Check for updates

Machine Learning Overview

2

2.1 Overview

Learning denotes the process of acquiring new declarative knowledge, the organization of new knowledge into general yet effective representations, and the discovery of new facts and theories through observation and experimentation. Learning is one of the most important skills that mankind can master, which also renders us different from the other animals on this planet. To provide an example, according to our past experiences, we know the sun rises from the east and falls to the west; the moon rotates around the earth; 1 year has 365 days, which are all knowledge we derive from our past life experiences.

As computers become available, mankind has been striving very hard to implant such skills into computers. For the knowledge which are clear for mankind, they can be explicitly represented in program as a set of simple reasoning rules. Meanwhile, in the past couple of decades, an extremely large amount of data is being generated in various areas, including the World Wide Web (WWW), telecommunication, climate, medical science, transportation, etc. For these applications, the knowledge to be detected from such massive data can be very complex that can hardly be represented with any explicit fine-detailed specification about what these patterns are like. Solving such a problem has been, and still remains, one of the most challenging and fascinating long-range goals of machine learning.

Machine learning is one of the disciplines, which aims at endowing programs with the ability to learn and adapt. In machine learning, experiences are usually represented as data, and the main objective of machine learning is to derive models from data that can capture the complicated hidden patterns. With these learned models, when we feed them with new data, the models will provide us with the inference results matching the captured patterns. Generally, to test the effectiveness of these learned models, different evaluation metrics can be applied to measure the performance of the inference results.

Existing machine learning tasks have become very diverse, but based on the supervision/label information used in the model building, they can be generally categorized into two types: "supervised learning" and "unsupervised learning." In supervised learning, the data used to train models are pre-labeled in advance, where the labels indicate the categories of different data instances. The representative examples of supervised learning task include "classification" and "regression." Meanwhile, in unsupervised learning, no label information is needed when building the models, and the representative example of unsupervised learning task is "clustering." Between unsupervised

© Springer Nature Switzerland AG 2019

J. Zhang, P. S. Yu, *Broad Learning Through Fusions*,
https://doi.org/10.1007/978-3-030-12528-8_2

jwzhanggy@gmail.com

learning and supervised learning, there also exists another type of learning tasks actually, which is named as the "semi-supervised learning." Semi-supervised learning is a class of learning tasks and techniques that make use of both labeled and unlabeled data for training, typically a small amount of labeled data with a large amount of unlabeled data. Meanwhile, besides these aforementioned learning tasks, there also exist many other categorizations of the learning tasks, like "transfer learning," "sparse learning," "reinforcement learning," and "ensemble learning."

To make this book self-contained, the goal of this chapter is to provide a rigorous, yet easy to follow, introduction to the main concepts underlying machine learning, including the detailed data representations, supervised learning and unsupervised learning tasks and models, and several classic evaluation metrics. Considering the popularity of deep learning [14] in recent years, we will also provide a brief introduction to deep learning models in this chapter. Many other learn tasks, like semi-supervised learning [7, 57], transfer learning [31], sparse learning [28], etc., will be introduced in the following chapters when discussing the specific research problems in detail.

2.2 Data Overview

Data is a physical representation of information, from which useful knowledge can be effectively discovered by machine learning and data mining models. A good data input can be the key to the discovery of useful knowledge. In this section, we will introduce some background knowledge about data, including data types, data quality, data transformation and processing, and data proximity measures, respectively.

2.2.1 Data Types

A data set refers to a collection of data instances, where each data instance denotes the description of a concrete information entity. In the real scenarios, the data instances are usually represented by a number of attributes capturing the basic characteristics of the corresponding information entity. For instance, let's assume we see a group of Asian and African elephants (i.e., elephant will be the information entity). As shown in Table 2.1, each elephant in the group is of certain weight, height, skin smoothness, and body shape (i.e., different attributes), which can be represented by these attributes as an individual data instance. Generally, as shown in Fig. 2.1, the mature Asian elephant is in a smaller size compared with the mature African elephant. African elephants are discernibly larger in size, about 8.2–13 ft (2.5–4 m) tall at the shoulder, and they weigh between 5000 and 14,000 lbs (2.5–7 t). Meanwhile, Asian elephants are more diminutive, measuring about 6.6–9.8 ft (2–3 m) tall at the shoulder and weighing between 4500 and 12,000 lbs (2.25–6 t). In addition, from their ear size,

Table 2.1 An example of the elephant data set (ear size: 1 denotes large; 0 denotes small)

Elephant ID	Weight (t)	Height (m)	Skin	Ear size	Trunk "finger"	···	Category
1	6.8	4.0	Wrinkled	1	2	···	African
2	5.8	3.5	Wrinkled	1	2	···	African
3	4.5	2.1	Smooth	0	1	···	Asian
4	5.8	3.1	Wrinkled	0	1	···	Asian
5	4.8	2.7	Wrinkled	1	2	···	African
6	5.6	2.8	Smooth	1	1	···	Asian
···	···	···	···	···	···	···	···

jwzhanggy@gmail.com

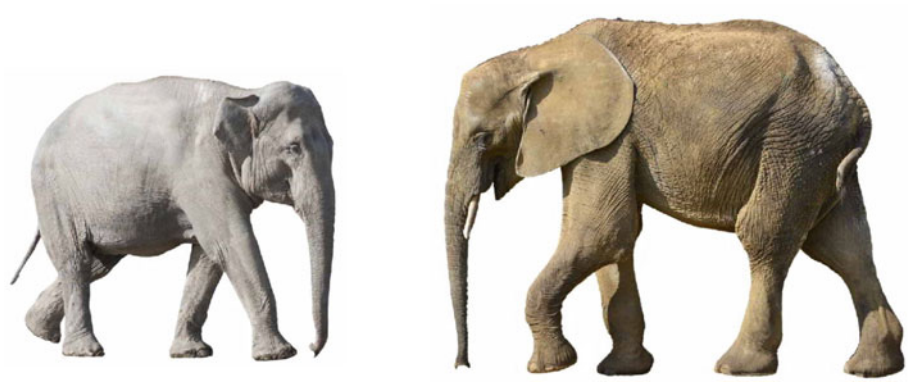

Fig. 2.1 A comparison of Asian elephant vs African elephant

skin smoothness, and trunk "finger" number, etc., we can also effectively differentiate them from each other. The group of the elephant data instances will form an elephant data set, as illustrated in Table 2.1.

Instance-attribute style data format is a general way for data representation. For different types of data sets, the attributes used to describe the information entities will be different. In the following parts of this subsection, we will talk about the categories of attributes first, and then introduce different data types briefly.

2.2.1.1 Attribute Types

Attribute is the basic element in describing information entities, and we provide its formal definition as follows.

Definition 2.1 (Attribute) Formally, an attribute denotes a basic property or characteristic of an information entity. Generally, attribute values can vary from one information entities to another in a provided data set.

In the example provided in Table 2.1, the elephant body weight, height, skin smoothness, ear size, and trunk "finger" number are all the attributes of the elephant information entities. Among these attributes, both body weight and height are the attributes with continuous values, which can take values in a reasonable range (e.g., [1 t, 7 t] for weight and [1 m, 5 m] for height, respectively); trunk "finger" number is a discrete attribute instead, which takes values from set {1, 2}; ear size is a transformed attribute, which maps the ear size into 2 levels (1: large; 0: small); and skin smoothness is an attribute with categorical values from set {Wrinkled, Smooth}. These attributes listed above are all the facts about elephants.

In Table 2.1, we use integers to denote the elephant ID, ear size, and trunk "finger" number. However, for the same number appearing in these three kinds of attributes, they will have different physical meanings. For instance, the elephant ID 1 of the 1st row in the table denotes the unique identifier of the elephant in the table; meanwhile, the ear size 1 of rows 1, 2, 5, and 6 in the table denotes the elephants have a large ear size; and the trunk "finger" number 1 of rows 3, 4, and 6 denotes the count of trunk fingers about the elephant instances. Therefore, to interpret these data attributes, we need to know their specific attribute types and the corresponding physical meanings in advance.

jwzhanggy@gmail.com

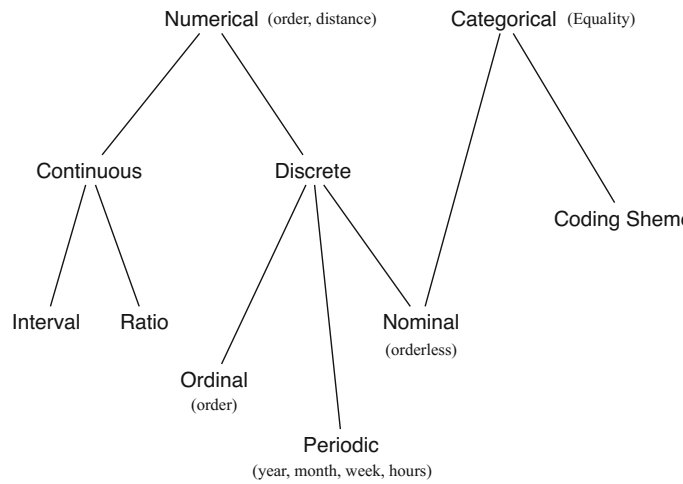

Fig. 2.2 A systematic categorization of attribute types

As shown in Fig. 2.2, the attribute types used in information entity description can usually be divided into two main categories, *categorical attributes* and *numeric attributes*. For instance, in Table 2.1, skin smoothness is a *categorical attribute* and weight is a *numeric attribute*. As to the ear size, originally this attribute is a *categorical attribute*, where the values have inherent orders in terms of the elephant ear size. Meanwhile, the transformed attribute (i.e., the integer numbers) becomes a numerical attribute instead, where the numbers display such order relationships specifically. We will talk about the attribute transformation later.

Generally, the *categorical attributes* are the qualitative attributes, while the *numeric attributes* are the quantitative attributes, both of which can be further divided into several sub-categories as well. For instance, depending on whether the attributes have order relationships or not, we can divide the *categorical attributes* into *nominal attributes* and *ordinal attributes*. On the other hand, depending on the continuity of attribute values, the *numeric attributes* can be further divided into *discrete attributes* and *continuous attributes*. Detailed descriptions about these above attribute categories are provided as follows.

- **Nominal Categorical Attribute**: Nominal categorical attributes provide enough information to distinguish one data instance from another, which don't have any internal relationships. For instance, for the elephant instances described in Table 2.1, the involved *nominal categorical attributes* include *elephant category* ({Asian, African}) and *elephant skin smoothness* ({Wrinkled, Smooth}). Besides this example, some other representative *nominal categorical attributes* include *colors* ({Red, Green, . . ., Purple}), *names* (people name: {Alice, Bob, . . ., Zack}, country name: {Afghanistan, Belgium, China, . . ., Zambia}), *ID numbers* ({1, 2, . . ., 99} or {id001, id002, . . ., id099}).
- **Ordinal Categorical Attributes**: Similar to the nominal categorical attributes, the ordinal categorical attributes also provide enough information to distinguish one object from another. Furthermore, the ordinal attribute values also bear order information. Representative examples of *ordinal categorical attributes* include *elephant ear size* ({Small, Large}), *goodness* ({Good, Better, Best}), *sweetness* ({Sour, Slightly Sour, . . ., Slightly Sweet, Sweet}), grades ({A, B, C, D, F}).
- **Discrete Numeric Attributes**: Discrete numeric attribute is one type of numeric attribute from a finite or countably infinite set of values. Representative examples of *discrete numeric attributes* include the attributes about *counts* (e.g., elephant trunk "finger" number, population of countries, number of employees in companies, number of days).

jwzhanggy@gmail.com

- **Continuous Numeric Attributes**: Continuous numeric attributes have real numbers as the attribute values. Representative examples include temperature, mass, size (like length, width, and height). Continuous numeric attributes are usually denoted as float-point real numbers in the concrete representations.

2.2.1.2 Types of Data Sets

Data can be collected from different disciplines, and the original storage representations of the data can be very diverse. The data storage types depend on various factors in data gathering, like the equipment applied to gather the data, the preprocessing and cleaning operations, the storage device format requirements, and specific application domains. To help provide a big picture about data set types used in this book, we propose to categorize data into *record data*, *graph data*, and *ordered data*. In this part, we will provide a brief introduction of these data types together with some illustrative examples.

- **Record Data**: Many data sets can be represented as a set of independent data records described by certain pre-specified attributes, like the elephant data set shown in Table 2.1. In the record style data sets, the records are mostly independent of each other with no dependence or correlation relationships among them. Besides the fixed attributes, for many other cases, the record attributes can be dynamic and may change differently from one record to another. A representative example of such a kind of data set is the "Market Transaction Data" (as illustrated in plot a of Fig. 2.3), which is a set of market shopping transaction records. Each record denotes the purchased items in the transaction, which may vary significantly for different transactions. In addition, in Fig. 2.3,

A

TID	ITEMS
1	Bread, Butter, Milk
2	Beer, Diaper
3	Beer, Diaper, Bread, Milk
4	Soda, Diaper, Milk

B

AID	Age	Has_Job	Own_House	Credit	Approval
1	Young	TRUE	FALSE	Good	Yes
2	Young	FALSE	FALSE	Fair	No
3	Middle	TRUE	TRUE	Excellent	Yes
4	Senior	FALSE	TRUE	Good	Yes

C

	computer	software	linux	knuth	love	mac	program	windows
Document 1	3	2	0	5	1	0	3	5
Document 2	0	0	1	3	2	0	2	0
Document 3	0	5	4	6	8	0	6	0
Document 4	7	2	7	0	0	5	0	4

Fig. 2.3 Examples of record data ((**a**) shopping transaction record; **(b)** loan application record; **(c)** document-word representation)

jwzhanggy@gmail.com

we also show two other examples of the record data sets, where one is about the loan application record and the other one is about the document-word representation table. In the loan application record, besides the application IDs, the involved attributes include "*Age*," "*Has Job*," "*Own House*," "*Credit*," and "*Approval*," where the first four attributes are about applicant profile and the last attribute is about the application decision. For the document-word representation table, the words are treated as the attributes and the numbers in the table denote the number of appearance of the words in certain documents.

- **Graph Data**: For the information entities with close correlations, graph will be a powerful and convenient representation for data, where the nodes can represent the information entities while the links indicate the relationships. Generally, in graphs, the links among the information entities may convey very important information. In some cases, the link can also be directed, weighted, or signed, which will indicate the direction, weight and polarity of the relationships among the information entities. As shown in Fig. 2.4, representative examples of graph data include: (1) Online social networks (OSNs), where the nodes denote users and the links represent the friendship/follow links among the users; (2) Chemical molecule graphs, where the nodes denote the atoms and the links represent the bonds among the atoms; and (3) World Wide Web (WWW), where the nodes represent the webpages while the links denote the hyperlinks among the webpages. Graph style data can be represented in the Instance-Attribute format as well, where both the instances and

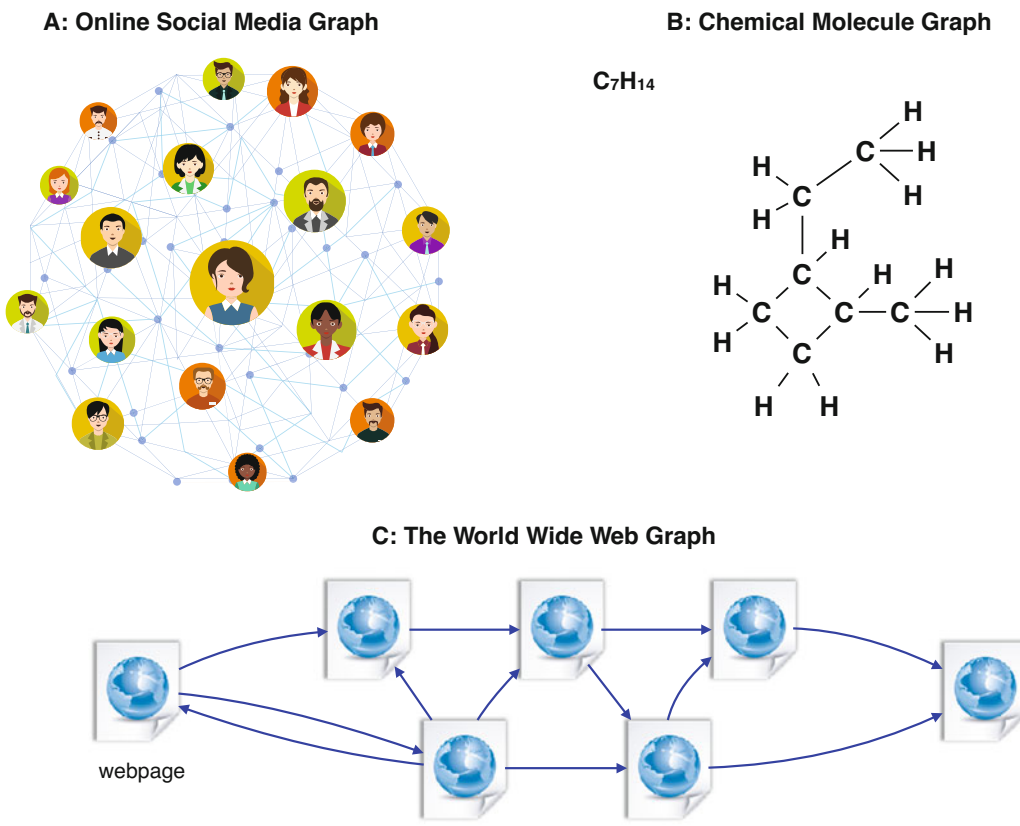

Fig. 2.4 Examples of graph data (**(a)** online social media graph; **(b)** chemical molecule graph; **(c)** the internet webpage graph)

jwzhanggy@gmail.com

attributes correspond to the information entities in the graphs, respectively. Such a representation will be very similar to the graph adjacency matrix to be introduced in Sect. 3.2.1, whose entries will have value 1 (or a signed weight) if they correspond to the connected information entities and 0 otherwise.

