDistance = distance;
        }
    }
    return lookUpTable.getEntry(minimumIndex);
}

```

Fig. 16.9 Regression process based on One Nearest Neighbor

16.2.4 Autonomous Moving

This section is concerned with the reinforcement learning for moving to the goal position autonomously. In the previous section, we studied the reinforcement learning for the classification and the regression that are familiar to us. The front view is given as the perception, and the selection of forward, left, and right is the action in the autonomous agent. Less distance from the goal point is given as the reward, and block or more distance from the goal is given as the penalty. This section is intended to describe the reinforcement learning in a simple autonomous moving, instead of the classification and the regression.

The environment for doing the autonomous moving is illustrated in Fig. 16.10. The front image is received through the sensors as the percept. One of the three is selected, and it is made into the action by the actuator. The reward is given when moving to the closer position to the goal, and the penalty is given when being blocked or moving to the father position. In this problem, it is assumed that each location is given as discrete ones.

The process of constructing the look-up table for doing the autonomous moving is illustrated in Fig. 16.11. The training examples are initially unlabeled, the three categories in the list are given as left, right, and forward, and the look-up table is constructed by assigning one of the three categories to each training example randomly. The front image is received through the sensors, the training example that is exactly matching or most similar is retrieved, and the action is taken, corresponding to the label of the retrieved one. If the penalty to the action is given, the label of the retrieved one is updated. The initial construction and the continual update of the look-up table become the reinforcement learning process.

The autonomous moving process after the reinforcement learning is illustrated as the pseudo code in Fig. 16.12. The percept is assumed to be the information

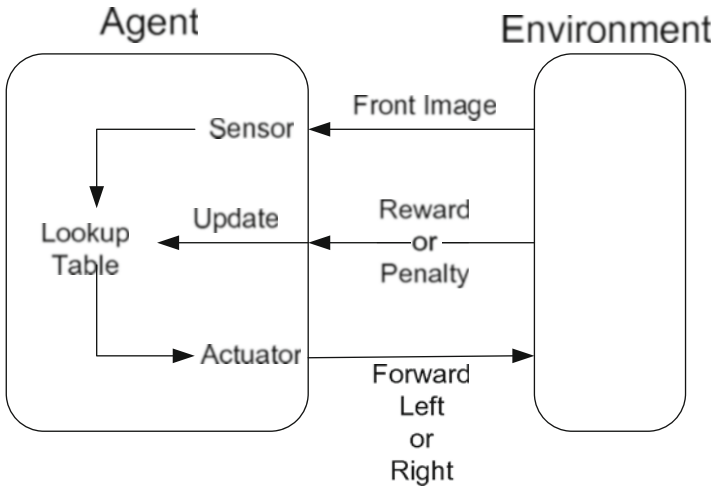


Fig. 16.10 Environment definition for autonomous moving

```

constructLookUpTable(List trainingExampleList, List categoryList){
    for trainingExample in trainingExampleList{
        categoryName = randomCategory(categoryList)
        lookUpTable.add(trainingExample, categoryName);
    }
    do until no penalty{
        for each entry in lookUpTable{
            if(isPenalty(entry)){
                classifiedCategoryName = entry.getClassifiedCategory();
                categoryName = randomOthers(categoryList, classifiedCategoryName);
                lookUpTable.update(entry, categoryName);
            }
        }
    }
}

```

Fig. 16.11 Look-up table construction in autonomous moving

about the current position, and a matching one or a similar one is retrieved from the look-up table. The action that is associated with the retrieved entry is taken as the output. The look-up table is initially constructed by setting an action to each entry at random and is updated subsequently by the reward or the penalty. When the path is replaced entirely with a new path, the look-up table should be re-updated.

Let us make some remarks on the learning process for the autonomous moving that is covered in this section. The path that is used for training the autonomous moving is given as very restricted space. The path where the autonomous moving in the real field is very much complicated. More actions may be added to the three basic ones, by diagonal directions as well as the perpendicular ones. The moving speed may be controlled rather than the constant moving speed, adding them as the actions.

```

decideAction(Table lookUpTable, Item percept){
    relevantEntry = selectRelevant(lookUpTable, percept);
    action = relevantEntry.getAction();
    return action;
}

selectRelevant(Table lookUpTable, Item example){
    for entry in loopUpTable{
        distance = computeDistance(example, entry);
        if(minDistance > distance){
            minimumIndex = currentIndex;
            minDistance = distance;
        }
    }
    return lookUpTable.getEntry(minimumIndex);
}

```

Fig. 16.12 Autonomous moving process

16.3 Q Learning

This section is concerned with the Q learning that is the reinforcement learning based on the Q table. In Sect. 16.3.1, we explain the Q table with respect to its content. In Sects. 16.3.2 and 16.3.3, we describe the process of updating the Q table as the Q learning process in both the finite states and the infinite states. In Sect. 16.3.4, we consider the reinforcement learning, in case where rewards are given differently to even same action. In this section, we focus on the Q learning process that updates the Q table.

16.3.1 Q Table

The Q table is defined as the collection of entries each of which consists of its percept and rewards and penalties of its actions. We studied the look-up table that consists of the percept and its target action in Sect. 16.2. The same action is assumed to be a cause of variable rewards or penalties, and the values of the reward and the penalties are assigned to all actions. The value that indicates the penalty degree is given as a negative continuous value, and one that indicates the reward degree is given as a positive continuous value; the values of the rewards and the penalties are defined arbitrarily in designing the reinforcement learning. This section is intended to describe the Q table as the policy of the reinforcement learning.

Percept	Action 1	Action 2	...	Action K
Percept 1	0	0		0
Percept 2	0	0		0
.....				
Percept N	0	0		0

Fig. 16.13 Initialization of Q table

Percept 1 → Action 2 → Reward
Percept 2 → Action K → Penalty



Percept	Action 1	Action 2	...	Action K
Percept 1	0	+K		0
Percept 2	0	0		-K
.....				
Percept N	0	0		0

Fig. 16.14 Update of Q table

The initialized Q table is illustrated in Fig. 16.13. Each row corresponds to a percept, and each column corresponds to a reward score of each action. The N percepts are defined, and the reward scores of all actions are initialized with zero values in each percept. The reward scores of actions may be initialized with non-zero values, depending on the prior knowledge. The Q table is updated after the initialization through the learning process.

The process of updating the Q table is illustrated in Fig. 16.14. The actions in each percept are initialized as zero values, and an action is selected at random. For example, action 1 is taken to percept 2, and action K is taken to percept 3. The reward is given to action 2 in percept 2, and the penalty is given to action K in percept 3. Action 2 in percept 2 is updated to a positive constant value, and action K in percept 3 is updated a negative constant value.

The final version of the Q table that is updated by the reward and the penalty is illustrated in Fig. 16.15. Action 1 has the highest reward, and action 4 has the highest penalty in percept 1. Action 3 is taken to the percept that is most similar to percept 2 or matching with it. Action 2 is generated to the percept that is similar to percept 3 or 4. It is assumed that the reward and the penalty are stochastic to the same percept, as time goes.

Percepts	Action 1	Action 2	Action 3	Action 4
Percept 1	5	-3	0	-5
Percept 2	-3	1	5	0
Percept 3	-6	4	1	0
Percept 4	0	6	-4	-3

Fig. 16.15 Actions from Q table

Let us make some remarks on the Q table for deciding an action to the percept. In the look-up table, each percept is associated with its desired action, whereas it is associated with a list of scores of all actions in the Q table. The action with its highest score becomes the desired action to the percept. The higher weight is assigned to the most recent reward or penalty in updating the Q table. One action is selected at random in the case of more than one action with a same score.

16.3.2 *Finite State*

This section is concerned with the process of updating the Q table for the classification task. In the Q learning, the input vector that represents an entry is a percept and its label is an action. The reward is given in the case where the action is consistent with the target label, and the penalty is given, otherwise. The Q learning, which is an instance of the reinforcement learning, is applied to the classification for understanding it easily. This section is intended to describe the Q table frame, the Q learning process, and the classification process.

The initialized Q table for the classification task by the reinforcement learning is illustrated in Fig. 16.16. In the Q table, each row corresponds to a percept and each column does to a label. The action in the reinforcement learning that is applied for the classification is the decision of labeling a novice item. When zero values are assigned to all categories, a label is decided at random. The Q table is continually updated, depending on the correct classification or the misclassification.

The Q table that is updated from the training examples is illustrated in Fig. 16.17. It is assumed that the training examples are labeled with their own labels, exclusively. A positive constant, +5, is assigned to the target label for each example. In the soft classification, the positive constants may be assigned to more than one label. The Q table for the classification is viewed as a set of the labeled training examples.

The process of classifying an item by the reinforcement learning algorithm is illustrated as a pseudo code in Fig. 16.18. A novice item is given as a percept, and it searches for an exactly matching one or the most similar one from the Q table. The label with the maximum reward point in the matching entry or the

Percept	Label 1	Label 2		Label K
Item 1	0	0		0
Item 2	0	0		0
.....				
Item N	0	0		0

Fig. 16.16 Q table in classification

Percept	Label 1	Label 2		Label K
Item 1	0	5		0
Item 2	5	0		0
.....				
Item N	0	0		5

Fig. 16.17 Updated Q table in classification

approximated one in the Q table is generated as the action. More than one percept may be retrieved from the Q table for deciding the action by averaging the reward points. The reinforcement learning for the classification task is interpreted into the One Nearest Neighbor that was covered in Chap. 5.

Let us make some remarks on the reinforcement learning for the classification task. In the Q table, each row corresponds to a percept, and each column does to an action score. The negative score is given as a penalty, and the positive one is given as the reward. The highest score is assigned to the action that corresponds to the target category, under the assumption that the labeled training examples are given. The reinforcement learning is used for the case where the target labels are not given, but the reward or the penalty is given to the decided action; the description of the reinforcement learning for the classification is intended to understanding the learning algorithm, easily.

16.3.3 Infinite State

This section is concerned with the reinforcement learning for the regression where the number of states is infinite. We studied the reinforcement learning using the Q table for the classification where the number of states is finite, in Sect. 16.3.2. The problem is expanded into a slightly more complicated problem, by assuming that

Fig. 16.18 Classification process based on One Nearest Neighbor with Q table

```
classifyExample(Table qTable, Item example, real distanceThreshold){
    relevantEntryList = selectRelevant(qTable, example, distanceThreshold);
    action = decideAction(qTable, relevantEntryList, example);
    return action;
}

selectRelevant(Table qTable, Item example, real distanceThreshold){
    for entry in qTable{
        distance = computeDistance(example, entry);
        if(distance < distanceThreshold)
            relevantList.add(entry);
    }
    return relevantList;
}

decideAction(Table qTable, List relevantEntryList, item example){
    maxActionReward = 0.0;
    maxActionIndex = 0;
    actionList = qTable.getActionList();
    for action in actionList{
        actionReward = 0.0;
        for relevantEntry in relevantList{
            distance = example.getDistance(relevantEntry);
            reward = relevantEntry.getReward(action);
            actionReward = actionReward + (1/distance)*reward;
        }
        if(maxActionReward < actionReward){
            maxActionReward = actionReward;
            maxActionIndex = action.getIndex();
        }
    }
    finalAction = actionList.getAction(maxActionIndex);
    return finalAction;
}
```

Percept	Interval 1	Interval 2		Interval K
Item 1	0	0		0
Item 2	0	0		0
.....				
Item N	0	0		0

Fig. 16.19 Q table in regression

the number of states is infinite. The problem that is covered in this section is the regression whose output values are continuous; the number of actions is infinite. This section is intended to describe the reinforcement learning for the regression.

The initial Q table for the regression is illustrated in Fig. 16.19. The output value in the regression is continuous, so we need to discretize it. The finite number of intervals is defined through the discretization. The regression is viewed into the classification of each input into one among the intervals. The reinforcement learning is applied into a multiple classification task in this case.

The updated Q table for the regression is illustrated in Fig. 16.20. The output value interval, which is called bin, is defined as an action. The reward score is assigned to the output value interval that corresponds to the target output value.

Percept	Interval 1	Interval 2		Interval K
Item 1	0	5		0
Item 2	5	0		0
....				
Item N	0	0		5

Fig. 16.20 Updated Q table in regression

```

regressExample(Table qTable, Item example, real distanceThreshold, real rate){
    relevantEntryList = selectRelevant(qTable, example, distanceThreshold);
    estimatedValue = estimateValue(qTable, relevantEntryList, example,rate);
    return action;
}

selectRelevant(Table qTable, Item example, real distanceThreshold){
    for entry in qTable{
        distance = computeDistance(example, entry);
        if(distance < distanceThreshold)
            relevantList.add(entry);
    }
    return relevantList;
}

estimateValue(Table qTable, List relevantEntryList, item example, real rate){
    totalValue = 0.0
    for relevantEntry in relevantList{
        distance = example.getDistance(relevantEntry);
        maxAction = relevantEntry.getMaxAction();
        estimatedValue = maxAction.getMidValue() + (1/reward) * distance * rate;
        totalVaue = totalValue + estimatedValue
    }
    return totalValue/relevantList.size();
}

```

Fig. 16.21 Regression process based on One Nearest Neighbor with Q table

The value, 5, is given as the reward score to the desired output value interval, as shown in Fig. 16.20. We may consider applying the penalty value that is given as a negative value to the interval that is far from the desired output value.

The regression process using the Q table is illustrated in Fig. 16.21. A novice input vector is initially given as the percept. Its most similar entries are selected from the Q table. The value intervals with their maximal scores are averaged. It looks similar to the KNN algorithm for doing the regression.

Let us make some remarks on the reinforcement learning for the regression task. The input vector is initially given as the percept, and the output value range is generated as the action. The Q table is updated, depending on the reward or the penalty, rather than the target output value. If the output value is fixed to

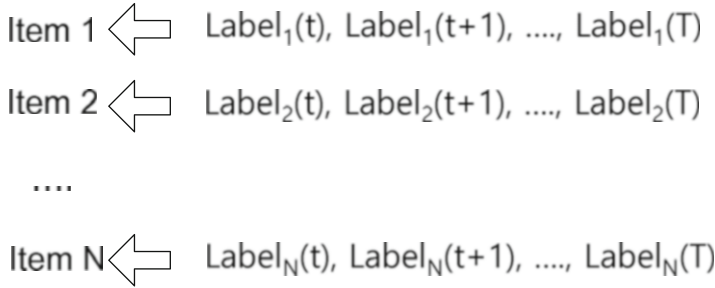


Fig. 16.22 Dynamic classification

the same input, the supervised learning is used for the regression. Otherwise, the reinforcement learning is recommended for the task.

16.3.4 Stochastic Reward

This section is concerned with the reinforcement learning for the classification or the regression where the output is stochastic to the same input. In the previous section, we explained the reinforcement learning for the tasks where the output is deterministic; it was intended to understand the learning process, easily. We need to consider the case where the output is different depending on the time to the same input. The penalty is given to the output that the reward was put previously. This section is intended to describe the reinforcement learning for the case of the stochastic output.