- **Ordered Data**: For the other data types, the instances or the attributes may be ordered in terms of time or space, and such a kind of relationship cannot be captured by either the record data or the graph data types, which will be represented as the ordered data instead. Representative examples of the ordered data include the *sequential transaction data*, *sequence data*, *text data*, *time series data*, and *spatial data*, some of which are shown in Fig. 2.5. In the real world, people tend to shop at the same market for multiple times (like customers C_1 and C_2 in plot a of Fig. 2.5), whose purchase records will have a temporal order relationship with each other, e.g., I_4 and I_5 are always purchased after I_1. For the gene data from biology/bioinformatics, as shown in plot c of Fig. 2.5, it can be represented as a sequence composed by A, T, G, and C (e.g., "GGTTCCTGCTCAAGGCCCGAA"), which defines the code of both human and animals. Data accumulated from the observations about nature or finance like temperature, air pressure, stock prices, and trading volumes can be represented as the ordered time series data (e.g., the Dow Jones Industrial Index as shown in plot b of Fig. 2.5). For the data collected from the offline world, like crime rate, traffic, as well as the weather observations, they will have some relationships in terms of their spatial locations, which can be represented as the ordered data as well.

A

Time	Customer	Item
T1	C1	{I1, I2}
T2	C2	{I1, I3}
T3	C1	{I4, I5}
T4	C2	{I2, I4, I5}

B

C

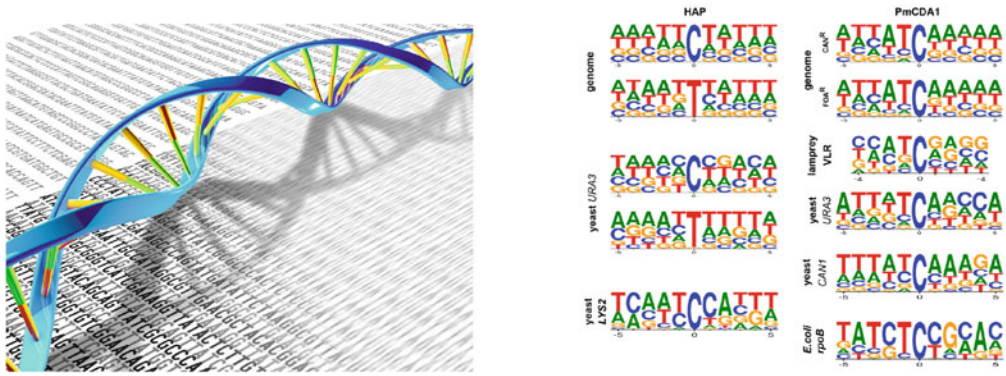

Fig. 2.5 Examples of ordered data (**(a)** sequential transaction record; **(b)** monthly Dow Jones industrial average; **(c)** DNA double helix and interpreted sequence data)

jwzhanggy@gmail.com

2.2.2 Data Characteristics

For machine learning tasks, several characteristics of the data sets can be very important and may affect the learning performance greatly, which include *quality*, *dimensionality*, and *sparsity*.

- **Quality**: Few data sets are perfect and real-world data sets will contain errors in the collection, processing, and storage process due to various reasons, like human errors, flaws in data processing, and limitations of devices. The data errors can also be categorized into various types, which include noise, artifacts, outlier data instances, missing value in data representation, inconsistent attribute values, and duplicate data instances. "*Garbage in, Garbage out*" is a common rule in machine learning. All these data errors will degrade the data set quality a lot, and may inevitably affect the learning performance.
- **Dimensionality**: Formally, the dimensionality of a data set denotes the number of attributes involved in describing each data instance. Data set dimensionality depends on both the data set and the data processing methods. Generally, data sets of a larger dimensionality will be much more challenging to handle, which is also referred to as the "*Curse of Dimensionality*" [2, 3, 52]. To overcome such a challenge, data processing techniques like dimension reduction [51] and feature selection [15] have been proposed to reduce the data dimensionality effectively by either projecting the data instances to a low-dimensional space or selecting a small number of attributes to describe the data instances.
- **Sparsity**: In many cases, in the Instance-Attribute data representations, a large number of the entries will have zero values. Such an observation is very common in application scenarios with a large attribute pool but only a small number of the attributes are effective in describing each data instance. Information sparsity is a common problem for many data types, like record data (e.g., transaction data), graph data (e.g., social network data and WWW data), and ordered data (e.g., text data and sequential transaction data). Great challenges exist in handling the data set with a large sparsity, which has very little information available for learning and model building. A category of learning task named "*sparse learning*" [28] has been proposed to handle such a problem, and we will introduce it in Sect. 7.6.

Before carrying out the machine learning tasks, these aforementioned characteristics need to be analyzed on the data set in advance. For the data sets which cannot meet the requirements or assumptions of certain machine learning algorithms, necessary data transformation and pre-processing can be indispensable, which will be introduced in the following subsection in detail.

2.2.3 Data Pre-processing and Transformation

To make the data sets more suitable for certain learning tasks and algorithms, several different common data pre-processing and transformation operations will be introduced in this part. To be more specific, the data operations to be introduced in this subsection include *data cleaning and pre-processing* [49], *data aggregation and sampling* [49], *data dimensionality reduction and feature selection* [15, 51], and *data transformation* [49].

2.2.3.1 Data Cleaning and Pre-processing

Data cleaning and pre-processing [49] focus on improving the quality of the data sets to make them more suitable for certain machine learning tasks. As introduced in the previous subsection, regular errors that will degrade the data quality include *noise*, *outliers*, *missing values*, *inconsistent values*,

jwzhanggy@gmail.com

and *duplicate data*. Many data cleaning and pre-processing techniques have been proposed to address these problems to improve the data quality.

Data noise denotes the random factor that will distort the data instance attribute values or the addition of spurious instances. Noise is actually very hard to be distinguished from non-noise data, which are normally mixed together. In the real-world learning tasks, noise is extremely challenging to detect, measure, and eliminate. Existing works on handling noise mainly focus on improving the learning algorithms to make them robust enough to handle the noise. Noise is very common in ordered data, like time series data and geo-spatial data, where redundant noisy signals can be gathered in data collection due to the problems with the device bandwidth or data collection techniques.

Outliers denote the data instances that have unique characteristics, which are different from the majority of normal data instances in terms of the instance itself or only certain attributes. Outlier detection has been a key research problem in many areas, like fraud detection, spam detection, and network intrusion detection, which all aim at discovering certain unusual data instances which are different from the regular ones. Depending on the concrete learning tasks and settings, different outlier detection techniques have been proposed already, e.g., supervised outlier detection methods (based on the extracted features about outliers) and unsupervised outlier detection methods (based on clustering algorithms to group instances into clusters, and the isolated instances will be outliers).

In data analysis, missing value is another serious problem, causes of which are very diverse, like unexpected missing data due to device fault in data collection, and intentional missing data due to privacy issues in questionnaire filling. Different techniques can be applied to handle the missing values, e.g., simple elimination of the data instance or attribute containing missing values, missing value estimation, and ignoring missing values in data analysis and model building. Elimination of the data instances or attributes is the simplest way to deal with the missing value problem, but it will also lead to problems in removing important data instances/attributes from the data set. As to the missing value estimation, methods like random missing value guess, mean value refilling, and majority value refilling can be effectively applied. Ignoring the missing values in data analysis requires the learning algorithms to be robust enough in data analysis, which requires necessary calibrations of the models and is out of the scope of data pre-processing.

Data inconsistency is also a common problem in the real-world data analysis, which can be caused by problems in data collection and storage, e.g., mis-reading of certain storage areas or failure in writing protection of some variables in the system. For any two data instances with inconsistent attribute values, a simple way will be to discard one, but extra information may be required in determining which instance to remove. Another common problem in data analysis is data duplication, which refers to the multiple-time occurrence of data instances corresponding to the same information entities in the data set. Data duplication is hard to measure, as it is very challenging to distinguish real duplicated data instances from legitimated data instances corresponding to different information entities. One way to address such a problem will be information entity resolution to identify the data instances actually corresponding to the same information entities.

2.2.3.2 Data Aggregation and Sampling

For many learning tasks, the data set available can be very big involving a large number of data instances. Learning from such a large-scale data set will be very challenging for many learning algorithms, especially those with a high time complexity. To accommodate the data sets for these existing learning algorithms, two data processing operations can be applied: data aggregation and data sampling [49].

Data aggregation denotes the operation of combining multiple data instances into one. As aforementioned, one motivation to apply data aggregation is to reduce the data size as well as the data analysis time cost. Based on the aggregated data sets, many expensive (in terms of space and

jwzhanggy@gmail.com

time costs) learning algorithms can be applied to analyze the data. Another motivation to apply data aggregation is to analyze the data set from a hierarchical perspective. Such an operation is especially useful for data instances with hierarchical attributes, e.g., sale transaction records in different markets, counties, cities, states in the USA. The sales of certain target product can be quite limited in a specific market, and the market level transaction record data will be very sparse. However, by aggregating (i.e., summing up) the sale transactions from multiple markets in the same counties, cities, and even states, the aggregated statistical information will be more dense and meaningful. Data aggregation can also have disadvantages, as it can lead to information loss inevitably, where the low-level data patterns will be no longer available in the aggregated high-level data sets.

Another operation that can be applied to handle the large-scale data sets is data sampling. Formally, sampling refers to the operation of selecting a subset of data instances from the complete data set with certain sampling methods. Data sampling has been used in data investigation for a long time, as processing the complete data set can be extremely time consuming, especially for those with a large size (e.g., billions of records). Selecting a subset of the records allows an efficient data analysis in a short period of time. Meanwhile, to preserve the original data distribution patterns, the sampled data instances should be representative enough to cover the properties about the original data set. Existing data sampling approaches can be divided into two main categories:

- **Random Sampling**: Regardless of the data instances, random sampling selects the data instances from data sets randomly, which is the simplest type of sampling approach. For such a kind of sampling approach, depending on whether the selected instances will be replaced or not, there exist two different variants: (1) random data sampling without replacement, and (2) random data sampling with replacement. In the approach with instance replacement, all the selected instances will be replaced back in the original data set and can be selected again in the rounds afterwards. Random sampling approaches will work well for most regular data sets with a similar size of instances belonging to different types (i.e., class balanced data sets).
- **Stratified Sampling**: However, when handling the data sets with imbalanced class distributions, the random data sampling approach will suffer from many problems. For instance, given a data set with 90% positive instances and 10% negative instances (positive and negative here denote two different classes of data instances, which will be introduced in Sect. 2.3 in detail), random sampling approach is applied to sample 10 of the instances from the original data set. In the sampled data instances, it is highly likely that very few negative instances will be selected due to their scarcity in the original data set. To overcome such a problem, the *stratified sampling* can be applied instead. Stratified sampling will select instances from both positive and negative instance sets separately, i.e., 9 positive instances and 1 negative data instance will be selected finally.

2.2.3.3 Data Dimensionality Reduction and Feature Selection

Besides the number of data instances, the number of attributes used to describe the data instances can be very large as well, which renders many learning algorithms fail to work. Due to the "*curse of dimensionality*" [2, 3, 52], the increase of data dimensionality will make the data much harder to handle. Both the classification and clustering (to be introduced later) tasks will suffer from such high-dimensional data sets due to the large number of variables to be learned in the models and the lack of meaningful evaluation metrics. There exist two classic methods to reduce the data dimensionality, i.e., *dimensionality reduction* and *feature selection* [15, 51], which will be introduced as follows.

Conventional data dimensionality reduction approaches include *principal components analysis (PCA)*, *independent component analysis (ICA)*, and *linear discriminant analysis (LDA)*, etc., which apply linear algebra techniques to project data instances into a lower-dimensional space. PCA is a statistical procedure that uses an orthogonal transformation to convert the observations of possibly correlated variables into a set of linearly uncorrelated variables, which are called the principal

jwzhanggy@gmail.com

components. The objective continuous attributes to be discovered in PCA should be (1) a linear combination of original attributes, (2) orthogonal to each other, and (3) able to capture the maximum amount of variation in the observation data. Different from PCA, ICA aims at projecting the data instances into several independent components, where the directions of these projections should have the maximum statistical independence. Several metrics can be used to measure the independence of these projection directions, like mutual information and non-Gaussianity. LDA can be used to perform supervised dimensionality reduction. LDA projects the input data instances to a linear subspace consisting of the directions which maximize the separation between classes. In LDA, the dimensions of the output are necessarily less than the number of classes. Besides these approaches introduced, there also exist so many other dimensionality reduction approaches, like *canonical correlation analysis (CCA)* and *singular value decomposition (SVD)*, which will not be introduced here since they are out of the scope of this book. A comprehensive review of these dimensionality reduction methods is available in [51].

Dimensionality reduction approaches can effectively project the data instances into a lower-dimensional space. However, the physical meanings of the objective space can be very hard to interpret. Besides dimensionality reduction, another way to reduce the data dimension will be to select a subset of representative attributes from the original attribute set to represent the data instances, i.e., *feature selection* [15]. Among the original attributes, many of them can be either *redundant* or *irrelevant* with each other. Here, the *redundant attributes* denote the attributes sharing duplicated information with the other attributes in the data, while the *irrelevant attributes* represent the attributes which are not useful for the machine learning tasks actually. The physical meanings of these selected attributes will be still the same as the original ones. As illustrated in Fig. 2.6, existing feature selection approaches can be categorized into three groups:

- **Filter Approaches**: Before starting the learning and mining tasks, filter approaches will select the features in advance, which are independent of the learning and mining tasks.

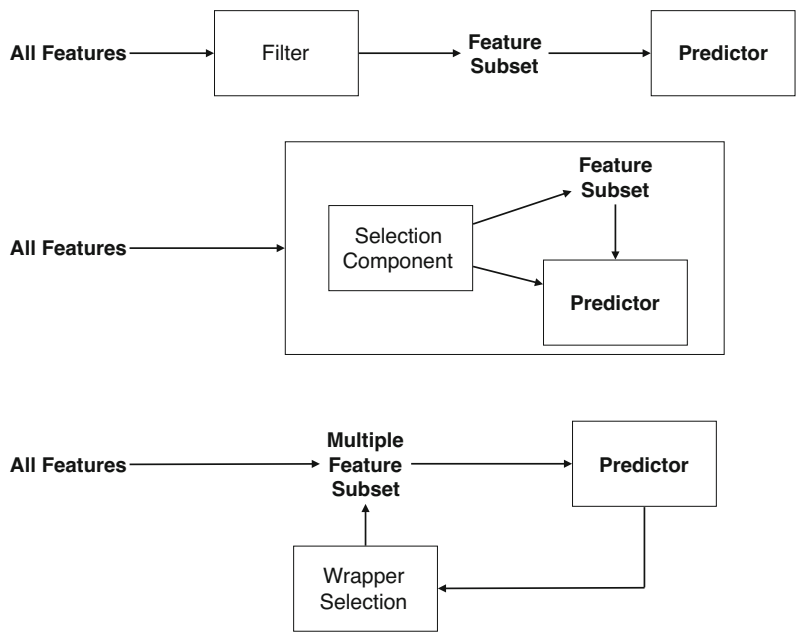

Fig. 2.6 Feature selection approaches (filter approach, embedded approach, and wrapper approach)

jwzhanggy@gmail.com

- **Embedded Approaches**: Some learning models have the ability to do feature selection as one component in the models themselves. Such feature selection approaches are named as the embedded approaches.
- **Wrapper Approaches**: Many other feature selection approaches use the learning model as a black box to find the best subset of attributes which are useful for the objective learning tasks. Such a feature selection approach involves model learning for many times and will not enumerate all the potential attribute subsets, which is named as the *wrapper approach*.

As to the feature subset search algorithms, there exist two classic methods, i.e., the *forward feature selection* and the *backward feature selection*. Forward feature selection approaches begin with an empty set and keep adding feature candidates into the subset, while backward feature selection approaches start with a full set and keep deleting features from the set. To determine which feature to add or delete, different strategies can be applied, like sequential selection and greedy selection. If the readers are interested in feature selection methods, you are suggested to refer to [15] for a more complete literature.

2.2.3.4 Data Transformation

In many cases, the input data set cannot meet the requirements of certain learning algorithms, and some basic data transformation operations will be needed. Traditional data transformation operations include *binarization*, *discretization*, and *normalization* [49].