The dynamic classification is defined as the process of classifying even the same item into different labels, as time goes, as shown in Fig. 16.22. The N items are presented in Fig. 16.22. As time goes, the label is assigned to each item differently. The target label of each item changes continually in the dynamic classification. The reinforcement learning is applied to this task, rather than the supervised learning.

A score is updated in the Q table in each iteration as shown in Fig. 16.23. The previous scores and the current one of the action, A_i are notated by $Q(t)$, $Q(t-1)$, ..., $Q(t-k)$. The weights are assigned to them, following $w_i(t)$, $w_i(t-1)$, ..., $w_i(t-k)$. The action score is updated by the linear combination of the weights and the scores, as shown in Eq. (16.1),

$$Q(t+1) = \sum_{j=0}^k w_i(t-j) Q(t-j) \quad (16.1)$$

Higher portion is assigned to the recent reward in updating the score than past ones.

Percept	Label 1	Label 2		Label K
Item 1	5	4		0
Item 2	4	5		0
.....				
Item N	3	4		5

Fig. 16.23 Updated Q Table

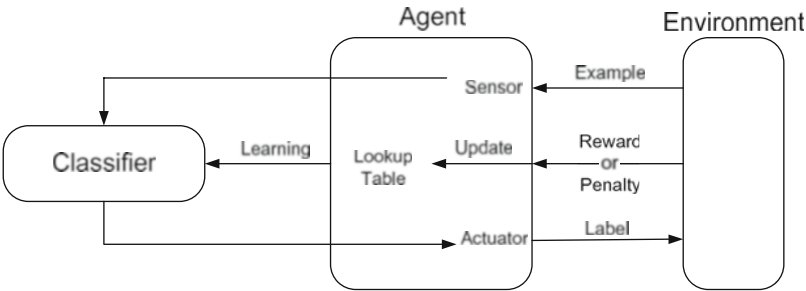


Fig. 16.24 Classification process based on reinforcement learning

The process of classifying an item based on the reinforcement learning is illustrated in Fig. 16.24. A novice example is given as the percept through the sensor, and the entry that matches with it is retrieved from the Q table. The action that is given as its label is generated by the actuator, and the agent waits for the reward and the penalty. The entry is updated by Eq. (16.1), to the corresponding action. In this section, the classification is identical to one that was mentioned in Sect. 16.3.2, except updating the score of the corresponding action.

Let us make some remarks on the reinforcement learning for the stochastic classification. It is assumed that the label of each item is not fixed; as time goes, the desired label changes. In computing the reward score in the Q table, higher weights are assigned to the desired label that is set recently. When the target label is fluctuated, we need to consider more additional attributes. The stochastic classification is expanded into the soft one where a list of desired categories changes continually.

16.4 Advanced Reinforcement Learning

This section is concerned with the reinforcement learning that is combined with the supervised learning or/and the unsupervised learning. In Sect. 16.4.1, we consider

the ensemble of reinforcement learning algorithms. In Sect. 16.4.2, we mention the combination of the reinforcement learning with the supervised learning. In Sect. 16.4.3, we try to combine it with the unsupervised learning as the alternative combination. In Sect. 16.4.4, we mention the prediction of future environment in advance.

16.4.1 Ensemble Reinforcement Learning

This section is concerned with the ensemble reinforcement learning as the combination of multiple reinforcement learning agents. We need the Q table from another agent that is learned in another environment to the given agent for improving its adaptability. The reinforcement agents are organized as shown in Fig. 16.25, and the voting is adopted as the combination scheme in this section. The schemes of combining the reinforcement learning agents with each other, such as the expert gate and the cascading, are considered as the further study. This section is intended to mention the combination of the learning agents by voting.

The agent and the environment for implementing the ensemble reinforcement learning are illustrated in Fig. 16.25. The M learning agents that are the committee members and the coordinator that controls them exist, rather than a single agent.

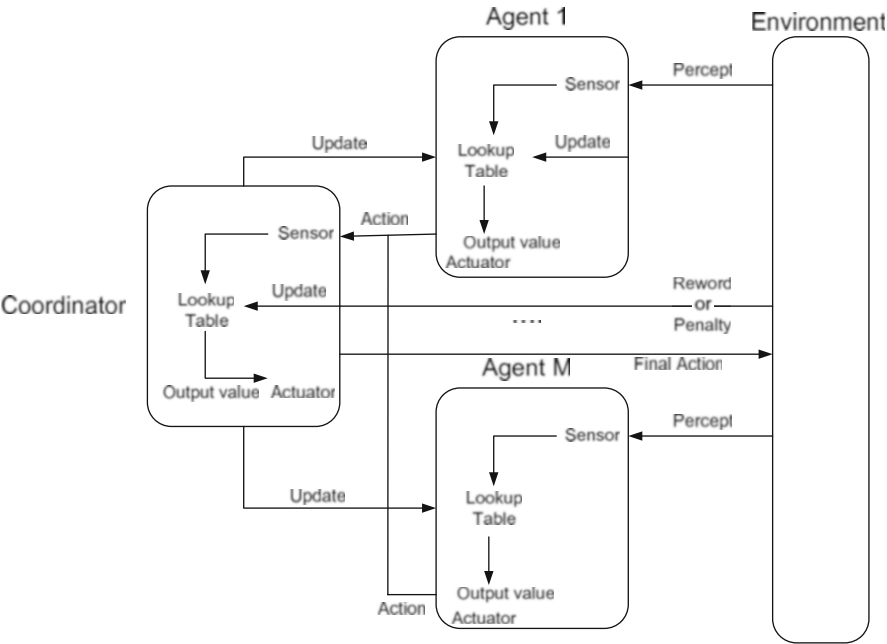


Fig. 16.25 Environment of ensemble reinforcement learning

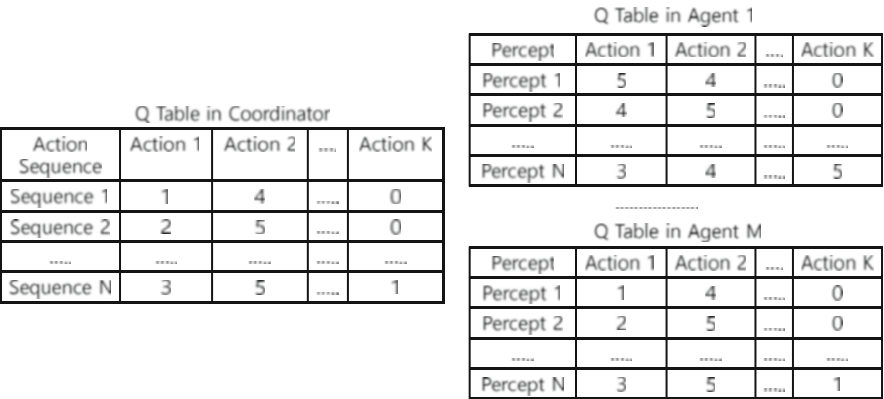


Fig. 16.26 Q tables in committee members and coordinator

Each committee member receives the percept from the external environment, and the actions from the committee members are sent to the coordinator, instead of generating it externally. The coordinator receives the actions and decides the final action. The process of receiving the percept and generating the action is like the supervised learning in the voting that was covered in Chap. 13.

The Q tables in both the committee members and the coordinator are illustrated in Fig. 16.26. Each column of the Q tables consists of K actions in both of them. Each row of the Q table indicates a percept to the committee members and an action sequence to the coordinator. The action sequence from the committee members are percepts to the coordinator. This case corresponds to one where the outputs from the committee members are the input to the coordinator in voting the supervised learning algorithms.

The generalization process in the voting of the reinforcement learning algorithms is illustrated in Fig. 16.27. Each agent in the committee receives the percept from the external environment. From each agent, its own action is generated, responding to the percept, and transferred to the coordinator. The coordinator receives the actions from the agents and decides the final action in its own Q table. The reinforcement learning in the coordinator belongs to the meta-learning that was mentioned in Sect. 13.3.

Let us make some remarks on the ensemble reinforcement learning that is illustrated in Figs. 16.25, 16.26, and 16.27. Each agent in the committee receives its percepts from the environment. The coordinator receives the actions that are taken by the committee members and decides the final action. The Q table is updated by both the coordinator and the committee members, depending on the reward and the penalty. The combination scheme of reinforcement learning algorithms, which are mentioned in this section, corresponds to the voting.

Fig. 16.27 Generalization in ensemble reinforcement learning

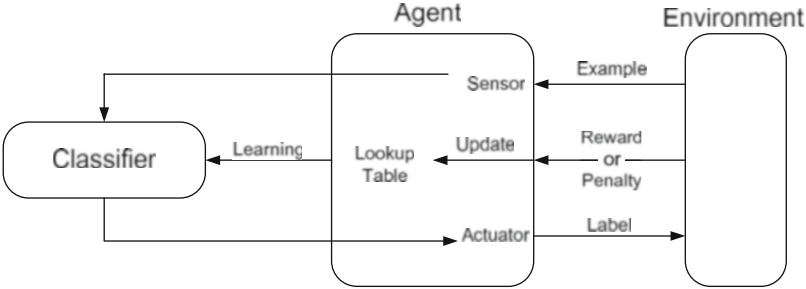
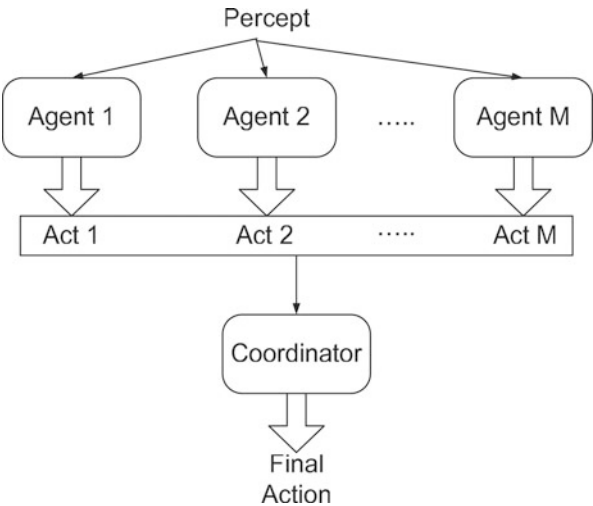


Fig. 16.28 Environment of Reinforcement + Supervised

16.4.2 Reinforcement + Supervised

This section is concerned with the combination of the reinforcement learning and the supervised learning. When starting this chapter, we mention the only reinforcement learning. The supervised learning is added for improving the adaptability in deciding the action. The Q table is updated through the reinforcement learning, and the action is decided with more flexibility by the classifier. This section is intended to describe the reinforcement learning that is supported by the supervised learning.

The environment, the autonomous agent, and the combination of the reinforcement learning and the supervised learning are illustrated in Fig. 16.28. The autonomous agent receives the percept from the external environment, and the classifier learns the entries in the Q table, in advance. The action is decided to the percept by the classifier. The entry of the Q table is updated with the reward or the penalty, and the classifier is retrained with the updated one. The action is decided by the supervised learning algorithm, instead of the reinforcement learning algorithm.

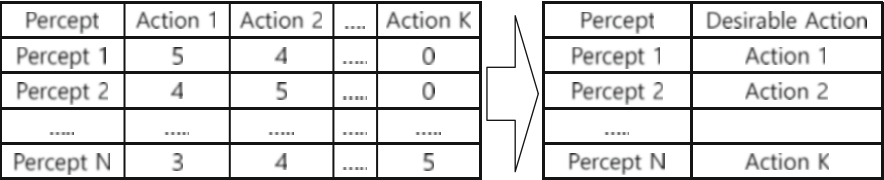


Fig. 16.29 Q tables and training examples for supervised learning

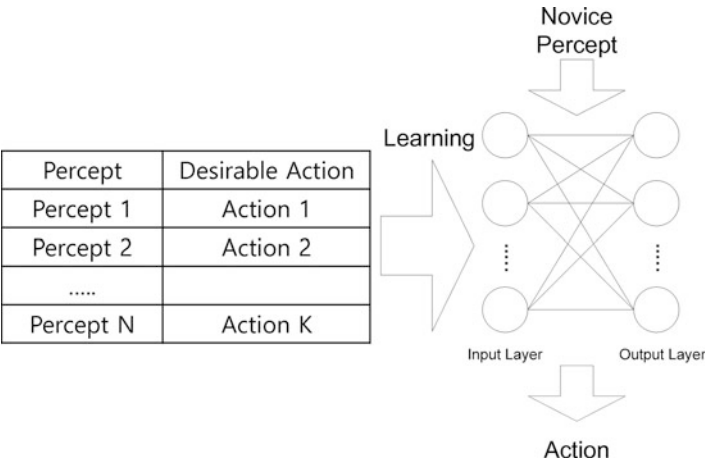


Fig. 16.30 Classification of perception into desirable action

The updated Q table and the training examples for the supervised learning are presented in Fig. 16.29. The left part of Fig. 16.29 is the Q table that is updated through the reinforcement learning. The right part of Fig. 16.29 shows the training examples that are used for training the classifier. The action with the maximum score in the Q table is given as the desirable category for each perception. The process of mapping the entry in the Q table into a labeled training example is presented in Fig. 16.29.

The process of training the neural networks that are given as classifier by the training examples from the Q table, as shown in Fig. 16.30. The mapping of the entries in the Q table into the training examples, each of which consists of the input vector and its desirable label, is presented in Fig. 16.29. The training examples are learned, reaching the capacity for classifying each percept into its desirable action. In the pure reinforcement learning, the desirable action is taken from the most relevant entry that is retrieved from the Q table, whereas the percept is classified into its desirable action by the classifier in this combination. In combining the reinforcement learning with the supervised learning, it is reliable to take the desirable action, but it takes much time for training the classifier as the payment.

Let us make some remarks on the combination of the reinforcement learning and the supervised learning as shown in Fig. 16.28. From the entry of the Q table, the

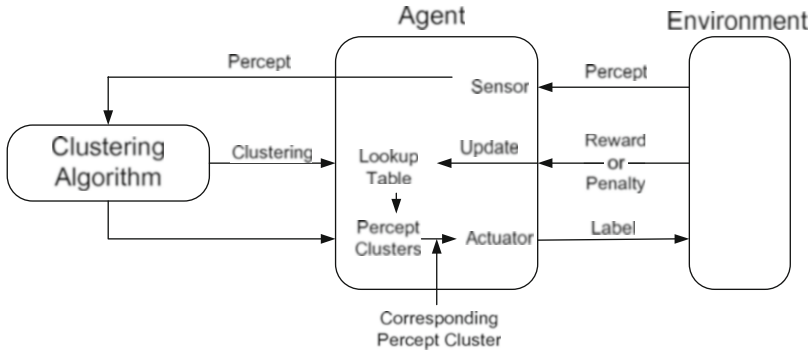


Fig. 16.31 Environment of Reinforcement + Unsupervised

action is extracted in the only reinforcement learning, directly. The entry of the Q table is classified into one among the action in adding the supervised learning. The Q table is updated through the reinforcement learning, and the entries of the updated version are used as the training examples for the supervised learning. When the entry is updated again by the reward and the penalty, the incremented learning is needed for training the updated ones, again.