- **Binarization**: For categorical attributes, we usually need to quantify them into binary representations before feeding them to many learning algorithms. Normally, there are many different categorical attribute binarization methods. For instance, given a categorical attribute with m potential values, we can quantify them into binary codes of length $\log_2^m$. For instance, for an apple sweetness attribute with sweetness degrees {Sour, Slightly Sour, Tasteless, Slight Sweet, Sweet}, it can be quantified into a code of length 3 (like Sour: 001; Slightly Sour: 010; Tasteless: 011; Slight Sweet: 100; Sweet: 101). However, such a quantification method will introduce many problems, as it will create some correlations among the attributes (like "Sour" and "Sweet' will be very similar in their code representations, i.e., "001" vs "101," which share two common digits). A more common way to quantify categorical attributes (of m different values) is to represent them with a code of length m instead. For instance, for the five different sweetness degrees, we use "00001" to represent "Sour," "00010" to represent "Slightly Sour," "00100" to represent "Tasteless," "01000" to represent "Slightly Sweet," and "10000" to represent "Sweet." Such a quantification approach will transform the categorical attributes into independent binary codes. The shortcoming of such a binarization lies in its code length: for the categorical attribute with lots of values, the code will be very long.
- **Discretization**: In many cases, we need to transform continuous attributes into discrete ones instead for easier classification or association analysis, and such a process is called the attribute discretization. For instance, given an attribute with continuous values in range [min, max], we want to discretize the attribute into n discrete values. An easy way to achieve such an objective will be to select $n - 1$ splitting points (e.g., x_1, x_2, ..., x_{n-1}) to divide the value range into n bins (i.e., [min, x_1], (x_1, x_2], ..., (x_{n-1}, max]), where the attribute values in each bin will be denoted as a specific discrete value. Depending on whether the supervision information is used in the splitting points selection or not, existing attribute discretization approaches can be divided into two categories: *supervised attribute discretization* and *unsupervised attribute discretization*. Conventional unsupervised attribute discretization approaches include *equal width* and *equal depth* based splitting points selection, which aims at dividing the data points into intervals of the same

jwzhanggy@gmail.com

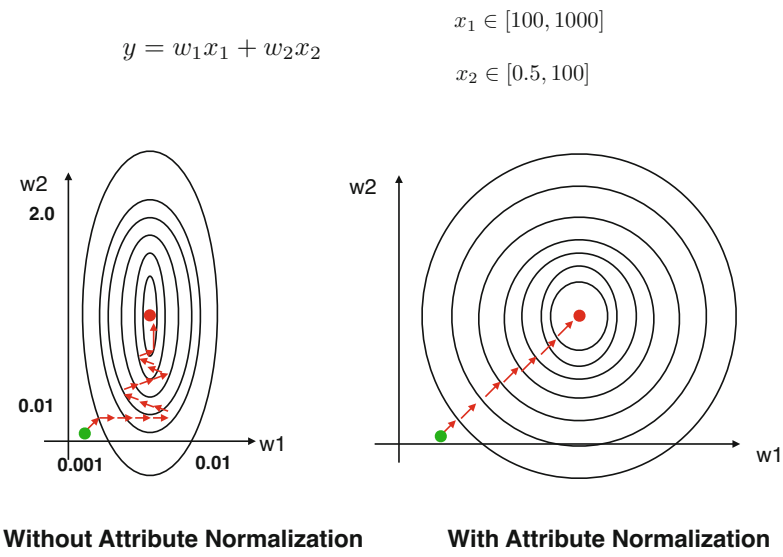

Fig. 2.7 An example of attribute normalization

interval length and the same data point numbers in each interval, respectively. Meanwhile, the supervised attribute discretization approaches will use the supervision information in splitting points selection, and a representative approach is called the *entropy-based* attribute discretization.

- **Normalization**: Attribute normalization is an operation used to standardize the range of independent variables or attributes of data. Since the range of values of raw data varies widely, in some machine learning algorithms, the objective functions can be extremely challenging to solve properly without normalization. The simplest attribute normalization method is Min-Max normalization. The general formula of the Min-Max rescaling approach is given as: $x_{new} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$, where x denotes the value of an attribute, $x_{\min}$ and $x_{\max}$ denote the minimum and maximum values of the correspond attribute, respectively. Besides the Min-Max normalization approach, some other normalization approach includes the Mean-Std normalization approach, whose general formula can be represented as $x_{new} = \frac{x - \bar{x}}{\sigma}$, where $\bar{x}$ and σ represent the mean and standard deviation about an objective attribute.

  To illustrate the motivations of attribute normalization, in Fig. 2.7, we show an example about solving the objective optimization function based on datasets with and without attribute normalization, respectively. Here, we aim at building a linear regression model $y = w_1x_1 + w_2x_2$, where $y \in [0, 1]$ and $x_1 \in [100, 1000]$, $x_2 \in [0.5, 100]$. To learn the model, the objective function will aim at identifying the optimal weight variables w_1 and w_2, which can minimize the model learning loss, i.e., the red dots at the center of the curve. As shown in the left plot of Fig. 2.7, without attribute normalization, the objective function curve is in an oval shape, where w_2 has a relatively wider feasible range compared with w_1. Searching for the optimal point based on such a curve with conventional optimization algorithms, e.g., gradient descent, will be extremely slow. However, with attribute normalization, the model variables w_1 and w_2 will have a similar feasible range and the objective function curve will be very close to a circular shape. The optimal point can be efficiently identified along the function gradient with a very small number of search rounds instead.

This section has covered a brief introductory description about data, data characteristics, and data transformations, which will be used for data analysis and processing in machine learning. Based on

jwzhanggy@gmail.com

the processed data set, we will introduce different kinds of learning tasks in detail in the following sections, respectively.

2.3 Supervised Learning: Classification

Supervised learning tasks aim at discovering the relationships between the input attributes (also called features) and a target attribute (also called labels) of data instances, and the discovered relationship is represented as either the structures or the parameters of the supervised learning models. Supervised learning tasks are mainly based on the assumption that the input features and objective labels of both historical and future data instances are independent and identically distributed (i.i.d.). Based on such an assumption, the supervised learning model trained with the historical data can also be applied to the objective label prediction on the future data. Meanwhile, in many real-world supervised learning tasks, such an assumption can hardly be met and may be violated to a certain degree.

In supervised learning tasks, as illustrated in Fig. 2.8, the data set will be divided into three subsets: training set, validation set (optional), and testing set. For the data instances in the training set and the validation set, their labels are known, which can be used as the supervision information for learning the models. After a supervised learning model has been trained, it can be applied to the data instances in the testing set to infer their potential labels. Depending on the objective label types, the existing supervised learning tasks can be divided into *classification tasks* and *regression tasks*, respectively, where the label set of classification tasks is usually a pre-defined class set $\mathcal{Y}$, while that of regression tasks will be the real number domain $\mathbb{R}$ instead.

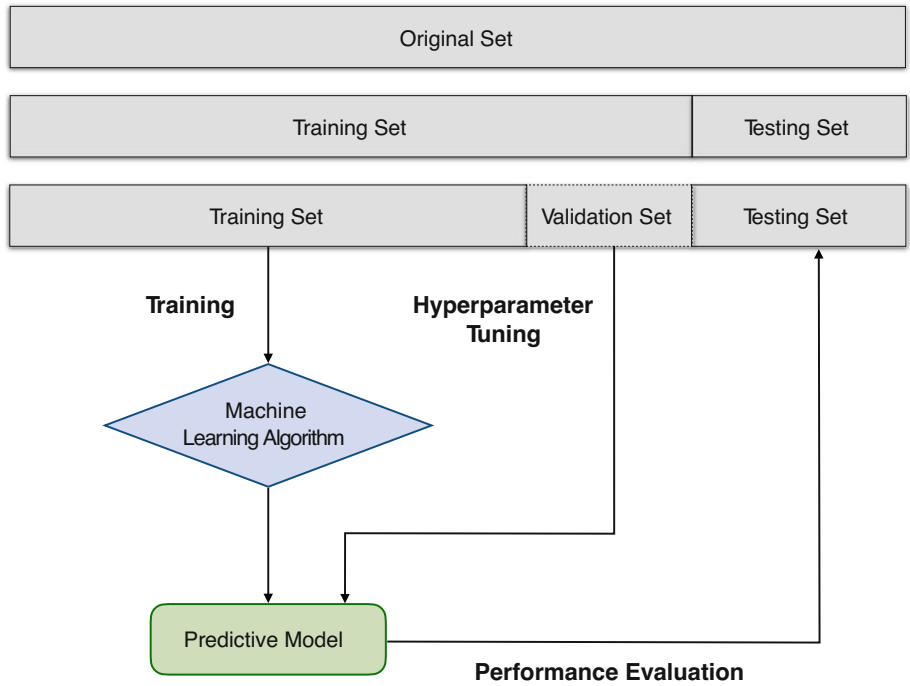

Fig. 2.8 Training set, validation set, and testing set split

jwzhanggy@gmail.com

For both classification and regression tasks, different machine learning models[1] have been proposed already, like classic classification models: *decision tree* [37] and *support vector machine* [9], and classic regression models: *linear regression* [55], *lasso* [50], and *ridge* [20]. In this section, we will mainly focus on introducing the classification learning task, including its learning settings and two classic classification models: *decision tree* and *support vector machine*. A brief introduction to the classic regression models will be provided in the next section.

2.3.1 Classification Learning Task and Settings

In classification problem settings, the data sets are divided into three disjoint subsets, which include a training set, a validation set, and a testing set. Generally, the data instances in the training set are used to train the models, i.e., to learn the model structure or parameters. The validation set is used for some hyperparameter selection and tuning. And the testing set is mainly used for evaluating the learning performance of the built models. Many different methods have been proposed to split the data set into these subsets, like *multiple random sampling* and *cross validation* [25].

- **Multiple Random Sampling**: Given a data set containing n data instances, *multiple random sampling* strategy will generate two separate subsets of instance for model training and validation purposes, where the remaining instances will be used for model testing only. In some cases, such a data set splitting method will encounter the unreliability problem, as the testing set can be too small to be representative for the overall data set. To overcome this problem, such a process will be performed n times, and in each time, different training and testing sets will be produced.
- **Cross Validation**: When the data set is very small, *cross validation* will be a common strategy for splitting the data set. There are different variants of cross validation, like *k-fold cross validation* and *leave-one-out cross validation*. As shown in Fig. 2.9, in the *k-fold cross validation*, the data set is divided into k equal sized subsets, where each subset can be picked as a testing set while the remaining $k-1$ subsets are used as the training set (as well as the validation set). Such a process runs for k times, and in each time a different subset will be used as the testing set. The *leave-one-out cross validation* works in a similar way, which picks one single instance as the testing set and the remaining instances will be used as the training set in each round. For the data sets of a large scale, the *leave-one-out cross validation* approach will suffer from the high time cost problem. In the real-world model learning, the *n-fold cross validation* is used more frequently for data set splitting compared with the other strategies.

Formally, in the classification task, we can denote the feature domain and label domain as $\mathcal{X}$ and $\mathcal{Y}$, respectively. The objective of classification tasks is to build a model with the training set and the validation set (optional), i.e., $f : \mathcal{X} \to \mathcal{Y}$, to project the data instances from their feature representations to their labels. Formally, the data instances in the training set can be represented as a set of n feature-label pairs $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathcal{X}$ denotes the feature vector of the ith data instance and $y_i \in \mathcal{Y}$ represents its corresponding label. The validation set $\mathcal{V}$ can be represented in a similar way with both features and known labels for the data instances. Meanwhile, the data instances in the testing set are different from those in the training and validation sets, which only have the feature representation only without any known labels. Formally, the testing set involving

[1]Machine learning models usually denote the well trained learning algorithms by some training data. In the sequel of this book, we will not differentiate the differences between machine learning models and machine learning algorithms by default.

jwzhanggy@gmail.com

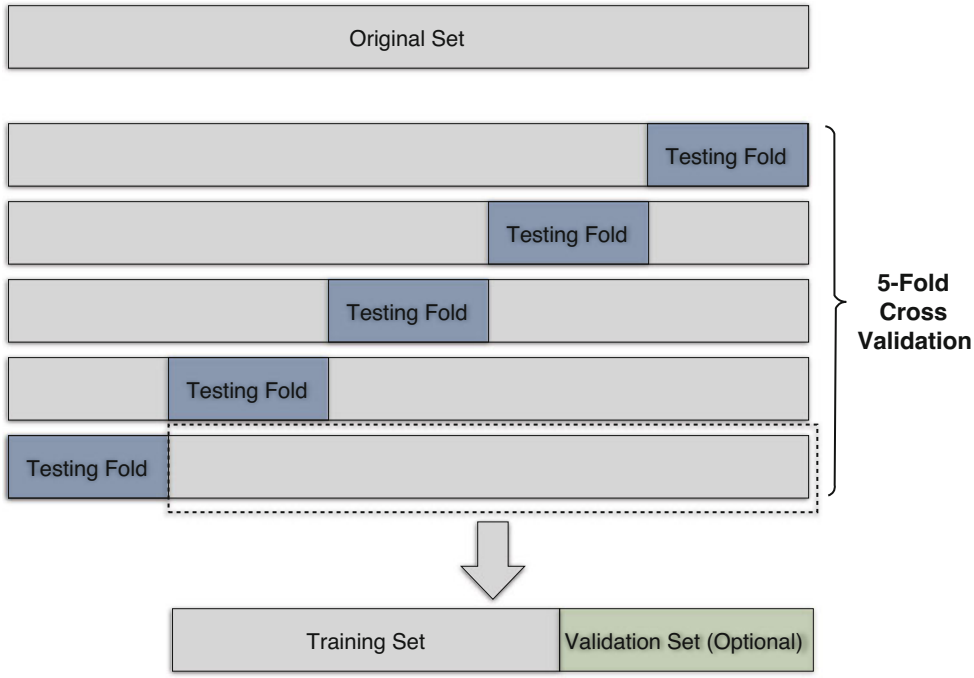

Fig. 2.9 An example of five-fold cross validation

m data instances can be represented as a set $\mathcal{S} = \{\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \ldots, \mathbf{x}_{n+m}\}$. Based on the well-trained models, we will be able to identify the labels of the data instances in the testing set.

Next, we will take the binary classification task as an example (i.e., $\mathcal{Y} = \{+1, -1\}$ or $\mathcal{Y} = \{+1, 0\}$) and introduce some classic classification algorithms, including *decision tree* [37] and *support vector machine* [9].

2.3.2 Decision Tree

In this section, we will introduce the classic decision tree classification algorithm [37], and will take the binary classification problem as an example. For binary classification tasks, the objective class domain involves two different pre-defined class labels $\{+1, -1\}$ (i.e., the positive class label and negative class label). The procedure of classifying data instances into positive and negative classes actually involves a series of decision-making about questions, like "whether this elephant has two trunk fingers?" or "is the elephant weight greater than 6 tons?" As indicated by the name, decision tree is such a kind of classification model, which is based on a series of decision-making procedure. Before achieving the final decision (i.e., data instances belonging to positive/negative classes), a group of pre-decisions will be made in advance. The final learned model can be represented as a tree structured diagram, where each internal node represents a "question" and each branch denotes a decision option. Decision tree is one of the most widely used machine learning algorithms for classification tasks due to its effectiveness, efficiency, and simplicity.

jwzhanggy@gmail.com

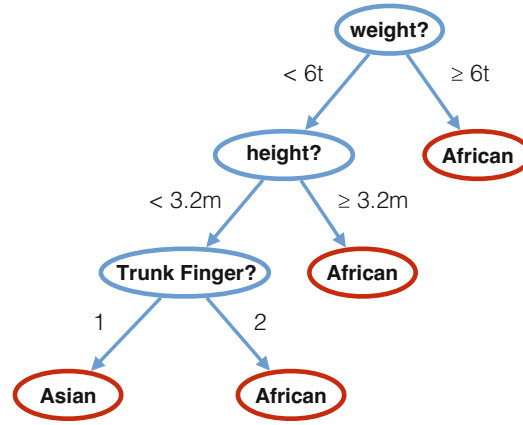

Fig. 2.10 An example of decision tree model built for the elephant data set

2.3.2.1 Algorithm Overview

To help illustrate the decision tree model more clearly, based on the elephant dataset shown in Table 2.1, we provide a trained decision tree model in Fig. 2.10.

Example 2.1 For the elephant classification example shown in Fig. 2.10, the attribute of interest is the "*Elephant Category*" attribute, which serves as the classification objective label and takes value from {Asian, African}. Based on the "*Weight*" attribute, the data instances can be divided into two subsets: (1) elephant (data instances) with "*Weight*" no less than 6 t (i.e., instance {1}), and (2) elephant (data instances) with "*Weight*" less than 6 t (i.e., {2, 3, 4, 5, 6}). For the data instances in the first group (i.e., weight $\geq$6 t), they all belong to the African elephant category; while the second group involves both Asian and African elephants. Meanwhile, based on the elephant height (i.e., height $\geq$3.2 m or height $<$3.2 m), we can further divide the second group into two sub-groups: {2} and {3, 4, 5, 6}, where instance 2 belongs to the African elephant category. The remaining elephants in {3, 4, 5, 6} can be precisely divided into {5} and {3, 4, 6} based on the trunk "finger" number attribute (i.e., 1 trunk finger or 2 trunk fingers), where the instance in the first group has the Asian elephant label and those in the second group all have the African elephant label.