16.4.3 Reinforcement + Unsupervised

This section is concerned with the combination of the reinforcement learning and the unsupervised learning. We already studied the combination of the reinforcement learning and the supervised learning where an entry of the Q table is classified into the desirable action, in Sect. 16.4.2. The Q table may be segmented into several entry clusters, based on their similarities, in combining the reinforcement learning with the unsupervised learning. In this combination type, we do not consider the desirable actions that are associated with the percepts; they are clustered by their similarities. This section is intended to describe the reinforcement learning that is supported by the clustering algorithm.

The environment, the agent, and the clustering algorithm, in this type of the reinforcement learning, are illustrated in Fig. 16.31. The agent has the receipt for receiving a precept from the external environment and the Q table that is updated by the reinforcement learning. The entries in the Q table are clustered into subgroups, each of which consists of similar percepts, by a clustering algorithm that is added to the system. The action is decided by not individual percept but by a percept cluster in the actuator. The classifier is replaced by a clustering algorithm in the combination of the reinforcement learning with the unsupervised learning.

The perception clusters are illustrated in Fig. 16.32. Each perception is assumed to be a d dimensional vector, and the cosine similarity or the inverse Euclidean

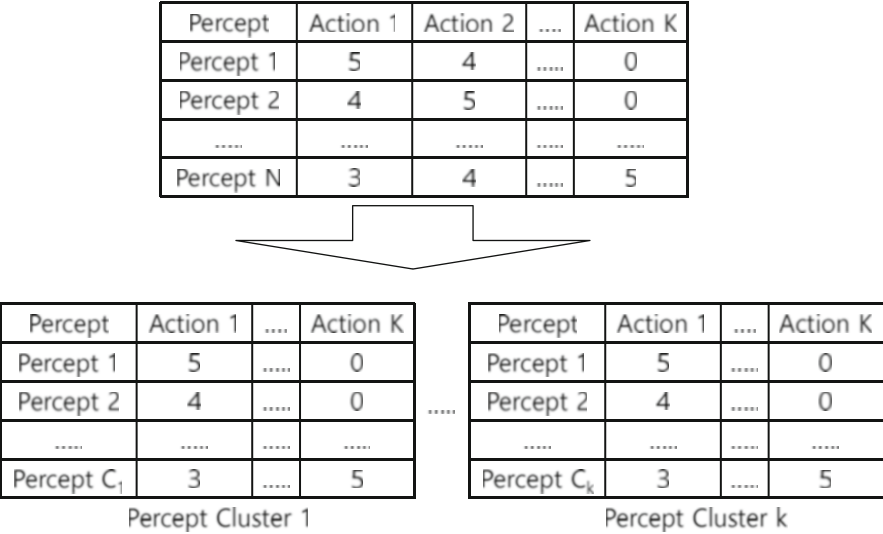


Fig. 16.32 Perception clustering

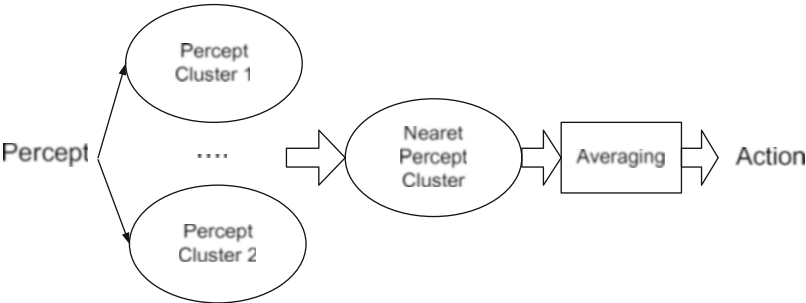


Fig. 16.33 Perception classification

distance is defined as the similarity metric. The Q table is partitioned into subtables, each of which indicates a perception cluster. Each subtable consists of similar perceptions or associated ones. Instead of the entire table, most matching entry is retrieved from a subtable; the retrieval of the matching one is speeded up in combining the reinforcement learning with the unsupervised learning.

The process of deciding the final action from the percept cluster is illustrated in Fig. 16.33. The percept is received by the agent. The entry cluster is decided by the clustering algorithm, and the reward scores of the actions are extracted from the entry cluster. The reward scores are averaged in the entry cluster, and the final action with the maximal averaged score is decided. The combination of the reinforcement learning with the unsupervised learning is intended to use the multiple similar entries in the Q table for making more reliable final decision.

Let us make some remarks on the combination of the reinforcement learning and the unsupervised learning with each other. The role of the reinforcement learning is to update the reward scores to the actions in the Q table. The role of the unsupervised learning is to segment the Q table into subtables, each of which consists of similar percepts. The percept cluster that is given in a subtable is retrieved from the Q table for deciding the desired action. The average over the reward scores or the most relevant percept in the subtable is chosen for deciding the final action.

16.4.4 Environment Prediction

This section is concerned with the reinforcement learning with the prediction of future states. In the reinforcement learning that we have studied until now, it is assumed that the state is available currently. In one that we study in this section, it is assumed that the future state is available as well as the current one. For deciding the action, we use dual Q tables: one that is updated by the current state and the other that is updated by the predicted state. This section is intended to describe the reinforcement learning that considers the predicted future state.

The interaction of the agent with the environment in the reinforcement learning with the next state prediction is illustrated in Fig. 16.34. The percept and the current state are received from the environment. The next state is decided based on the current state, and one more Q table that is updated by the reward and the penalty in the predicted state is constructed. The action is decided by voting the two tables, and the final action and the predicted state are generated. The reason for putting the predicted state into the environment is to get the feedback on the insistency between the predicted next state and the actual next state.

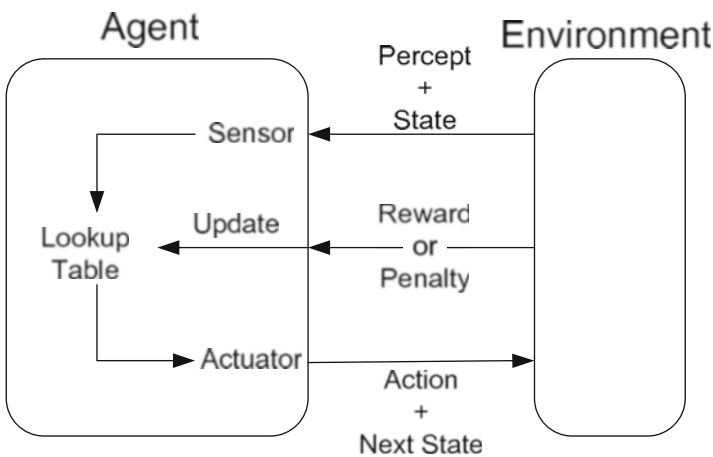


Fig. 16.34 Environment for state prediction

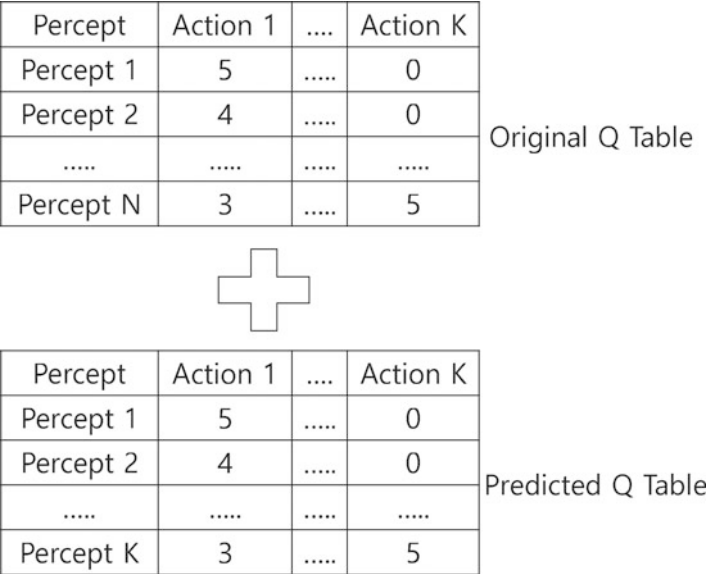


Fig. 16.35 Dual Q tables: original Q table and predicted Q table

```
classifyExample(Table qTable, Table predictedQTable, Item example, real distanceThreshold){
    relevantEntryList1 = selectRelevant(qTable, example, distanceThreshold);
    relevantEntryList2 = selectRelevant(predictedQTable, example, distanceThreshold);
    action1 = decideAction(qTable, relevantEntryList1, example);
    action2 = decideAction(predictedQTable, relevantEntryList2, example);
    action = weightedVote(action1, action2);
    return action;
}
```

Fig. 16.36 Reinforcement learning with dual Q tables

Let us present the dual Q tables illustrated in Fig. 16.35: one in the traditional reinforcement learning and the other that is updated by the predicted state. The predicted state may be different from the real state afterward. The risk of using Q table that is updated by the predicted state exists for deciding the action. In order to reduce the risk, the action is decided by both versions of the Q table. The portion of the second Q table is decided inversely proportionally to the prediction error.