From the built decision tree model shown in Fig. 2.10, we observe that there exist two types of nodes in the tree: (1) decision node (i.e., the non-leaf node in blue color), and (2) result node (i.e., the leaf node in red color). Each decision node represents an attribute test, which divides the data instances into two groups (i.e., two branches). In the ideal case, each result node represents a classified label, which is usually pure in terms of the label values. In other words, the training data instances classified into each result node should all have the same label. For instance, in the tree shown in Fig. 2.10, we can exactly determine the "*Elephant Category*" attribute of all the data instances in Table 2.1. In other words, the decision tree model in Fig. 2.10 is well trained and can be applied to precisely classify the elephant data instances in Table 2.1.

Decision tree applies the *divide-and-conquer* strategy in model learning, which partitions the data instances into different groups based on their attributes. The pseudo-code of decision tree learning is provided in Algorithm 1. From the pseudo-code, we observe that the training process of the decision tree model involves a recursive process. In each recursion, the returning conditions of the algorithm include:

1. All the data instances belong to the same class and no need for further division.
2. The attribute set is empty and the data instances cannot be further divided.

jwzhanggy@gmail.com

Algorithm 1 DecisionTree

Require: Training Set $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$
Attribute Set $\mathcal{A} = \{a_1, a_2, \cdots, a_d\}$.
Ensure: A Trained Decision Tree Model
generate a node N
if instances in $\mathcal{T}$ belong to the same class **then**
  mark node N as the result node; **Return**
end if
if $\mathcal{A} == \emptyset$ **OR** instances in $\mathcal{T}$ take the same values in attribute set $\mathcal{A}$ **then**
  mark node N as the result node, whose corresponding label is the majority class of instances in $\mathcal{T}$; **Return**
end if
select the optimal partition attribute a^* from $\mathcal{A}$
for all values a_v^* for attribute a^* **do**
  generate a branch node $N_{a_v^*}$ for node N
  select an instance subset $\mathcal{T}_{a_v^*} \subset \mathcal{T}$ taking value a_v^* for attribute a^*
  if $\mathcal{T}_{a_v^*} == \emptyset$ **then**
    mark branch node $N_{a_v^*}$ as the result node, whose corresponding label will be the majority class of instances in $\mathcal{T}$; **Continue**
  else
    $SubTree$ = DecisionTree($\mathcal{T}_{a_v^*}$, $\mathcal{A} \setminus \{a^*\}$)
    assign root node of $SubTree$ to the branch node $N_{a_v^*}$
  end if
end for
Return a decision tree model with N as the root

3. All the data instances take the same values in the provided attribute set, which cannot be further divided.
4. The provided data instance set is empty and cannot be further divided.

In addition, for these different cases, the operations to be performed in Algorithm 1 will be different.

- In case 1, the current node N will be marked as a result node, and its corresponding label will be the common class label of data instances in $\mathcal{T}$. The current node N will also be returned as a single-node decision tree to the upper level function call.
- In cases 2 and 3, the current node N is marked as the result node, and its corresponding label will be the majority class of instances in $\mathcal{T}$. The current node N will be returned as a single-node decision tree to the upper level function call.
- In case 4, the generated node $N_{a_v^*}$ is marked as a result node, and its corresponding label will be the majority class of instances in $\mathcal{T}$. The algorithm will continue to the next possible value of the selected optimal division attribute a^*.
- Otherwise, the algorithm will make a recursive call of the DecisionTree function, whose returned root node will be assigned to the branch node $N_{a_v^*}$.

2.3.2.2 Attribute Selection Metric: Information Gain

According to Algorithm 1, the structure of the built decision tree is heavily dependent on the selection of the optimal division node a^* in each recursive step, which will affect the performance of the model greatly. Generally, via a sequential attribute test from the decision tree root node to the result node, the data instances will be divided into several different groups and each decision tree result node corresponds to one data instance group. In general, we may want the data instances in each divided group to be as pure as possible. Here, the concept "pure" denotes that most/all the data instances in each group should have the same class label, and attributes that can introduce the maximum "*purity*

jwzhanggy@gmail.com

increase" will be selected first by the model. Viewed in such a perspective, the quantification of "*purity*" and "*purity increase*" will be very important for the decision tree model building.

Given the current training set $\mathcal{T}$, depending on the data instance labels, the ratio of the data instances belong to the lth class can be denoted as p_l ($l \in \{1, 2, \ldots, |\mathcal{Y}|\}$). Meanwhile, based on a certain attribute $a_i \in \mathcal{A}$ in the attribute set, the data instances in $\mathcal{T}$ can be partitioned into k groups $\mathcal{T}_{a_{i,1}}, \mathcal{T}_{a_{i,2}}, \ldots, \mathcal{T}_{a_{i,k}}$, where $\{a_{i,1}, a_{i,2}, \ldots, a_{i,k}\}$ denotes the value set of attribute a_i. Therefore, the ratio of data instances taking value $a_{i,j}$ for attribute a_i in the current data instance can be represented as $\frac{|\mathcal{T}_{a_{i,j}}|}{|\mathcal{T}|}$.

Based on these notations, different "*purity*" and "*purity increase*" concept quantification metrics have been proposed for the optimal attribute selection already, like *Entropy* [47] and *Information Gain* [39, 47]. Formally, given a data set $\mathcal{T}$ together with the data instance class distribution ratios $\{p_l\}_{l \in \{1,2,\ldots,|\mathcal{Y}|\}}$, the "*purity*" of all the data instances can be represented with the *information entropy* concept as follows:

$$Entropy(\mathcal{T}) = -\sum_{l=1}^{|\mathcal{Y}|} p_l \log_2 p_l. \tag{2.1}$$

Meanwhile, after picking an attribute a_i, the data instances in $\mathcal{T}$ will be further divided into k subsets $\mathcal{T}_{a_{i,1}}, \mathcal{T}_{a_{i,2}}, \ldots, \mathcal{T}_{a_{i,k}}$. If the class label distribution in each of these k subsets is pure, i.e., the overall entropy is small, the selected attribute a_i will be a good choice. Formally, the *information gain* [39, 47] introduced by attribute a_i on data instance set $\mathcal{T}$ can be represented as

$$\begin{aligned} Gain(\mathcal{T}, a_i) = &\, Entropy(\mathcal{T}) \\ &- \sum_{a_{i,j}} \frac{|\mathcal{T}_{a_{i,j}}|}{|\mathcal{T}|} Entropy(\mathcal{T}_{a_{i,j}}). \end{aligned} \tag{2.2}$$

The optimal attribute in the current attribute set $\mathcal{A}$ that can lead to maximum *information gain* can be denoted as

$$a^* = \arg_{a_i \in \mathcal{A}} \max Gain(\mathcal{T}, a_i). \tag{2.3}$$

The famous ID3 Decision Tree algorithm [37] applies *information gain* as the division attribute selection metric. Besides *information gain*, many other measures can be used as the best attribute selection metrics as well, e.g., *information gain ratio* and *Gini index*, which will be introduced in the following part.

2.3.2.3 Other Attribute Selection Metrics

Actually, *information gain* is a biased attribute selection metric, which favors the attributes with more potential values to take and may have negative effects on the attribute selection. To overcome such a disadvantage, a new metric named *information gain ratio* [39] is proposed, which normalizes the *information gain* by the corresponding attributes' *intrinsic values*. Formally, based on the current data instance set $\mathcal{T}$, the *intrinsic value* of an attribute a_i can be represented as

$$IV(\mathcal{T}, a_i) = -\sum_{a_{i,j}} \frac{|\mathcal{T}_{a_{i,j}}|}{|\mathcal{T}|} \log \frac{|\mathcal{T}_{a_{i,j}}|}{|\mathcal{T}|}. \tag{2.4}$$

jwzhanggy@gmail.com

Based on the above notation, the *information gain ratio* introduced by attribute a_i can be formally represented as follows:

$$GainRatio(\mathcal{T}, a_i) = \frac{Gain(\mathcal{T}, a_i)}{IV(\mathcal{T}, a_i)}. \tag{2.5}$$

The optimal attribute to be selected in each round will be those which can introduce the maximum *information gain ratio* instead, and the selection criterion can be formally represented as

$$a^* = \arg_{a_i \in \mathcal{A}} \max GainRatio(\mathcal{T}, a_i). \tag{2.6}$$

However, *information gain ratio* is also shown to be a biased metric. Different from the *information gain*, the *information gain ratio* metric favors the attributes with less potential values instead. Therefore, the famous C4.5 Decision Tree algorithm [38] applies heuristics to pre-select the attribute candidates with *information gain* that is larger than the average, and then applies *information gain ratio* to select the optimal attribute.

Another regularly used division node selection metric is *Gini index* [39]. Formally, the *Gini* metric of a given data instance set $\mathcal{T}$ can be represented as

$$Gini(\mathcal{T}) = \sum_{i=1}^{|\mathcal{Y}|} \sum_{i'=1, i' \neq i}^{|\mathcal{Y}|} p_i p_{i'} = 1 - \sum_{i=1}^{|\mathcal{Y}|} p_i^2. \tag{2.7}$$

Here, p_i denotes the data instance ratio belonging to the ith class. In general, mixed data instance set with class labels evenly distributed will have larger *Gini* scores. Based on the *Gini* metric, we can define the *Gini index* [39] of the data set about attribute $a_i \in \mathcal{A}$ as

$$GiniIndex(\mathcal{T}, a_i) = \sum_{a_{i,j}} \frac{|\mathcal{T}_{a_{i,j}}|}{|\mathcal{T}|} Gini(\mathcal{T}_{a_{i,j}}). \tag{2.8}$$

The optimal selection of the division attribute will be that introducing the minimum *Gini index*, i.e.,

$$a^* = \arg_{a_i \in \mathcal{A}} \min GiniIndex(\mathcal{T}, a_i). \tag{2.9}$$

2.3.2.4 Other Issues

Besides the optimal attribute selection, there also exist many other issues that should be studied in the decision tree algorithm.

1. Overfitting and Branch Pruning *Overfitting* [17] is a common problem encountered in the learning tasks with supervision information. Model overfitting denotes the phenomenon that the model fits the training data "too good," which treats and captures some specific pattern in the training set as a common pattern in the whole data set, but will achieve a very bad performance when being applied to some unseen data instances. Formally, given the training data set, testing set, and two trained decision tree models f_1 and f_2, model f_1 is said to overfit the data set $\mathcal{T}$, if the other f_1 achieves higher accuracy on the training set than f_2, but performs much worse than f_2 on the unseen testing set. In the overfitting scenario, the built decision tree can be very deep with so many branches, which will classify all the instances in the training set perfectly without making any mistake, but can hardly be generalized to the unseen data instances.

jwzhanggy@gmail.com

To overcome such a problem, different techniques can be applied in the decision tree model building process, like *branch pruning* [38]. The *branch pruning* strategy applied in decision tree training process includes both *pre-pruning* and *post-pruning*, as introduced in [38]. For the *pre-pruning* strategy, in building the decision tree model, before generating a child node for the decision nodes, certain tests will be performed. If dividing the current decision node into several child nodes will not generalize the model to improve its performance, the current decision node will be marked as the result node. On the other hand, for the *post-pruning* strategy, after the original decision tree model has been built, the strategy will check all the decision nodes. If replacing the decision node as a result node will generalize the mode to improve the performance, the sub-tree rooted at the current decision node will be replaced by a result node instead.

Generally, *pre-pruning* strategy can be dangerous, as it is actually not clear what will happen if the tree is extended further without *pre-pruning*. Meanwhile, the *post-pruning* strategy is more useful, as it is based on the complete built decision tree model, and it is clear which branch of the tree is useful and which one is not. *Post-pruning* has been applied in many existing decision tree learning algorithms.

2. Missing Value In many cases, there can exist some missing values for the data instances in the data set, which is very common for data obtained from areas, like social media and medical care. Due to the personal privacy protection concerns, people may hide some personal important information or sensitive information in the questionnaire. The classic way to handle the missing values in data mining will be to fill in the entries with some special values, like "*Unknown*." In addition, if the attribute takes discrete values, we can also fill in the missing entries with the most frequent value of that attribute; if the attribute is continuous, we can fill in the missing entries with mean value of the attribute.

Meanwhile, for the decision tree model, besides these common value filling techniques, some other methods can also be applied to handle the missing value problem. In the classic decision tree algorithm C4.5 [38], at a tree decision node regarding a certain attribute, it can distribute the training data instances with missing values for that attribute to each branch of the tree proportionally according to the distribution of the training instances. For example, let's take a as an attribute to be dealt with at the current decision tree, and assume $\mathcal{T}$ to be the current data instance set and $\mathcal{T}_{a_i} \subset \mathcal{T}$ to be a subset of data instances with value a_i for attribute a. For each data instances $\mathbf{x} \in \mathcal{T}$, a weight $w_{\mathbf{x}}$ will be assigned to $\mathbf{x}$. We propose to define the ratio of data instances with value a_i to be

$$r_{a_i} = \frac{\sum_{\mathbf{x} \in \mathcal{T}_{a_i}} w_{\mathbf{x}}}{\sum_{\mathbf{x} \in \mathcal{T}} w_{\mathbf{x}}}. \tag{2.10}$$

If data instance $\mathbf{x}$ has no value for attribute a, $\mathbf{x}$ will be assigned to all the child nodes of a, i.e., the branch corresponding to values $\{a_1, a_2, \ldots, a_k\}$. What's more, the weight of $\mathbf{x}$ for the branch corresponding to value a_i will be $r_{a_i} \cdot w_{\mathbf{x}}$.

3. Multi-Variable Decision Tree Traditional decision tree model tests one single attribute at each of the decision node once. If we take each attribute as a coordinate of a data instance, the decision boundaries outlined by the decision tree model will be parallel to the axes, which renders the classification results interpretable but the decision boundary can involve too many small segments to fit real decision boundaries.

jwzhanggy@gmail.com

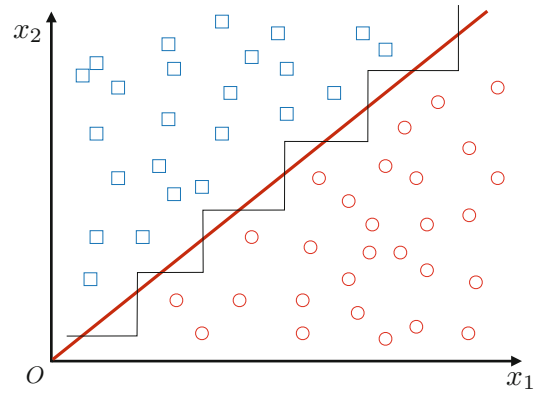

Fig. 2.11 Classic decision tree to fit complex decision boundary

Example 2.2 For instance, as shown in Fig. 2.11, given the data instances with two features $\mathbf{x} = [x_1, x_2]^\top$ and one label y with decision boundary denoted by the red line, where instances lying at the top left are positive (i.e., the blue squares) while those at the bottom right are negative (i.e., the red circles) instead. To fit such a decision boundary, the decision tree model will involve a series of zig-zag segments (i.e., the black lines) as the decision boundary learned from the data set.

One way to resolve such a problem is to involve multiple variables in the tests of decision nodes. Formally, the decision test at each decision node can be represented as $\sum_i w_i a_i = t$, where w_i denotes the weight of the ith attribute a_i. Formally, all the involved variables, i.e., weight w_i for $\forall a_i$ together with the threshold value t, can be learned from the data. Formally, the decision tree algorithm involving multiple variables at each decision node test is named as the *multi-variable decision tree* [6]. Different from the classic *single-variable decision tree* algorithm, at each decision node, instead of selecting the optimal attribute for division, *multi-variable decision tree* aims at finding the optimal linear classifier, i.e., the optimal weights and threshold. Based on the *multi-variable decision tree*, the data instances shown in Fig. 2.11 can be classified by test function $x_1 - x_2 = 0$ perfectly, where instances with attributes $x_1 - x_2 < 0$ will be partitioned into one branch and those with attributes $x_1 - x_2 > 0$ will be partitioned into another branch instead.

2.3.3 Support Vector Machine

In this part, we will introduce another well-known classification model, which is named as the support vector machine (*SVM*) [9]. *SVM* is a supervised learning algorithm that can analyze data used for classification tasks. Given a set of training instances belonging to different classes, the *SVM* learning algorithm aims at building a model that assigns the data instances to one class or the other, making it a non-probabilistic binary linear classifier. In *SVM*, the data instances are represented as the points in a feature space, which can be divided into two classes by a clear hyperplane learned by the model, where the gap between the division boundary should be as wide as possible. New data instances will be mapped into the same space and classified into a class depending on which side of the decision boundary they fall in. In addition to performing linear classification, *SVM* can also efficiently perform a non-linear classification using the kernel trick [9], implicitly mapping their inputs into a high-dimensional feature space. In this part, we will introduce the *SVM* algorithm in detail, including its objective function, dual problem, and the kernel trick.

jwzhanggy@gmail.com

2.3.3.1 Algorithm Overview

Given a training set involving n labeled instances $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ belonging to binary classes $\{+1, -1\}$, the *SVM* model aims at identifying a hyperplane to separate the data instances. Formally, in the feature space, a division hyperplane can be represented as a linear function

$$\mathbf{w}^\top \mathbf{x} + b = 0, \tag{2.11}$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_d]^\top$ denotes the scalars of each feature dimension. Scalar vector $\mathbf{w}$ also denotes the direction of the hyperplane, and b indicates the shift of the hyperplane from the original point.

For any data instance in the training set, e.g., $(\mathbf{x}_i, y_i) \in \mathcal{T}$, the distance from point $\mathbf{x}_i$ to the division hyperplane can be mathematically represented as

$$d_{\mathbf{x}_i} = \frac{|\mathbf{w}^\top \mathbf{x}_i + b|}{\|\mathbf{w}\|}. \tag{2.12}$$

For a good division hyperplane, it should be able to divide the data instances into different classes correctly. In other words, for the data instance $\mathbf{x}_i$ above the hyperplane, i.e., $\mathbf{w}^\top \mathbf{x}_i + b > 0$, it should belong to one class, e.g., the *positive class* with $y_i = +1$. Meanwhile, for the data instance below the hyperplane, it should belong to the other class, e.g., the *negative class* with $y_i = -1$. To select the optimal division hyperplane, *SVM* rescales the variables $\mathbf{w}$ and b to define two other hyperplanes (namely, the *positive* and *negative* hyperplanes) with the following equations:

$$H_+ : \mathbf{w}^\top \mathbf{x} + b = +1, \tag{2.13}$$

$$H_- : \mathbf{w}^\top \mathbf{x} + b = -1, \tag{2.14}$$

such that the following equations can hold:

$$\begin{cases} \mathbf{w}^\top \mathbf{x}_i + b \geq +1, & \forall (\mathbf{x}_i, y_i) \in \mathcal{T}, \text{if } y_i = +1, \\ \mathbf{w}^\top \mathbf{x}_i + b \leq +1, & \forall (\mathbf{x}_i, y_i) \in \mathcal{T}, \text{if } y_i = -1. \end{cases} \tag{2.15}$$

We know that the division hyperplane is parallel to the *positive* and *negative* hyperplanes defined above with equal distance between them. Meanwhile, for the data instances which actually lie in the *positive* and *negative* hyperplanes, they are called the *support vectors*. The distance between the positive and negative hyperplanes can be formally represented as

$$d_{+,-} = \frac{2}{\|\mathbf{w}\|}. \tag{2.16}$$

The *SVM* model aims at finding a classification hyperplane which can maximize the distance between the positive and negative hyperplanes $\left(\text{or minimize } \frac{\|\mathbf{w}\|^2}{2} \text{ equivalently}\right)$, while ensuring all the data instances are correctly classified. Formally, the objective function of the *SVM* model can be represented as

$$\begin{aligned} &\min_{\mathbf{w},b} \frac{\|\mathbf{w}\|^2}{2}, \\ &s.t.\ y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall (\mathbf{x}_i, y_i) \in \mathcal{T}. \end{aligned} \tag{2.17}$$

jwzhanggy@gmail.com

The objective function is actually a convex quadratic programming problem involving $d + 1$ variables and n constraints, which can be solved with the existing convex programming toolkits. However, solving the problem can be very time consuming. A more efficient way to address the problem is to transform the problem to its dual problem and solve the dual problem instead, which will be talked about in the following part.

2.3.3.2 Dual Problem

The primal objective function of *SVM* is convex. By applying the Lagrangian multiplier method, the corresponding Lagrange function of the objective function can be represented as

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \alpha_i (1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)), \tag{2.18}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^\top$ $(\alpha_i \geq 0)$ denotes the vector of multipliers.

By taking the partial derivatives of $L(\mathbf{w}, b, \boldsymbol{\alpha})$ with regard to $\mathbf{w}$, b and making them equal to 0, we will have

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \tag{2.19}$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{2.20}$$

According to the representation, by replacing $\mathbf{w}$ and $\sum_{i=1}^{n} \alpha_i y_i$ with $\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ and 0 in Eq. (2.18), respectively, we will have

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j. \tag{2.21}$$

With the above derivatives, we can achieve a new representation of $L(\mathbf{w}, b, \boldsymbol{\alpha})$ together with the constraints $\alpha_i \geq 0, \forall i \in \{1, 2, \ldots, n\}$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$, which actually defines the dual problem of the objective function:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ &s.t. \sum_{i=1}^{n} \alpha_i y_i = 0, \\ &\quad \alpha_i \geq 0, \forall i \in \{1, 2, \ldots, n\}. \end{aligned} \tag{2.22}$$

Why should we introduce the dual problem? To answer this question, we need to introduce an important function property as follows. For function $L(\mathbf{w}, b, \boldsymbol{\alpha})$, we have

$$\max_{\boldsymbol{\alpha}} \min_{\mathbf{w}, b} L(\mathbf{w}, b, \boldsymbol{\alpha}) \leq \min_{\mathbf{w}, b} \max_{\boldsymbol{\alpha}} L(\mathbf{w}, b, \boldsymbol{\alpha}). \tag{2.23}$$

The proof to the above property will be left as an exercise at the end of this chapter.

jwzhanggy@gmail.com

In other words, the optimal dual problem actually defines an upper bound of the optimal solution to the primal problem. Here, we know $\frac{1}{2}\|\mathbf{w}\|^2$ is convex and $\mathbf{w}^\top\mathbf{x}+b$ is affine. In Eq. (2.23), the equal sign $=$ can be achieved iff $\mathbf{w}$, $\mathbf{b}$, and $\boldsymbol{\alpha}$ can meet the following KKT (Karush-Kuhn-Tucker) [9] conditions:

$$\begin{cases} \alpha_i \geq 0, \\ y_i(\mathbf{w}^\top\mathbf{x}_i+b)-1 \geq 0, \\ \alpha_i(y_i(\mathbf{w}^\top\mathbf{x}_i+b)-1)=0. \end{cases} \tag{2.24}$$

In other words, in such a case, by solving the dual problem, we will be able to achieve the primal problem as well.

According to the third KKT conditions, we observe that $\forall(\mathbf{x}_i, y_i) \in \mathcal{T}$, at least one of the following two equations must hold:

$$\alpha_i = 0, \tag{2.25}$$

$$y_i(\mathbf{w}^\top\mathbf{x}_i+b)=1. \tag{2.26}$$

For the data instances with the corresponding $\alpha_i = 0$, they will not actually appear in the objective function of neither the primal nor the dual problem. In other words, these data instances are "*useless*" in determining the model variables (or the decision boundary). Meanwhile, for the data instances with $y_i(\mathbf{w}^\top\mathbf{x}_i+b)=1$, i.e., those data points lying in the *positive* and *negative* hyperplanes H_+ and H_-, we need to learn their corresponding optimal multiplier scalar α_i, which will affect the final learned models. Therefore, the *SVM* model variables will be mainly determined by these *support vectors*, which is also the reason why the model is named as the *support vector machine*.

As we can observe, the dual problem is also a quadratic programming problem involving n variables and n constraints. However, in many cases, solving the dual problem is still much more efficiently than solving the primal, especially when $d \gg n$. Solving the dual objective function doesn't depend on the dimension of the feature vectors, which is very important for feature vectors of a large dimension or the application of *kernel tricks* when the data instances are not linearly separable.

Therefore, by deriving and addressing the dual problem, we will be able to understand that *support vectors* play an important role in determining the classification boundary of the *SVM* model. In addition, the *dual problem* also provides the opportunity for the efficient model learning especially with the *kernel tricks* to be introduced in the following part.

Some efficient learning algorithms have been proposed to solve the dual objective function, like SMO (sequential minimal optimization) algorithm [36], which will further reduce the learning cost. We will not introduce SMO here, since it is not the main focus of this textbook. Based on the learned optimal $\boldsymbol{\alpha}^*$, we can obtain the optimal $\mathbf{w}^*$ and b^* variables of the *SVM* model, which will be used to classify the future data instances (e.g., featured by vector $\mathbf{x}$) based on the sign of function $(\mathbf{w}^*)^\top\mathbf{x}+b^*$.

2.3.3.3 Kernel Trick

In the case when the data instances are not linearly separable, one effective way to handle the problem is to project the data instances to a high-dimensional feature space, in which the data instances will be linearly separable by a hyperplane, and such a technique is called the *kernel trick* (or *kernel method*) [9]. The *kernel trick* has been shown to be very effective when applied in *SVM*, which allows *SVM* to classify the data instances following very complicated distributions.

jwzhanggy@gmail.com

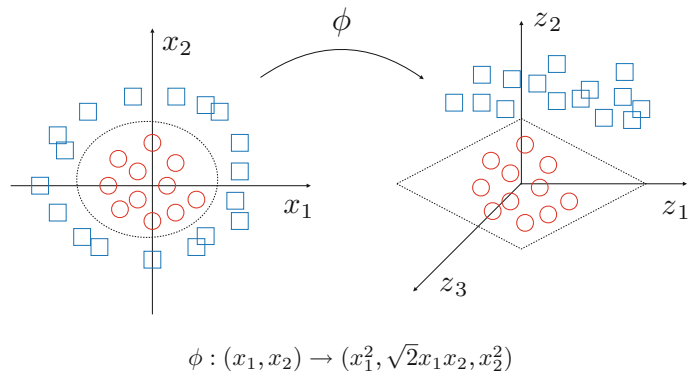

Fig. 2.12 An example of kernel function in SVM

Example 2.3 In Fig. 2.12, we show an example to illustrate the *kernel trick* with *SVM*. Given a group of data instances in two different classes, where the *red circle* denotes the *positive class* and the *blue square* denotes the *negative class*. According to the data instance distribution in the original feature space (represented by two features x_1 and x_2), we observe that they cannot be linearly separated by drawing a line actually. To divide these data instances, a non-linear division boundary will be needed, i.e., the dashed line in black. Meanwhile, if we project the data instances from the two-dimensional feature space to a three-dimensional feature space with the indicated kernel function $\phi : (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$, we can observe that those data instances will become linearly separable with a hyperplane in the new feature space.

Formally, let $\phi : \mathcal{R}^d \rightarrow \mathcal{R}^D$ be a mapping that projects the data instances from a d-dimensional feature space to another D-dimensional feature space. In the new feature space, let's assume the data instances can be linearly separated by a hyperplane in the *SVM* model. Formally, the hyperplane that can separate the data instances can be represented as

$$\mathbf{w}^\top \phi(\mathbf{x}) + b = 0, \tag{2.27}$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_D]^\top$ and b are the variables to be learned in the model.

According to Sect. 2.3.3.2, the primal and dual optimization objective functions of the *SVM* model in the new feature space can be formally represented as

$$\begin{aligned} &\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 \\ &s.t.\ y_i(\mathbf{w}^\top \phi(\mathbf{x}_i) + b) \geq 1, i = 1, 2, \ldots, n; \end{aligned} \tag{2.28}$$

and

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) \\ &s.t. \ \sum_{i=1}^{n} \alpha_i y_i = 0, \\ &\quad \alpha_i \geq 0, \forall i \in \{1, 2, \ldots, n\}. \end{aligned} \tag{2.29}$$

jwzhanggy@gmail.com

Here, $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ denotes the inner projection of two projected feature vectors, calculation cost of which will be very expensive if the new feature space dimension D is very large. For simplicity, we introduce a new notation $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ to represent $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$, and rewrite the above dual objective function as follows:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ &s.t. \sum_{i=1}^{n} \alpha_i y_i = 0, \\ &\alpha_i \geq 0, \forall i \in \{1, 2, \ldots, n\}. \end{aligned} \tag{2.30}$$

By solving the above function, we can obtain the optimal $\boldsymbol{\alpha}^*$, based on which the classifier function can be represented as

$$\begin{aligned} f(\mathbf{x}) &= (\mathbf{w}^*)\phi(\mathbf{x}) + b^* \\ &= \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}) + b \\ &= \sum_{i=1}^{n} \alpha_i^* y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b, \end{aligned} \tag{2.31}$$

where $\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(\mathbf{x}_i)^\top$ according to the derivatives in Eq. (2.19).

We can observe that in both the training and testing processes, we don't really need the concrete representations of the projected feature vectors $\{\phi(\mathbf{x}_i)\}_{\mathbf{x}_i}$ but a frequent calculation of $\kappa(\cdot, \cdot)$ will be needed instead. The representation of $\kappa(\cdot, \cdot)$ is determined by the definition of the projection function $\phi(\cdot)$. Formally, the function $\kappa(\cdot, \cdot)$ is defined as the *kernel function* in *SVM* (it has also been widely applied in many other learning algorithms). If the calculation cost of $\kappa(\cdot, \cdot)$ is lower than that of $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$, based on the *kernel function*, the overall learning cost of non-linear *SVM* will be reduced greatly.

Example 2.4 Let $\phi([x_1, x_2]^\top) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]^\top$ be a function which projects the data instances from a two-dimensional feature space to a three-dimensional feature space. Let $\mathbf{x} = [x_1, x_2]^\top$ and $\mathbf{z} = [z_1, z_2]^\top$ be two feature vectors, we can represent the inner product of $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$ as

$$\begin{aligned} \phi(\mathbf{x})^\top \phi(\mathbf{z}) &= \left[x_1^2, \sqrt{2}x_1x_2, x_2^2\right] \left[z_1^2, \sqrt{2}z_1z_2, z_2^2\right]^\top \\ &= x_1^2 z_1^2 + 2x_1x_2z_1z_2 + x_2^2 z_2^2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2. \end{aligned} \tag{2.32}$$

Computing the inner product with the kernel function $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2$ involves an inner product operation in a two-dimensional feature space (i.e., $\mathbf{x}^\top \mathbf{z} = [x_1, x_2][z_1, z_2]^\top$) and a real-value square operation (i.e., $(\mathbf{x}^\top \mathbf{z})^2$), whose cost is lower than that introduced in feature vector projection and the

jwzhanggy@gmail.com

inner product operation in the 3-dimension space with equation, i.e., $\phi(\mathbf{x})$, $\phi(\mathbf{z})$ and $\phi(\mathbf{x})^\top \phi(\mathbf{z}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2][z_1^2, z_2^2, \sqrt{2}z_1z_2]^\top$.

The advantages of applying the kernel function in training non-linear SVM will be more significant in the case where $d \ll D$. Normally, instead of defining the projection function $\phi(\cdot)$, we can define the kernel function $\kappa(\cdot, \cdot)$ directly. Some frequently used kernel functions in *SVM* are listed as follows:

- **Polynomial Kernel**: $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + \theta)^d$ $(d \geq 1)$.
- **Gaussian RBF Kernel**: $\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right)$ $(\sigma > 0)$.
- **Laplacian Kernel**: $\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|}{2\sigma}\right)$ $(\sigma > 0)$.
- **Sigmoid Kernel**: $\kappa(\mathbf{x}, \mathbf{z}) = \tanh(\beta \mathbf{x}^\top \mathbf{z} + \theta)$ $(\beta > 0, \theta < 0)$.

Besides these aforementioned functions, there also exist many other *kernel functions* used in either *SVM* or the other learning models, which will not be introduced here.

2.4 Supervised Learning: Regression

Besides the classification problem, another important category of supervised learning tasks is *regression*. In this section, we will introduce the *regression* learning task, as well as three well-known regression models, i.e., *linear regression* [55], *Lasso*, [50] and *Ridge* [20], respectively.

2.4.1 Regression Learning Task

Regression differs from *classification* tasks in the domain of labels. Instead of inferring the pre-defined classes that the data instances belong to, *regression* tasks aim at predicting some real-value attributes for the data instances, like the box office of movies, price of stocks, and population of countries. Formally, given the training data $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i = [x_{i,1}, x_{i,1}, \ldots, x_{i,d}]^\top \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, *regression* tasks aim at building a model that can predict the real-value labels y_i based on the feature vector $\mathbf{x}_i$ representation of the data instances. In this section, we will take the regression models which combine the features linearly as an example, whose predicted label can be represented as

$$\hat{y}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \cdots + w_d x_{i,d}, \tag{2.33}$$

where $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_d]^\top$ denotes the weight and bias variables of the regression model.

Generally, by minimizing the difference between the predicted labels and the ground truth labels, we can learn the parameter $\mathbf{w}$ in the model. Depending on the loss objective function representations, three different regression models, i.e., *linear regression* [55], *Lasso* [50], and *Ridge* [20], will be introduced as follows.