The process of classifying an item in this type of the reinforcement learning is illustrated in Fig. 16.36. The two Q tables are used: one is updated by the traditional reinforcement learning and the other is updated by the predicted state. The entries are retrieved from the two Q tables, and the actions are looked up from the entries. The final action is decided by voting the two actions. The prediction of the next state and the update of the Q table will be considered in the next study.

Let us make some remarks on the reinforcement learning with the predicted state. There are risks in predicting the next state for improving the reinforcement learning. The decision of the desired action may be damaged by misclassifying the state. This

reinforcement learning type depends on the scheme of predicting the next state. In the next study, we consider the prediction scheme of the next state.

16.5 Summary and Further Discussions

Let us summarize entirely what is studied in this chapter. We studied the simple reinforcement learning based on a look-up table in the three tasks: the classification, the regression, and the autonomous navigation. We expanded the reinforcement learning based on the look-up table into one based on the Q table. The reinforcement learning was improved by combining it with the supervised learning and the unsupervised one. In this chapter, we studied the simple reinforcement learning, the Q table based one, and the improved ones with its combination with the other types of learning.

We need to discretize the output value for implementing reinforcement learning. We need to discretize input values in applying the Naive Bayes, the Bayesian Learning, and the decision tree to the classification tasks. We need to define a finite number of actions in the reinforcement learning, so it is required to discretize output values. The number of actions is the number of columns of the Q table; we update the reward score for each action in the Q table, continually. The issue in using the reinforcement learning is how to decide the interval size in discretizing output values.

We may consider multiple Q tables in the reinforcement learning. Combining multiple reinforcement learning algorithms results in multiple Q tables. In using a single reinforcement learning algorithm, multiple Q tables are updated. We considered merge of multiple Q tables into a Q table. We need to maintain multiple Q tables in using agents in different environments.

Let us consider combining the reinforcement learning with the semi-supervised learning. In Sect. 16.4, we combined the reinforcement learning with the supervised learning or the unsupervised learning. The percepts in the Q table are set as labeled examples, and ones beyond it that are generated at random based on the prior knowledge are set as unlabeled examples. We may use the semi-supervised version of the k means algorithm or the Kohonen Networks or the combination of the supervised learning and the unsupervised learning, such as the case of combining the Naive Bayes with the EM algorithm as the semi-supervised learning. The combination of the reinforcement learning with the semi-supervised learning is viewed as the expansion of that of it with the supervised learning.

Let us consider the co-learning in the reinforcement learning. We studied one in the supervised learning in Sect. 14.4.4. The Q table may be updated by exchanging entries between two reinforcement learning agents as the co-learning. It may be expanded into the cellular reinforcement learning where learning agents exist and the Q tables are exchanged between two agents at random. The co-learning in the supervised learning is to exchange training examples, whereas one in the reinforcement learning is to exchange entries of Q tables.

Index

A

- Advanced clustering
 - cluster index (*see* Clustering index)
 - parameter tuning (*see* Parameter tuning)
- Advanced reinforcement learning
 - combination of
 - reinforcement and supervised, 378–380
 - of reinforcement and unsupervised, 380–382
 - ensemble reinforcement learning, 376–378
 - environment prediction, 382–384
- Advanced supervised learning
 - co-learning, 329–331
 - incremental learning, 331–333
 - machine learning algorithms, 325
 - resampling, 326–327
 - virtual training examples, 327–329
- Agglomerative hierarchical clustering (AHC)
 - algorithm, 89
 - clustering results, 195
 - cluster similarity, 192–194
 - combinations of simple clustering algorithms, 320–323
 - fuzzy clustering, 195–197
 - initial/standard version, 194–195
 - machine learning algorithms, 89
 - multiple algorithms, 321
 - and online linear clustering, 322
 - variants, 198–199
- Artificial intelligence, 16–18
- Attribute discrimination, 105–106
- Attribute values, 142, 151
- Autonomous agent, 3, 17, 359, 360, 364, 366, 378

- Autonomous moving process, 366–368
- Auto Regression and Moving Average (ARMA), 7

B

- Bayes classifier
 - Bayes rule, 120–121
 - classification, 123–124
 - Gaussian distribution, 121–122
 - probabilities, 118–119
- Bayesian learning
 - Bayesian Network, 131–133
 - causal relation, 133–135
 - comparison, 136–137
 - learning process, 135–136
- Bayesian Network, 131–133
- Bayes rule, 117, 120–121
- Binary classification, 4, 70–71
- Binary clustering, 7, 8

C

- Cascading, 289–290, 296–298
- Cellular machine learning algorithms, 305
- Cellular models, 290–291, 298–300
- Classification process, 330, 331
- Classification process, SVM
 - generalization, 173–174
 - kernel function, 170–171
 - Lagrange multipliers, 171–173
 - linear classifier, 168–170
- Clustering, 7–8

- Clustering distribution, EM algorithm
 - fuzzy distribution, 247–248
 - Gaussian distribution, 244–245
 - Poisson distribution, 245–247
 - uniform distribution, 242–243
 - Clustering governance
 - cluster maintenance, 274–276
 - cluster naming, 272–274
 - constructed clusters, 261
 - data clustering, 261
 - hard organization, 275, 276
 - integration, multiple clustering results, 278–280
 - multiple viewed clustering, 276–278
 - soft organization, 275, 276
 - Clustering index
 - as clustering evaluation metric, 261
 - computation process, 262–264
 - definition, 261, 262
 - fuzzy clustering evaluation, 265–267
 - hard clustering evaluation, 264–265
 - hierarchical clustering evaluation, 267–268
 - inter-cluster similarity, 262, 263
 - intra-cluster similarity, 262, 263
 - metrics, 261
 - parameter tuning, 261, 262 (*see also* Parameter tuning)
 - Clustering process
 - E-step, 250–252
 - initializing clustering in EM algorithm, 249–250
 - issues in using EM algorithm, 253–254
 - M-step, 252–253
 - Cluster-item matrix, 241, 256, 257, 259, 324
 - Cluster membership value
 - EM algorithm, 245, 246, 250–253, 255–259
 - Cluster naming, 272–274
 - Codification, 152
 - Co-learning, 316, 329–331
 - Color image matrix, 64
 - Combined learning algorithms
 - combination paradigms
 - cascading, 319, 320
 - expert gate, 318, 319
 - intra-clustering index, 318, 320
 - unsupervised learning algorithms, 318
 - voting, 318, 319
 - EM algorithm, 324–325
 - K means algorithm, 323–324
 - KNN algorithm, 323–324
 - Naive Bayes, 324–325
 - simple clustering algorithms, 320–323
 - Concentric Nearest Neighbor (CNN), 108–109
 - Conditional probability, 117
 - Cost function, 71
- D**
- Database Management System (DBMS), 50, 51
 - Data clustering, 4
 - Data encoding
 - definition, 47
 - image data, 47–48
 - numerical vectors, 48
 - relational data, 47
 - textual data, 47
 - Data mining, 19
 - Decision graph, 162–164
 - Decision list, 158–160
 - Decision tree
 - attributes, 141
 - branches, 141
 - classification process, 141
 - basic structure, 142–144
 - binary classification, 144, 145
 - multiple classification, 145–146
 - root node, 146
 - rule extraction, 148–150
 - text, 147–148
 - learning process, 141
 - interior nodes, 154–155
 - preprocessing, 151–152
 - pruning, 155–156
 - root node, 152–154
 - root node, 141
 - symbolic learning algorithm, 142
 - validation set, 141
 - variants
 - decision graph, 162–164
 - decision list, 158–160
 - random forest, 160–162
 - regression version, 156–158
 - Decomposition, 74, 75
 - Deep learning algorithms, 305
 - Denormalization, 52
 - Diagonal matrix, 30
 - Discrete Markov model
 - state diagram, 336–337
 - state path probability, 338–340
 - state transition probabilities, 337–338
 - time series prediction, 340–341
 - Discretization, 74

Divisive clustering algorithm
 binary clustering, 200–202
 clusters, 191
 definition, 191
 evolutionary binary clustering, 202–204
 standard version, 204–205
 variants, 205–207
 Dual learning process, 330
 Dynamic Data Organization (DDO), 222–224
 Dynamic Document Organization (DDO)
 system, 326, 327

E

Edges, 142, 143
 Eigen values, 40–41, 61
 Eigen vectors, 40–41, 60
 Encoding process, 52–53
 Ensemble learning
 combination schemes
 cascading, 289–290
 cellular models, 290–291
 expert gates, 287–289
 voting, 286–287
 definition, 285
 distributed processing, 286
 heterogeneous combination, 306
 homogenous combination, 306
 meta-learning, 285 (*see* Meta-learning)
 multiple machine learning algorithms, 285
 neural networks, 306
 parallel processing, 286
 partitions, 285
 semi-supervised learning, 306
 unsupervised learning algorithms, 306
 Exchangeable Image File Format (EXIF), 63
 Expectation–Maximization (EM) algorithm
 cluster distribution (*see* Clustering
 distribution, EM algorithm)
 clustering process
 E-step, 250–252
 initialization, 249–250
 issues, 253–254
 M-step, 252–253
 cluster-item matrix, 241
 E-step, 241
 membership, 241–242
 M-step, 241
 semi-supervised learning
 classification performance, 254
 Gaussian distribution, 254–255
 initialization, 255–256
 likelihood estimation, 256–257
 machine learning algorithm, 254

Naive Bayes, 255
 parameter estimation, 257–259
 regression performance, 254
 Expert gates, 287–289, 294–296

F

Feature extraction, 58
 Feature selection, 59
 Feature value assignment, 60
 Fuzzy clustering
 AHC algorithm, 195–197
 binary clustering, 200
 clustering index, 265–267
 clustering process, 197
 EM algorithm, 247–248
 K means algorithm, 227–229
 online linear clustering algorithm, 210–212
 Fuzzy SVM, 181–182

G

Gaussian distribution, 121–122, 327
 Gaussian-Jordan elimination process, 37
 Generalization, 3
 Gray scaled image matrix, 63

H

Hidden Markov model (HMM)
 discrete Markov model, 341
 HMM learning, 349–351
 initial parameters, 342–343
 observation sequence probability, 343–345
 state sequence estimation, 346–349
 text topic analysis, 351, 358 (*see also*
 Temporal text topic analysis)
 Hierarchical clustering
 clustering index, 267–268
 K means algorithm, 229–230
 Hybrid task, 8–10
 Hyperplane, 79–80, 331, 332

I

Image data, 47–48
 encoding issues, 66
 encoding process, 65–66
 file formats, 63
 matrix, 63–65
 Image File Directory (IFD), 63
 Image file formats, 63
 Image File Header (IFH), 63
 Image matrix, 63–65
 Image pixel, 64

Incremental learning, 331–333

Inner product, 26–28

Instance based learning

definition, 93

example similarity, 98–99

look-up example, 94–95

1-NN, 99–100

rule based approach, 95–98

similarity matrix, 93

training set, 93

Interior nodes, 154–155

Inverse matrix, 35–37

J

Joint Photographic Expert Group (JPEG), 63

JPEG File Interchange Format (JFIF), 63

K

Kernel function

definition, 170

inner product of two vectors in the feature space, 170

mathematical characterization, 170–171

string kernel, 171

in SVM, 170

KKT (Karush–Kuhn–Tucker) condition, 177–179, 186

K means algorithm

clustering algorithm, 217

clustering process

fuzzy clustering, 227–229

hard clustering, 225–227

hierarchical clustering, 229–230

initialization, 224–225

clustering tool, 217

learning process, 217

supervised and unsupervised learning

DDO, 222–224

learning paradigm transition, 218–219

semi-supervised KNN, 221–222

unsupervised KNN, 219–221

variants (*see* Variants)

K medoid algorithm, 217, 230–233

KNN (K Nearest Neighbor) algorithm., 11

attribute discriminations, 105–106

nearest neighbors, 101–103

notations, 100–101

variants (*see* Variants)

voting, 103–105

Kohonen Networks

initial version

architecture, 311

learning process, 311, 312

SOM, 312, 313

unsupervised learning, 310, 311

weight vectors, 310, 311

vs. K means algorithm, 317–318

LVQ, 313–315

semi-supervised version, 315–316

L

Lagrange multipliers

general SVM equation, 173

as learning process, optimization process

dual problem, 176–177

genetic algorithm, 180

hill climbing, 180

primal problem, 174–176

simulated annealing, 180

SMO algorithm, 177–179

support vectors, 180

linear classifier, 171

optimization of weights, 172

support vectors, 172

training examples, 171

Learning Vector Quantization (LVQ), 313–315

Least square SVM, 183–185

Linear classifiers

binary classification, 86

classification task, 86

definition, 168

hyperplane, 86, 87

hyperplane equation, 84–86

linear boundary, 169

as linear equation, 167

linear separability, 168, 169

multiple classification, 87

perceptron, 87–88

regression, 86

separability, 82–84

training examples, 169–170

Linear equation, 35, 79

Linear independence, 28–29

Linear separability, 82–84

Line equation, 79

M

Machine learning, 298

application

classification, 4–6

clustering, 7–8

regression, 6–7

artificial intelligence, 16–18

data mining, 19

- definition, 3–4
- hybrid task, 8–10
- neural networks, 16, 18–19
- optimization tasks, 16
- soft computing, 20–21
- types
 - reinforcement learning, 15–16
 - semi-supervised learning, 14–15
 - supervised learning, 11–12
 - unsupervised learning, 12–13
- Mean vector matching, 80, 81
- Meta-learning, 285
 - cascading, 296–298
 - cellular model, 292, 298–300
 - expert gate, 294–296
 - voting, 292–294