2.4.2 Linear Regression

Given the training set $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, the linear regression model adopts the mean square error as the loss function, which computes the average square error between the

jwzhanggy@gmail.com

prediction labels and the ground-truth labels of the training data instances. Formally, the optimal parameter $\mathbf{w}^*$ can be represented as

$$\begin{aligned}\mathbf{w}^* &= \arg_{\mathbf{w}} \min E(\hat{\mathbf{y}}, \mathbf{y}) \\ &= \arg_{\mathbf{w}} \min \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \\ &= \arg_{\mathbf{w}} \min \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 ,\end{aligned} \tag{2.34}$$

where $\mathbf{X} = \left[\bar{\mathbf{x}}_1^\top, \bar{\mathbf{x}}_2^\top, \ldots, \bar{\mathbf{x}}_n^\top\right] \in \mathbb{R}^{n\times(d+1)}$ and $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$ denote the feature matrix and label vectors of the training instances. In the representation, vector $\bar{\mathbf{x}}_i = [x_{i,1}, x_{i,1}, \ldots, x_{i,d}, 1]^\top \in \mathbb{R}^{(d+1)}$, where a dummy feature $+1$ is appended to the feature vectors so as to incorporate the bias term w_0 also as a feature weight in model learning.

The mean square error used in the *linear regression* model has very a good mathematical property. Meanwhile, the method of computing the minimum loss based on the mean square error is also called the *least square method*. In linear regression, the *least square method* aims at finding a hyperplane, the sum of distance between which and the training instances can be minimized. The above objective function can be resolved by making derivative of the error term regarding the parameter $\mathbf{w}$ equal to 0, and we can have

$$\begin{aligned}&\frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{w}} = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \\ &\Rightarrow \mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y},\end{aligned} \tag{2.35}$$

Here, to obtain the closed-form optimal solution of $\mathbf{w}^*$, it needs to involve matrix inverse operation of $\mathbf{X}^\top\mathbf{X}$. Depending on whether $\mathbf{X}^\top\mathbf{X}$ is invertible or not, there will be different solutions to the above objective function:

- If matrix $\mathbf{X}^\top\mathbf{X}$ is of full rank or $\mathbf{X}^\top\mathbf{X}$ is positive definite, we have

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}. \tag{2.36}$$

- However, in the case that $\mathbf{X}^\top\mathbf{X}$ is not full rank, there will be multiple solutions to the above objective function. For the linear regression model, all these solutions will lead to the minimum loss. Meanwhile, in some cases, there will be some preference about certain types of parameter $\mathbf{w}$, which can be represented as the regularization term added to the objective function, like the *Lasso* and *Ridge* regression models to be introduced as follows.

2.4.3 Lasso

Lasso [50] is also a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent. For this reason, *Lasso* and its variants are

jwzhanggy@gmail.com

fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero weights. Mathematically, the objective function of *Lasso* can be represented as

$$\arg_{\mathbf{w}} \min \frac{1}{2n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \alpha \cdot \|\mathbf{w}\|_1 , \tag{2.37}$$

where the coefficient vector $\mathbf{w}$ is regularized by its L_1-norm, i.e., $\|\mathbf{w}\|_1$.

Considering that the L_1-norm regularizer $\|\mathbf{w}\|_1$ is not differentiable, and no closed-form solution exists for the above objective function. Meanwhile, a wide variety of techniques from convex analysis and optimization theory have been developed to extremize such functions, which include the subgradient methods [10], least-angle regression (LARS) [11], and proximal gradient methods [32]. As to the choice of scalar α, it can be selected based on a validation set of the data instances.

In addition to the regression models in the linear form, there also exist a large number of regression models in the high-order polynomial representation, as well as more complicated representations. For more information about the other regression models, their learning approaches and application scenarios, please refer to [41] for more detailed information.

2.4.4 Ridge

Ridge [20] addresses some of the problems of ordinary least squares (OLS) by imposing a penalty on the size of coefficient $\mathbf{w}$. The ridge coefficients minimize a penalized residual sum of squares, i.e.,

$$\arg_{\mathbf{w}} \min \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \alpha \cdot \|\mathbf{w}\|_2^2 , \tag{2.38}$$

where $\alpha > 0$ denotes the complexity parameter used to control the size shrinkage of coefficient vector $\mathbf{w}$.

Generally, a larger α will lead to a greater shrinkage of coefficient $\mathbf{w}$ and thus the coefficient will be more robust to collinearity. Learning of the optimal coefficient $\mathbf{w}^*$ of the *ridge regression* model is the same as the learning process of the *linear regression* model, and the optimal $\mathbf{w}^*$ can be represented as

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \alpha \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{2.39}$$

Here, we observe that the optimal solution depends on the choice of the scalar α a lot. As $\alpha \to 0$, the optimal solution $\mathbf{w}^*$ will be equal to the solution to the *linear regression* model, while as $\alpha \to \infty$, $\mathbf{w}^* \to \mathbf{0}$. In the real-world practice, the optimal scalar α can usually be selected based on the validation set partitioned from cross validation.

2.5 Unsupervised Learning: Clustering

Different from the supervised learning tasks, in many cases, no supervision information is available to guide the model learning and the data instances available have no specific object labels at all. The tasks of learning some inner rules and patterns from such unlabeled data instances are formally called the unsupervised learning tasks, which actually provide some basic information for further data analysis. Unsupervised learning involves very diverse learning tasks, among which *clustering* can be the main research focus and has very broad applications. In this section, we will introduce the

jwzhanggy@gmail.com

clustering learning tasks and cover three well-used clustering algorithms, including *K-Means* [16], *DBSCAN* [12], and *Mixture-of-Gaussian* [40].

2.5.1 Clustering Task

Clustering tasks aim at partitioning the data instances into different groups, where instances in each cluster are more similar to each other compared with those from other clusters. For instance, movies can be divided into different genres (e.g., comedy vs tragedy vs Sci-fi), countries can be partitioned into different categories (e.g., developing countries vs developed countries), a basket of fruits can be grouped into different types (e.g., apple vs orange vs banana). Generally, in real-world applications, clustering can be used as a single data mining procedure or a data pre-processing step to divide the data instances into different categories.

Formally, in the clustering tasks, the unlabeled data instances can be represented as set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where each data instance can be represented as a d-dimensional feature vector $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^\top$. Clustering tasks aim at partitioning the data instances into k disjoint groups $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, where $\mathcal{C}_i \subset \mathcal{D}, \forall i \in \{1, 2, \ldots, k\}$, while $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset, \forall i, j \in \{1, 2, \ldots, k\}, i \neq j$, and $\bigcup_i \mathcal{C}_i = \mathcal{D}$. Generally, we can use the cluster index $y_i \in \{1, 2, \ldots, k\}$ to indicate the cluster ID that instance $\mathbf{x}_i$ belongs to, and the cluster labels of all the data instances in $\mathcal{D}$ can be represented as a vector $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$.

In clustering tasks, a formal definition of the distance among the data instances is required, and many existing clustering algorithms heavily rely on the data instance distance definition. Given a function $dist(\cdot, \cdot) : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$, it can be defined as a distance measure iff it can meet the following properties:

- **Non-negative**: $dist(\mathbf{x}_i, \mathbf{x}_j) \geq 0$,
- **Identity**: $dist(\mathbf{x}_i, \mathbf{x}_j) = 0$ iff. $\mathbf{x}_i = \mathbf{x}_j$,
- **Symmetric**: $dist(\mathbf{x}_i, \mathbf{x}_j) = dist(\mathbf{x}_j, \mathbf{x}_i)$,
- **Triangular Inequality**: $dist(\mathbf{x}_i, \mathbf{x}_j) \leq dist(\mathbf{x}_i, \mathbf{x}_k) + dist(\mathbf{x}_k, \mathbf{x}_j)$.

Given two data instances featured by vectors $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^\top$ and $\mathbf{x}_j = [x_{j,1}, x_{j,2}, \ldots, x_{j,d}]^\top$, a frequently used distance measure is the *Minkowski distance*:

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^{d} |x_{i,k} - x_{j,k}|^p \right)^{\frac{1}{p}}. \tag{2.40}$$

The *Minkowski distance* is a general distance representation, and it covers various well-known distance measures depending on the selection of value p:

- **Manhattan Distance**: in the case that $p = 1$, we have $dist_m(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{d} |x_{i,k} - x_{j,k}|$.
- **Euclidean Distance**: in the case that $p = 2$, we have $dist_e(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^{d} |x_{i,k} - x_{j,k}|^2 \right)^{\frac{1}{2}}$.
- **Chebyshev Distance**: in the case that $p \to \infty$, we have $dist_c(\mathbf{x}_i, \mathbf{x}_j) = \max(\{|x_{i,k} - x_{j,k}|\}_{k=1}^{d})$.

Besides the *Minkowski distance*, there also exist many other distance measures, like VDM (value difference metric), which works well for the unordered attributes. In many cases, different attributes actually play a different role in the distance calculation and should be weighted differently. Some

jwzhanggy@gmail.com

distance metrics assign different weights to the attributes to differentiate them. For instance, based on the *Minkowski distance*, we can define the weighted *Minkowski distance* to be

$$dist_w = \left(\sum_{k=1}^{d} w_k \cdot |x_{i,k} - x_{j,k}|^p \right)^{\frac{1}{p}}, \tag{2.41}$$

where $w_i \geq 0, \forall i \in \{1, 2, \ldots, d\}$ and $\sum_{k=1}^{d} w_i = 1$.

Generally, clustering tasks aim at grouping similar data instances (i.e., with a smaller distance) into the same cluster, while different data instances (i.e., with a larger distance) into different clusters. Meanwhile, depending on the data instance-cluster belonging relationships, the clustering tasks can be categorized into two types: (1) *hard clustering*: each data instance belongs to exact one cluster, and (2) *soft clustering*: data instances can belong to multiple clusters with certain confidence scores. In the following part of this section, we will introduce two *hard clustering* algorithms: *K-Means* [16] and *DBSCAN* [12], and one *soft clustering* algorithm: *Mixture-of-Gaussian* [40].

2.5.2 K-Means

In the unlabeled data set, the data instances usually belong to different groups. In each of the groups, there can exist a representative data instance which can outline the characteristics of the group internal data instances. Therefore, after identifying the prototype data instances, the clusters can be discovered easily. Meanwhile, how to define the cluster prototypes is an open problem, and many different algorithms have been proposed already, among which *K-Means* [16] is one of the most well-known clustering algorithms.

Let $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be the set of unlabeled data instances, and *K-Means* algorithm aims at partitioning $\mathcal{D}$ into k clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$. For the data instances belonging to each cluster, the cluster prototype data instance is defined as the center of the cluster in *K-Means*, i.e., the mean of the data instance vectors. For instance, for cluster $\mathcal{C}_i$, its center can be represented as follows formally:

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{x} \in \mathcal{C}_i} \mathbf{x}, \forall \mathcal{C}_i \in \mathcal{C}. \tag{2.42}$$

There can exist different ways to partition the data instances, to select the optimal one from which a clear definition of clustering quality will be required. In *K-Means*, the quality of the clustering result $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ can be measured by the introduced square of the Euclidean distance between the data instances and the prototypes, i.e.,

$$E(\mathcal{C}) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathcal{C}_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|_2^2. \tag{2.43}$$

Literally, the clustering result which can minimize the above loss term will bring about a very good partition about the data instances. However, identifying the optimal prototypes and the cluster partition by minimizing the above square loss is a very challenging problem, which is actually NP-hard. To address such a challenge, the *K-Means* algorithm adopts a greedy strategy to find the prototypes and cluster division iteratively. The pseudo-code of the *K-Means* algorithm is available in Algorithm 2. As shown in the algorithm, in the first step, *K-Means* algorithm first randomly picks k data instances

jwzhanggy@gmail.com

Algorithm 2 KMeans

Require: Data Set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$;
Cluster number k.
Ensure: Clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$
Randomly pick k data instances from $\mathcal{D}$ as the prototype data instances $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_k\}$
Stable-tag = $False$
while Stable-tag == $False$ **do**
  Let $\mathcal{C}_i = \emptyset, \forall i = 1, 2, \cdots, k$
  for data instance $\mathbf{x}_i \in \mathcal{D}$ **do**
    $\lambda = \arg_{j \in \{1,2,\cdots,k\}} \min dist(\mathbf{x}_i - \boldsymbol{\mu}_j)$
    $\mathcal{C}_\lambda = \mathcal{C}_\lambda \cup \{\mathbf{x}_i\}$
  end for
end while
for $j \in \{1, 2, \cdots, k\}$ **do**
  compute the new prototype data instance $\boldsymbol{\mu}'_j = \frac{1}{|\mathcal{C}_j|} \sum_{\mathbf{x} \in \mathcal{C}_j} \mathbf{x}$
  if $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_k\} == \{\boldsymbol{\mu}'_1, \boldsymbol{\mu}'_2, \cdots, \boldsymbol{\mu}'_k\}$ **then**
    Stable-tag = $True$
  else
    $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_k\} = \{\boldsymbol{\mu}'_1, \boldsymbol{\mu}'_2, \cdots, \boldsymbol{\mu}'_k\}$
  end if
end for
Return $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$

from $\mathcal{D}$ as the prototypes, and the data instances will be assigned to the clusters centered by these prototypes. For each data instance $\mathbf{x}_i \in \mathcal{D}$, *K-Means* selects the prototype $\boldsymbol{\mu}_j$ closest to $\mathbf{x}_i$, and adds $\mathbf{x}_i$ to cluster $\mathcal{C}_j$, where the distance measure $dist(\mathbf{x}_i - \boldsymbol{\mu}_j)$ can be the Euclidean distance between $\mathbf{x}_i$ and $\boldsymbol{\mu}_j$ (or the other distance measures we define before). Based on the obtained clustering result, *K-Means* re-computes the prototype centers of each newly generated clusters, which will be treated as the new prototypes. If in the current round, none of the prototypes changes, *K-Means* will return the current data instance partition as the final clustering result.

Example 2.5 In Fig. 2.13, we use an example to further illustrate the *K-Means* algorithm. For instance, given a group of data instance as shown in Fig. 2.13a, *K-Means* first randomly selects three points as the centers from the space as shown in Fig. 2.13b. All the data instances will be assigned to their nearest centers, and they will form three initial clusters (in three different colors) as illustrated in Fig. 2.13c. Based on the initial clustering results, *K-Means* recomputes the centers as the average of the cluster internal data instances in Fig. 2.13d, and further partitions the data set with the new centers in Fig. 2.13e. As the results converge and there exist no changes for partition, the *K-Means* algorithm will stop and output the final data partition results.

The *K-Means* algorithm is very powerful in handling data sets from various areas, but may also suffer from many problems. According to the algorithm descriptions and the example, we observe that the performance of *K-Means* is quite sensitive to the initial selection of prototypes, especially in the case when the number of data instance is not so large. Furthermore, in *K-Means*, the hyperparameter k needs to be selected beforehand, which is actually very challenging to infer from the data without knowing the distributions of the data. It will also introduce lots of parameter tuning works for *K-Means*. Next, we will introduce a density-based clustering algorithm, named *DBSCAN*, which doesn't need the cluster number parameter k as the input.

jwzhanggy@gmail.com

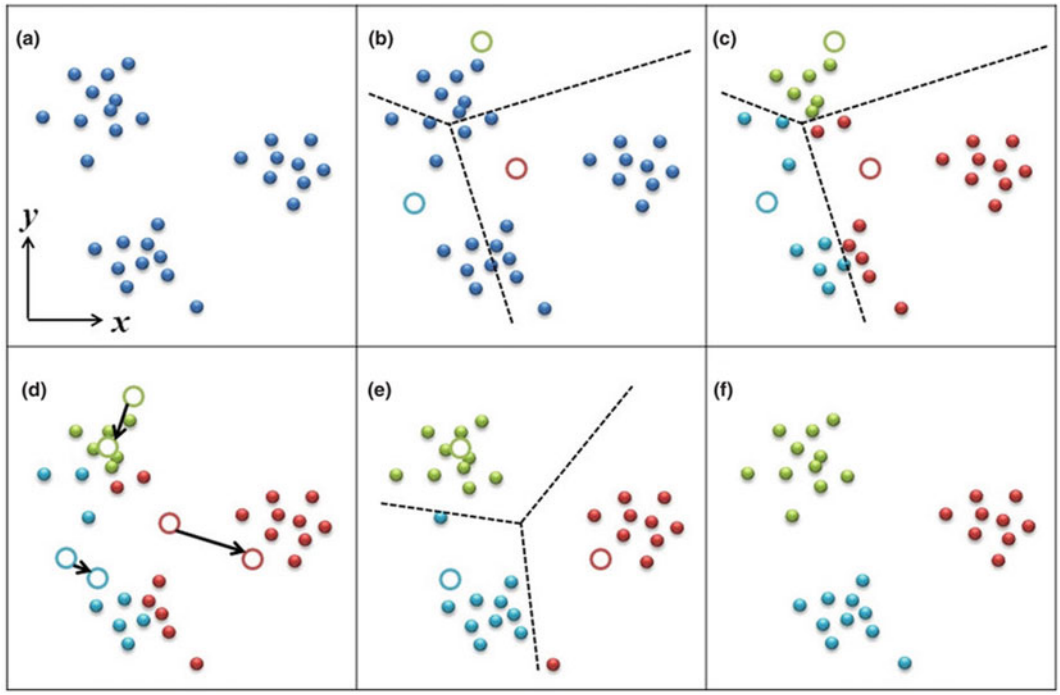

Fig. 2.13 An example of the K-Means algorithm in partitioning data instances ((**a**) input data instances; (**b**)–(**f**) steps of K-Means in clustering the data input)