- Multiple classification, 5, 71–72
- Multiple clustering, 7, 8
- Multiple Layer Perceptron (MLP), 18, 87, 303–304, 327
- Multiplication, 32–34

N

- Naive Bayes, 255, 324–325
 - application to text classification, 129–131
 - classification, 124–126
 - learning process, 126–127
 - variants, 128–129
- Neural networks, 16, 18–19, 87, 89
- Neural Text Categorizer (NTC), 18
- Neural Text Self Organizer (NTSO), 18
- Nodes, 142, 143
- Normal distribution
 - Bayes classifier, 121
 - EM algorithm, 241–244
- Numerical vectors, 61, 65
 - definition, 23
 - dimension, 23
 - machine learning algorithms, 24
 - matrix, 23, 24
 - operations
 - binary, 25
 - definition, 24–25
 - inner product, 26–28
 - linear independence, 28–29
 - norm, 26
 - scalar multiplication, 26
 - scalar value, 25
 - vectors, 25, 26
 - operations on matrices
 - addition and deletion, 31
 - definition, 30–31
 - inverse matrix, 35–37

- multiplication, 32–34
 - scalar multiplication, 32
 - scalar multiplication, 23
 - vector and matrix
 - determinant, 38–39
 - Eigen value, 40–41
 - Eigen vector, 40–41
 - PCA, 43–45
 - SVD, 42–43

O

- One Nearest Neighbor (1-NN), 21, 80, 81, 99–100, 363, 365, 371
- Online linear clustering algorithm
 - fuzzy clustering, 210–212
 - initial stage, 209
 - representative selection scheme, 207
 - simple and fast, 192
 - variants, 212–214
- Overfitting, 77

P

- Pairwise SVM, 182–183
- Parameter tuning
 - AHC algorithm, 199
 - clustering index to unlabeled items, 268–269
 - EM algorithm, 253
 - evolutionary clustering, 271–272
 - k means algorithm, 270–271
 - simple clustering algorithms, 269–270
- Parametric classification algorithm, 121
- Partition
 - architecture, 303–304
 - attribute set, 302–303
 - distributed learning, 304–305
 - parallel learning, 304–305
 - training set, 300–302
- Perceptron
 - architecture, 87
 - classification, 381
 - clustering, 381
 - learning, 88
 - linear classifier, 87
 - vs. LVQ, 314
 - MLP, 18, 184, 327
 - neural networks, 18
 - SVM (*see* Support vector machine (SVM))
- Plane equation, 80
- Portable Network Graphics (PNG), 63
- Primitive SVM, 168
- Principal Component Analysis (PCA), 43–45, 65

Probabilistic learning

- Bayes classifier (*see* Bayes classifier)
- Bayesian learning (*see* Bayesian learning)
- categories, 117
- conditional probability, 117
- definition, 117
- likelihood, 117–118
- machine learning algorithms, 117, 118
- Naive Bayes (*see* Naive Bayes)
- posterior, 117
- supervised learning algorithms, 118

Problem decomposition, 74–76

Pseudo inverse matrix, 35

Q

Q learning

- finite state, 370–371
- infinite state, 371–374
- Q table, 368–370
- regression, 372
- stochastic reward, 374–375

Q table

- actions, 370
- definition, 368
- final version, 369
- initialization, 369
- module, 359–360
- and reinforcement learning, 359
- update, 369

R

Radius Nearest Neighbor (RNN), 108–110, 120

Random forest, 160–162

Random variable, 119

Red, Green, and Blue (RGB), 64

Regression, 3, 6–7, 53, 73, 156–158

Reinforcement learning, 15–16. *See also*

- Advanced reinforcement learning
- classification task, 362–364
- definition, 359
- Q table, 359
- regression task, 364–366
- simple autonomous moving, 366–368
- single example, 360–362

Relational data, 47

- database, 50–51
- data records, 48, 49
- definition, 48
- encoding issues
 - different field names, 54, 55
 - missing values, 54

numerical vectors, 53

unreliable values, 54

encoding process, 52–53

fields, 48

index records, 48, 49

primary key, 48, 49

table, 50

Resolution, 63

Root node, 152–154

Rule based approach, 81, 82

S

Self-Organizing Map (SOM), 18, 310, 312, 313

Semi-supervised learning, 12, 14–15, 298

clusters, 333

constraint clustering, 309

definition, 309

EM algorithm

classification performance, 254

Gaussian distribution, 254–255

initialization, 255–256

likelihood estimation, 256–257

machine learning algorithm, 254

Naive Bayes, 255

parameter estimation, 257–259

regression performance, 254

initial clusters, 309

K means algorithm, 221–222, 234–236, 310, 333

Kohonen Networks, 310

sets of training, 309

virtual training, 310

Semi-supervised version, 315–316

Sensors, 359, 362, 364–366

Set of events, 119

Simple classification algorithms

hyperplane, 79–80

machine learning, 80–82

rectangle, 77–78

threshold rule, 76–77

Simple clustering algorithms

bottom-up, 191 (*see also* Agglomerative hierarchical clustering (AHC) algorithm)

data clustering, 191

fast, 191

top-down, 191 (*see also* Divisive clustering algorithm)

Simple machine learning algorithms

binary classification, 69, 70

classification

binary, 70–71

- multiple, 71–72
 - problem decomposition, 74–76
 - regression, 73
 - scalar values, 69
 - set of training, 69
 - two dimensional numerical vector, 69
 - Singular Value Decomposition (SVD), 42–43, 60, 65
 - Soft computing, 20–21
 - Spans, 28
 - Sparse SVM, 185–186
 - Stemming rule, 56, 57
 - Supervised learning, 3, 11–12
 - Support vector machine (SVM), 18, 83, 87
 - classification process (*see* Classification process, SVM)
 - definition, 167
 - dual space, 167
 - EM algorithm, 187
 - kernel function, 168
 - learning process
 - dual problem, 176–177
 - primal problem, 174–176
 - SMO algorithm, 177–179
 - linear classifier, 167
 - linear separability, 167
 - surveying approaches, 186
 - variants
 - fuzzy SVM, 181–182
 - LMS SVM, 183–185
 - pairwise SVM, 182–183
 - sparse SVM, 185–186
 - Symmetry matrix, 30
- T**
- Tagged Image File Format (TIFF), 63
 - Temporal learning
 - conditional probabilities of states, 335–336
 - definition, 335
 - discrete Markov model (*see* Discrete Markov model)
 - HMM (*see* Hidden Markov model (HMM))
 - parameters, 335
 - Temporal text topic analysis
 - definition, 351
 - keyword extraction from paragraph, 353
 - learning, 354–356
 - paragraph categorization, 353
 - sampling, 353–354
 - task specification, 351–353
 - topic sequence, 356–357
 - Term Frequency-Inverse Document Frequency (TF-IDF), 58
 - Text classification, 62
 - Text clustering, 62
 - Text encoding, 58–60
 - Text indexing, 55–57
 - Textual data, 47
 - dimension reduction, 60–61
 - encoding, 58–60
 - encoding issues, 61–62
 - indexing, 55–57
 - Threshold rule, 76–77
 - Tokenization, 56
 - Traditional machine learning, 276, 333–334
 - Traditional reinforcement learning, 383
 - Traditional SVM, 183–186
 - Trapezoid distribution, 248
- U**
- Unsupervised learning, 3, 7, 12–13
- V**
- AHC algorithm, 198–199
 - divisive clustering algorithm, 205–207
 - K means algorithm
 - constraint clustering, 236–239
 - dynamic K means algorithm, 233–234
 - k medoid algorithm, 230–233
 - semi-supervised version, 234–236
 - KNN algorithm
 - CNN, 108–109
 - dynamic nearest neighbor, 106–108
 - hierarchical nearest neighbor, 110–112
 - hub example, 112–114
 - RNN, 108–109
 - online linear clustering algorithm, 212–214
 - Vector, 30
 - Voting, 286–287, 292–294