2.5.3 DBSCAN

DBSCAN [12] is short for *density-based spatial clustering of applications with noise*, and it is a density-based clustering algorithm, which assumes that the cluster number and cluster structure can be revealed by the data instance distribution density. Generally, density-based clustering algorithms are based on the connectivity relationships among data instances, via which data instances can keep expanding until achieving the final clustering results. To characterize the connectivity relationships among data instances, *DBSCAN* introduces a concept called *neighborhood* parameterized by (ϵ, η). Given a data set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ involving n data instances, several important concepts used in *DBSCAN* are defined as follows:

- **ϵ-Neighborhood**: For data instance $\mathbf{x}_i \in \mathcal{D}$, its ϵ-neighborhood represents a subset of data instances from $\mathcal{D}$, with a distance shorter than ϵ from $\mathbf{x}_i$, i.e., $N_\epsilon(\mathbf{x}_i) = \{\mathbf{x}_j \in \mathcal{D} | dist(\mathbf{x}_i, \mathbf{x}_j) \leq \epsilon\}$ (function $dist(\cdot, \cdot)$ denotes the Euclidean distance by default).
- **Core Objects**: Based on the data instances and their corresponding ϵ-neighborhoods, the core objects denote the data instances whose ϵ-neighborhood contains at least η data instances. In other words, data instance $\mathbf{x}_i$ is a core object iff $|N_\epsilon(\mathbf{x}_i)| \geq \eta$.
- **Directly Density-Reachable**: If data instance $\mathbf{x}_j$ lies in the ϵ-neighborhood of $\mathbf{x}_i$, $\mathbf{x}_j$ is said to be *directly density-reachable* from $\mathbf{x}_i$, i.e., $\mathbf{x}_i \rightarrow \mathbf{x}_j$.
- **Density-Reachable**: Given two data instances $\mathbf{x}_i$ and $\mathbf{x}_j$, $\mathbf{x}_j$ is said to be *density-reachable* from $\mathbf{x}_i$ iff there exists a path $\mathbf{v}_1 \rightarrow \mathbf{v}_2 \rightarrow \cdots \rightarrow \mathbf{v}_{k-1} \rightarrow \mathbf{v}_k$ connecting $\mathbf{x}_i$ with $\mathbf{x}_j$, where $\mathbf{v}_l \in \mathcal{D}$ and $\mathbf{v}_1 = \mathbf{x}_i, \mathbf{v}_k = \mathbf{x}_j$. Sequence $\mathbf{v}_l \rightarrow \mathbf{v}_{l+1}$ represents that $\mathbf{v}_{l+1}$ is *directly density-reachable* from $\mathbf{v}_l$.

jwzhanggy@gmail.com

Algorithm 3 DBSCAN

Require: Data Set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$;
Neighborhood parameters (ϵ, η).
Ensure: Clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$
Initialize core object set $\Omega = \emptyset$
for $\mathbf{x}_i \in \mathcal{D}$ **do**
  Obtain the ϵ-neighborhood of $\mathbf{x}_i$: $N_\epsilon(\mathbf{x}_i)$
  if $|N_\epsilon(\mathbf{x}_i)| \geq \eta$ **then**
    $\Omega = \Omega \cup \{\mathbf{x}_i\}$
  end if
end for
Initialize cluster number $k = 0$ and unvisited data instance set $\Gamma = \mathcal{D}$
while $\Omega \neq \emptyset$ **do**
  Keep a record of current unvisited data instances $\Gamma' = \Gamma$
  Randomly select a data instance $\mathbf{o} \in \Omega$ to initialize a queue $Q = (\mathbf{o})$
  $\Gamma = \Gamma \setminus \{\mathbf{o}\}$
  while $Q \neq \emptyset$ **do**
    Get the head data instance $\mathbf{q} \in Q$
    if $|N_\epsilon(\mathbf{q})| \geq \eta$ **then**
      Add $N_\epsilon(\mathbf{q}) \cap \Gamma$ into queue Q
      $\Gamma = \Gamma \setminus N_\epsilon(\mathbf{q})$
    end if
  end while
  $k = k + 1$, and generate cluster $\mathcal{C}_k = \Gamma' \setminus \Gamma$
  $\Omega = \Omega \setminus \mathcal{C}_k$
end while
Return $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$

- **Density-Connected**: Given two data instance $\mathbf{x}_i$ and $\mathbf{x}_j$, $\mathbf{x}_i$ is said to be *density-connected* to $\mathbf{x}_j$, iff $\exists \mathbf{x}_k \in \mathcal{D}$ that $\mathbf{x}_i$ and $\mathbf{x}_j$ are both *density-reachable* from $\mathbf{x}_k$.

DBSCAN aims at partitioning the data instances into several densely distributed regions. In *DBSCAN*, a cluster denotes a maximal subset of data instances, in which the data instances are *density-connected*. Formally, a subset $\mathcal{C}_k \subset \mathcal{D}$ is a cluster detected by *DBSCAN* iff both the following two properties hold:

- **Connectivity**: $\forall \mathbf{x}_i, \mathbf{x}_j \in \mathcal{C}_k$, $\mathbf{x}_i$ and $\mathbf{x}_j$ are *density-connected*.
- **Maximality**: $\forall \mathbf{x}_i \in \mathcal{C}_k, \mathbf{x}_j \in \mathcal{D}$, $\mathbf{x}_j$ is *density-reachable* from $\mathbf{x}_i$ implies $\mathbf{x}_j \in \mathcal{C}_k$.

To illustrate how *DBSCAN* works, we provide its pseudo-code in Algorithm 3. According to the algorithm, at the beginning, *DBSCAN* selects a set of core objects as the seeds, from which *DBSCAN* expands the clusters based on the ϵ-neighborhood concept introduced before iteratively to find the *density-reachable* clusters. Such a process continues until all the core objects have been visited. The performance of *DBSCAN* depends on the selection order of the seed data instances a lot, and different selection orders will lead to totally different clustering results. *DBSCAN* doesn't need the cluster number as the input parameter, but requires (ϵ, η) to define the *ϵ-neighborhood* and *core objects* of the data instances.

Both the *K-Means* and *DBSCAN* algorithms are actually *hard clustering* algorithms, where instances are partitioned into different groups and each data instance only belongs to one single cluster. Besides such a type of clustering algorithms, there also exist some other clustering algorithms that allow data instances to belong to multiple clusters at the same time, like the *Mixture-of-Gaussian* algorithm to be introduced in the next subsection.

jwzhanggy@gmail.com

2.5.4 Mixture-of-Gaussian Soft Clustering

Different from the previous two clustering algorithms, the *Mixture-of-Gaussian* clustering algorithm [40] uses probability to model the cluster prototypes. Formally, in the d-dimensional feature space, given a feature vector $\mathbf{x}$ that follows a certain distribution, e.g., the Gaussian distribution, its probability density function can be represented as

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \times \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{2.44}$$

where $\boldsymbol{\mu}$ denotes the d-dimensional mean vector and $\boldsymbol{\Sigma}$ is a $d \times d$ covariance matrix.

In the *Mixture-of-Gaussian* clustering algorithm, each cluster is represented by a Gaussian distribution, where $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ can denote the parameters of the jth Gaussian distribution. Formally, the probability density function of the *Mixture-of-Gaussian* distribution can be represented as

$$p_M(\mathbf{x}) = \sum_{j=1}^{k} \alpha_j \cdot p(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j), \tag{2.45}$$

where α_j denotes the mixture coefficient and $\sum_{j=1}^{k} \alpha_j = 1$.

The data instances in the training set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ are assumed to be generated from the *Mixture-of-Gaussian* distribution. Given a data instance $\mathbf{x}_i \in \mathcal{D}$, the probability that the data instance is generated by the jth Gaussian distribution (i.e., its cluster label $y_i = j$) can be represented as

$$\begin{aligned} p_M(y_i = j|\mathbf{x}_i) &= \frac{p(y_i = j) \cdot p(\mathbf{x}_i|y_i = j)}{p_M(\mathbf{x}_i)} \\ &= \frac{\alpha_j \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{l=1}^{k} \alpha_l \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}. \end{aligned} \tag{2.46}$$

Meanwhile, in the *Mixture-of-Gaussian* clustering algorithm, a set of parameters $\{\boldsymbol{\mu}_j\}_{j=1}^{k}$, $\{\boldsymbol{\Sigma}_j\}_{j=1}^{k}$ and $\{\alpha_j\}_{j=1}^{k}$ are involved, which can be inferred from the data. Formally, based on the data set $\mathcal{D}$, we can represent its log-likelihood for the data instances to be distributed by following the *Mixture-of-Gaussian* to be

$$\begin{aligned} \mathcal{L}(\mathcal{D}) &= \ln\left(\prod_{i=1}^{n} p_M(\mathbf{x}_i)\right) \\ &= \sum_{i=1}^{n} \ln\left(\sum_{j=1}^{k} \alpha_j \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right). \end{aligned} \tag{2.47}$$

For the parameters which can maximize the above log-likelihood function will be the optimal solution to the *Mixture-of-Gaussian* model.

jwzhanggy@gmail.com

The EM (Expectation Maximization) algorithm can be applied to learn the optimal parameters for the *Mixture-of-Gaussian* clustering algorithm. By taking the derivatives of the objective function with regard to $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ and making them equal to 0, we can have

$$\begin{cases} \boldsymbol{\mu}_j & = \frac{\sum_{i=1}^{n} p_M(y_i=j|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{n} p_M(y_i=j|\mathbf{x}_i)}, \\ \boldsymbol{\Sigma}_j & = \frac{\sum_{i=1}^{n} p_M(y_i=j|\mathbf{x}_i)(\mathbf{x}_i-\boldsymbol{\mu}_j)(\mathbf{x}_i-\boldsymbol{\mu}_j)^\top}{\sum_{i=1}^{n} p_M(y_i=j|\mathbf{x}_i)}. \end{cases} \tag{2.48}$$

As to the weights $\{\alpha_j\}_{j=1}^k$, besides the objective function, there also exist some constraints $\alpha_j \geq 0$ and $\sum_{j=1}^k \alpha_j = 1$ on them. We can represent the Lagrange function of the objective function to be

$$\begin{aligned} L(\{\alpha_j\}_{j=1}^k, \lambda) = & \sum_{i=1}^{n} \ln \left(\sum_{j=1}^{k} \alpha_j \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \right) \\ & + \lambda \left(\sum_{j=1}^{k} \alpha_j - 1 \right). \end{aligned} \tag{2.49}$$

By making the derivative of the above function to α_j equal to 0, we can have

$$\sum_{i=1}^{n} \frac{p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{l=1}^{k} \alpha_l \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_l)} + \lambda = 0. \tag{2.50}$$

Meanwhile, by multiplying the above equation by α_j and summing up the equations by enumerating all Gaussian distribution prototypes, we have

$$\begin{aligned} & \sum_{j=1}^{k} \sum_{i=1}^{n} \frac{\alpha_j \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{l=1}^{k} \alpha_l \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \sum_{j=1}^{k} \alpha_j \cdot \lambda = 0 \\ & \Rightarrow \sum_{j=1}^{k} \sum_{i=1}^{n} p_M(y_i = j|\mathbf{x}_i) + \lambda = 0 \\ & \Rightarrow \lambda = -n. \end{aligned} \tag{2.51}$$

Furthermore, we have

$$\begin{aligned} & \sum_{i=1}^{n} \frac{\alpha_j \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{l=1}^{k} \alpha_l \cdot p(\mathbf{x}_i|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \lambda \cdot \alpha_j = 0 \\ & \Rightarrow \sum_{i=1}^{n} p_M(y_i = j|\mathbf{x}_i) + \lambda \cdot \alpha_j = 0 \\ & \Rightarrow \alpha_j = \frac{1}{-\lambda} \sum_{i=1}^{n} p_M(y_i = j|\mathbf{x}_i) \\ & \qquad = \frac{1}{n} \sum_{i=1}^{n} p_M(y_i = j|\mathbf{x}_i). \end{aligned} \tag{2.52}$$

jwzhanggy@gmail.com

Algorithm 4 Mixture-of-Gaussian

Require: Data Set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$;
Gaussian distribution prototype number k.
Ensure: Clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$
Initialize the parameters $\{\alpha_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\}_{j=1}^k$
$Converge - tag = False$
while $Converge - tag == False$ **do**
  for $i \in \{1, 2, \cdots, n\}$ **do**
    Calculate the posterior probability $p_M(y_i = j|\mathbf{x}_i)$
  end for
  for $j \in \{1, 2, \cdots, k\}$ **do**
    Calculate new mean vector $\boldsymbol{\mu}'_j = \frac{\sum_{i=1}^n p_M(y_i=j|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^n p_M(y_i=j|\mathbf{x}_i)}$
    Calculate new covariance matrix $\boldsymbol{\Sigma}'_j = \frac{\sum_{i=1}^n p_M(y_i=j|\mathbf{x}_i)(\mathbf{x}_i-\boldsymbol{\mu}_j)(\mathbf{x}_i-\boldsymbol{\mu}_j)^\top}{\sum_{i=1}^n p_M(y_i=j|\mathbf{x}_i)}$
    Calculate new weight $\alpha'_j = \frac{1}{n}\sum_{i=1}^n p_M(z_i = j|\mathbf{x}_i)$
  end for
  if $\{\alpha_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\}_{j=1}^k == \{\alpha'_j, \boldsymbol{\mu}'_j, \boldsymbol{\Sigma}'_j)\}_{j=1}^k$ **then**
    $Converge - tag = True$
  else
    $\{\alpha_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\}_{j=1}^k = \{\alpha'_j, \boldsymbol{\mu}'_j, \boldsymbol{\Sigma}'_j)\}_{j=1}^k$
  end if
end while
Initialize $\mathcal{C} = \{\mathcal{C}_j\}_{j=1}^k$, where $\mathcal{C}_j = \emptyset$
for $\mathbf{x}_i \in \mathcal{D}$ **do**
  Determine the cluster label $y_i^* = \arg_{y\in\{1,2,\cdots,k\}} p_M(y_i = y|\mathbf{x}_i)$
  $\mathcal{C}_{y_i^*} = \mathcal{C}_{y_i^*} \cup \{\mathbf{x}_i\}$
end for
Return $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k\}$

According to the analysis, the expected values of $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ are the weighted sum of the mean and covariance of the data instances in the provided data set. Meanwhile, weight α_j is the average posterior probability of data instance belonging to the jth Gaussian distribution. The pseudo-code of parameter learning and cluster inference parts of the *Mixture-of-Gaussian* clustering algorithm is provided in Algorithm 4.

2.6 Artificial Neural Network and Deep Learning

Artificial neural network (ANN) is a computational algorithm aiming at modeling the way that a biological brain solves problems with large clusters of connected biological neurons. Artificial neural network models include both supervised and unsupervised learning models, which can work on both classification and regression tasks. Artificial neural networks as well as the recent deep learning [14] models have achieved a remarkable success in addressing various difficult learning tasks. Therefore, we lift them up as a separate section to talk about. In this part, we will provide a brief introduction to the artificial neural networks, including the basic background knowledge, *perceptron* [30, 42], *multi-layer feed-forward neural network* [48], *error back propagation algorithm* [43–45, 54], and several well-known deep learning models, i.e., *deep autoencoder* [14,53], *deep recurrent neural network* [14], and *deep convolutional neural network* [14, 26, 27].

jwzhanggy@gmail.com

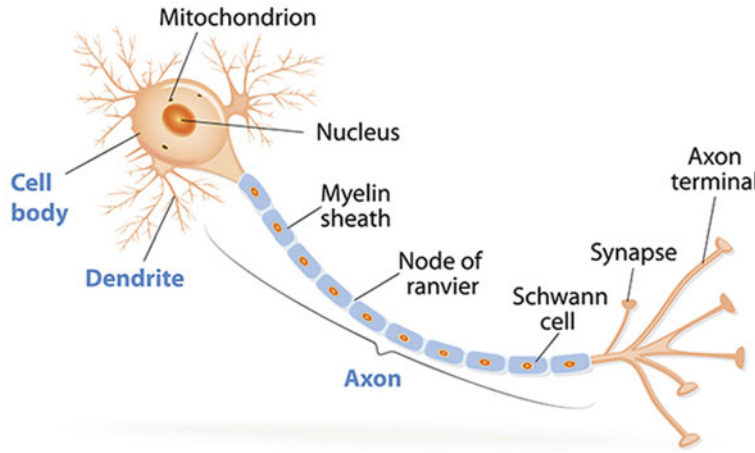

Fig. 2.14 A picture of neuron

2.6.1 Artificial Neural Network Overview

As shown in Fig. 2.14, human brains are composed by a group of *neurons*, which are connected by the *axons* and *dendrites*. In human brains, signal can transmit from a neuron to another one via the *axon* and *dendrite*. The connecting point between *axon* and *dendrite* is called the *synapse*, where human brains can learn new knowledge by changing the connection strength of these *synapses*. Similar to the biological brain network, artificial neural networks are composed of a group of connected artificial neurons, which receive inputs from the other neurons. Depending on the inputs and the activation thresholds, artificial neurons can be either activated to transmit information to the other artificial neuron or stay inactive.

In Fig. 2.15a, we show an example of the classic McCulloch-Pitts (M-P) neuron model [29]. As shown in the model, the neuron takes the inputs $\{x_1, x_2, \ldots, x_n\}$ from n other neurons and its output can be represented as y. The connection weights between other neurons and the current neuron can be denoted as $\{w_1, w_2, \ldots, w_n\}$, and the inherent activating threshold of the current neuron can be represented as θ. The output y of the current neuron depends on both the inputs $\{x_1, x_2, \ldots, x_n\}$, connection weights $\{w_1, w_2, \ldots, w_n\}$, and the activating threshold θ. Formally, the output of the current neuron can be represented as

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right), \tag{2.53}$$

where $f(\cdot)$ is usually called the *activation function*. *Activation function* can project the input signals to the objective output, and many different *activation functions* can be used in the real-world applications, like the *sign function* and *sigmoid function*.

Formally, the *sign function* can be represented as

$$f(x) = \begin{cases} 1, & \text{if } x > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{2.54}$$

In the case where the *sign function* is used as the *activation function*, if the weighted input sum received from the other neurons is greater than the current neuron's threshold, the current neuron will

jwzhanggy@gmail.com

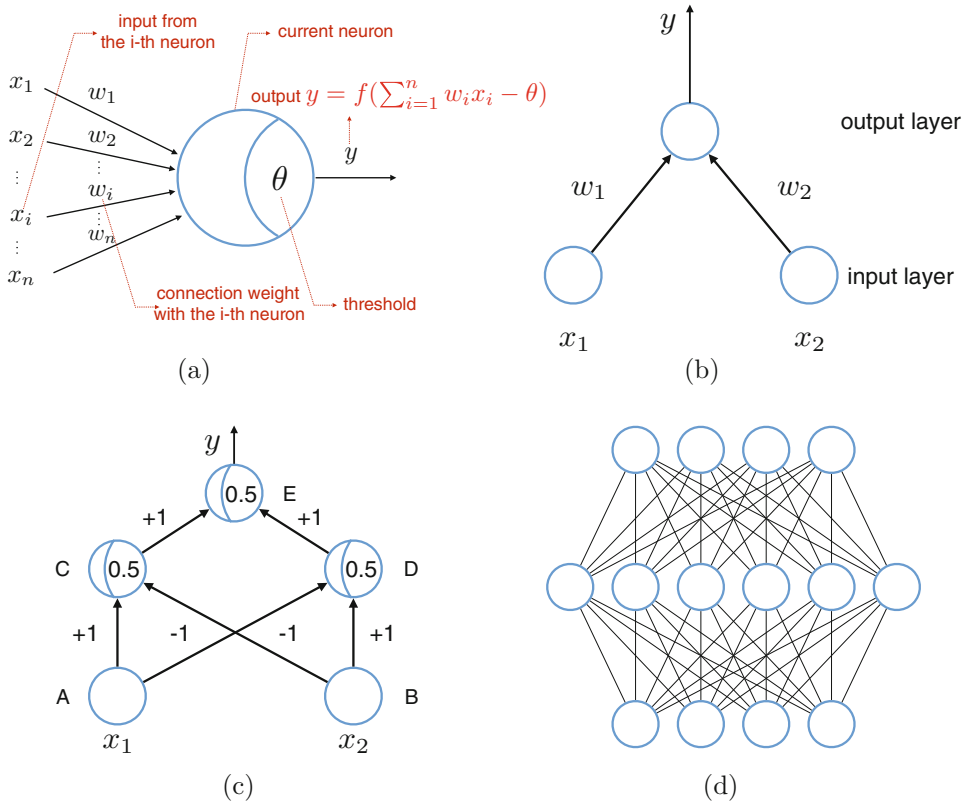

Fig. 2.15 Neural network models: **(a)** McCulloch-Pitts (M-P) neuron model, **(b)** perceptron mode, **(c)** multi-layer perceptron model, and **(d)** multi-layer feed-forward model

be activated and output 1; otherwise, the output will be 0. The *sign function* is very simple, and works well in modeling the active/inactive states of the neurons. However, the mathematical properties, like *discontinuous* and *nonsmooth*, of the *sign function* are not good, which render the *sign function* rarely used in real-world artificial neural network models.

Different from the *signed function*, the *sigmoid function* is a continuous, smooth, and differentiable function. Formally, the *sigmoid function* can be represented as

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.55}$$

which outputs a value in the (0, 1) range for all inputs $x \in \mathbb{R}$. When the *sigmoid function* is used as the *activation function*, if the input is greater than the neuron's *activation threshold*, the output will be greater than 0.5 and will approach 1 as the weighted input sum further increases; otherwise, the output will be smaller than 0.5 and approaches 0 when the weighted input sum further decreases.

2.6.1.1 Perceptron and Multi-Layer Feed-Forward Neural Network

Based on the M-P neuron, several neural network algorithms have already been proposed. In this part, we will introduce two classic artificial neural network models: (1) *perceptron* [30, 42] and (2) *multi-layer feed-forward neural network* [48].

jwzhanggy@gmail.com

The architecture of *perceptron* [30, 42] is shown in Fig. 2.15c. *Perceptron* consists of two layers of neurons: (1) the input layer, and (2) the output layer. The input layer neurons receive external inputs and transmit them to the output layer, while output layer is an M-P neuron which receives the input and uses the *sign function* as the *activation function*. Given the training data, the parameters involved in *perceptron*, like weights $\mathbf{w} = [w_1, w_2, \ldots, w_n]^\top$ and threshold θ, can all be effectively learned. To simplify the learning process, here, we can add one extra dummy input feature "-1" for the M-P neuron whose connection weight can be denoted as θ. In this way, we can unify the activating threshold with the connection weights as $\mathbf{w} = [w_1, w_2, \ldots, w_n, \theta]^\top$. In the learning process, given a training instance $(\mathbf{x}, y)$, the weights of the *perceptron* model can be updated by the following equations until convergence:

$$w_i = w_i + \partial w_i, \tag{2.56}$$

$$\partial w_i = \eta(y - \hat{y})x_i, \tag{2.57}$$

where $\eta \in (0, 1)$ represents the learning rate and $\hat{y}$ denotes output value of *perceptron*.

Perceptron is one of the simplest artificial neural network models, and can only be used to implement some simple linear logical operations, like *and*, *or* and *not*, where the above learning process will converge to the optimal variables.

- **AND** ($x_1 \wedge x_2$): Let $w_1 = w_2 = 1$ and $\theta = 1.9$, we have $y = f(1 \cdot x_1 + 1 \cdot x_2 - 1.9)$, which achieves value $y = 1$ iff $x_1 = x_2 = 1$.
- **OR** ($x_1 \vee x_2$): Let $w_1 = w_2 = 1$ and $\theta = 0.5$, we have $y = f(1 \cdot x_1 + 1 \cdot x_2 - 0.5)$, which achieves value $y = 1$ if $x_1 = 1$ or $x_2 = 1$.
- **NOT** ($\neg x_1$): Let $w_1 = -0.6$, $w_2 = 0$ and $\theta = -0.5$, we have $y = f(-0.6 \cdot x_1 + 0 \cdot x_2 + 0.5)$, which achieves value $y = 1$ if $x_1 = 0$; and value $y = 0$ if $x_1 = 1$.

However, as pointed out by Minsky in [30], *perceptron* cannot handle non-linear operations, like *xor* (i.e., $x_1 \oplus x_2$), as no convergence can be achieved with the above weight updating equations.

To overcome such a problem, *multi-layer perceptron* [48] has been introduced, which can classify the instances that are not linearly separable. Besides the input and output layers, the *multi-layer perceptron* also involves a hidden layer. For instance, to fit the *XOR* function, the *multi-layer perceptron* architecture is shown in Fig. 2.15c, where the connection weights and neuron thresholds are also clearly indicated.

- **XOR** ($x_1 \oplus x_2$): Between the input layer and hidden layer, let the weights $w_{A,C} = w_{B,D} = 1$, $w_{A,D} = w_{B,C} = -1$, thresholds $\theta_C = \theta_D = 0.5$. We can have $y_C = f(1 \cdot x_1 - 1 \cdot x_2 - 0.5)$ and $y_D = f(-1 \cdot x_1 + 1 \cdot x_2 - 0.5)$. Between the hidden layer and output layer, let $w_{C,E} = w_{D,E} = 1$ and threshold $\theta_E = 0.5$, we will have the model output $y = f(1 \cdot y_C + 1 \cdot y_D - 0.5) = f(f(x_1 - x_2 - 0.5) + f(-x_1 + x_2 - 0.5) - 0.5)$. If $x_1 = x_2 = 0$ or $x_1 = x_2 = 1$, we have $y = 0$, while if $x_1 = 1, x_2 = 0$ or $x_1 = 0, x_2 = 1$, we have $y = 1$.

In addition to the *multi-layer perceptron*, a more general multi-layer neural network architecture is shown in Fig. 2.15d, where multiple neuron layers are involved. Between different adjacent layers, the neurons are connected, while those in the non-adjacent layers (i.e., the same layer or skipped layers) are not connected. The artificial neural networks in such an architecture are named as the *feed-forward neural networks* [48], which receive input from the input layer, process the input via the hidden layer, and output the result via the output layer. For the *feed-forward neural networks* shown in Fig. 2.15d, it

jwzhanggy@gmail.com

involves one single hidden layer, which is called the *single hidden-layer feed-forward neural network* in this book. Meanwhile, for the neural networks involving multiple hidden layers, they will be called the *multi-layer neural networks*.

Besides *perceptron*, *feed-forward neural network* models, there also exist a large number of other neural network models with very diverse architectures, like the *cascade-correlation neural network* [13], *Elman neural network* [14], and *Boltzmann neural network* [46]. We will not introduce them here, since they are out of the scope of this textbook. Interested readers may refer to the cited literatures for more information about these models. Generally, for the neural networks involving hidden layers, the model variable learning algorithm will be more challenging and different from *perceptron*. In the following part, we will introduce the well-known *error back propagation algorithm* [43–45, 54] for neural network model learning.

2.6.1.2 Error Back Propagation Algorithm

To this context so far, the most successful learning algorithm for *multi-layer neural networks* is the *Error Back Propagation* (*BP*) algorithm [43–45, 54]. The *BP* algorithm has been shown to work well for many different types of *multi-layer neural network* as well as the recent diverse *deep neural network* models. In the following part, we will use the *single hidden-layer feed-forward neural network* as an example to illustrate the *BP* algorithm.

As shown in Fig. 2.16, in the *single-hidden layer feed-forward neural network*, there exist d different input neurons, q hidden neurons, and l output neurons. Let the data set used for training the model be $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$. Given a data instance featured by vector $\mathbf{x}_i$ as the input, the neural network model will generate a vector $\hat{\mathbf{y}}_i = [\hat{y}_i^1, \hat{y}_i^2, \ldots, \hat{y}_i^l]^\top \in \mathbb{R}^l$ of length l as the output. We will use θ_j to denote the activating threshold of the jth output neuron, and γ_h to denote the activating threshold of the hth hidden neuron. Between the input layer and hidden layer, the connection weight between the ith input neuron and the hth hidden neuron can be represented as $v_{i,h}$. Meanwhile, between the hidden layer and the output layer, the connection weight between the hth hidden neuron and the jth output neuron can be represented as $w_{h,j}$. For the hth hidden neuron, its received input can be represented as $\alpha_h = \sum_{i=1}^{d} v_{i,h} x_i$, and for the jth output neuron, its received input can be denoted as $\beta_j = \sum_{h=1}^{q} w_{h,j} b_h$. Here, we assume the *sigmoid function* is used as the *activation function*, which will project the input to the output in both hidden layer and output layer.

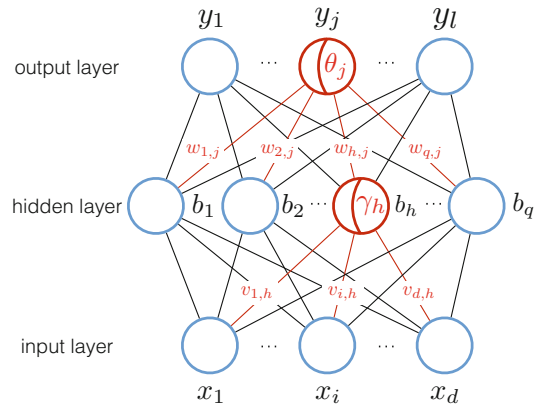

Fig. 2.16 An example of error back propagation algorithm

jwzhanggy@gmail.com

Given a training data instance $(\mathbf{x}_i, \mathbf{y}_i)$, let's assume the output of the neural network to be $\hat{\mathbf{y}}_i = [\hat{y}_i^1, \hat{y}_i^2, \ldots, \hat{y}_i^l]^\top$, and square error is used as the evaluation metric. We can represent the output $\hat{y}_i^j$ and training error as

$$\hat{y}_i^j = f(\beta_j - \theta_j), \tag{2.58}$$

$$E_i = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_i^j - y_i^j)^2, \tag{2.59}$$

where $\frac{1}{2}$ is added in the error function to remove the constant factors when computing the partial derivatives.

BP algorithm proposes to update the model variables with the gradient descent approach. Formally, in gradient descent approach, the model variables will be updated with multiple rounds, and the updating equation in each round can be represented as follows:

$$w \leftarrow w + \eta \cdot \partial w, \tag{2.60}$$

where η denotes the learning rate and it is usually a small constant value. Term ∂w represents the negative gradient of the error function regarding variable w.

We can take $w_{h,j}$ as an example to illustrate the learning process of the BP algorithm. Formally, based on the given training instance $(\mathbf{x}_i, \mathbf{y}_i)$ and the chain rule of derivatives, we can represent the partial derivative of the introduced error regarding variable $w_{h,j}$ as follows:

$$\partial w_{h,j} = -\frac{\partial E_i}{\partial w_{h,j}} = -\frac{\partial E_i}{\partial \hat{y}_i^j} \cdot \frac{\partial \hat{y}_i^j}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{h,j}}. \tag{2.61}$$

Furthermore, according to the definition of E_i, β_j and the property of sigmoid function (i.e., $f'(x) = f(x)(1 - f(x))$), we have

$$\frac{\partial E_i}{\partial \hat{y}_i^j} = \hat{y}_i^j - y_i^j, \tag{2.62}$$

$$\frac{\partial \hat{y}_i^j}{\partial \beta_j} = \hat{y}_i^j(1 - \hat{y}_i^j), \tag{2.63}$$

$$\frac{\partial \beta_j}{\partial w_{h,j}} = b_h. \tag{2.64}$$

Specifically, to simplify the representations, we will introduce a new notation g_j, which denotes

$$g_j = -\frac{\partial E_i}{\partial \hat{y}_i^j} \cdot \frac{\partial \hat{y}_i^j}{\partial \beta_j} = (y_i^j - \hat{y}_i^j) \cdot \hat{y}_i^j(1 - \hat{y}_i^j). \tag{2.65}$$

By replacing the above terms into Eq. (2.61), we have

$$\partial\ w_{h,j} = g_j \cdot b_h. \tag{2.66}$$

jwzhanggy@gmail.com

In a similar way, we can achieve the updating equation for variable θ_j as follows:

$$\theta_j \leftarrow \theta_j + \eta \cdot \partial\theta_j, \text{ where } \partial\theta_j = -g_j. \tag{2.67}$$

Furthermore, by propagating the error back to the hidden layer, we will also be able to update the connection weights between the input layer and hidden layer, as well as the activating threshold of the hidden neurons. Formally, we can represent the updating equation of connection weight $v_{i,h}$ and γ_h as follows:

$$v_{i,h} = v_{i,h} + \eta \cdot \partial v_{i,h}, \text{ where } \partial v_{i,h} = e_h \cdot x_i, \tag{2.68}$$

$$\gamma_h = \gamma_h + \eta \cdot \partial\gamma_h, \text{ where } \partial\gamma_h = -e_h. \tag{2.69}$$

In the above equations, term e_h is an introduced new representation, which can be represented as follows:

$$\begin{aligned} e_h &= -\frac{\partial E_i}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \\ &= -\sum_{j=1}^{l} \frac{\partial E_i}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \\ &= b_h(1 - b_h) \sum_{j=1}^{l} w_{h,j} \cdot g_j. \end{aligned} \tag{2.70}$$

In the updating equation, the learning rate $\eta \in (0, 1)$ can actually be different in updating different variables. Furthermore, according to the updating equations, the term g_j calculated in updating the weights between the hidden layer and output layer will also be used to update the weights between input layer and hidden layer. It is also the reason why the method is called the *Error Back Propagation* algorithm [43–45, 54].

2.6.2 Deep Learning

In recent years, deep learning [14], as a rebranding of artificial neural networks, has become very popular and many different types of deep learning models have been proposed already. These *deep learning* models generally have a large *capacity* (in terms of containing information), as they are usually in a very complex and deep structure involving a large number of variables. For this reason, *deep learning* models are capable to capture very complex projections from the input data to the objective outputs. Meanwhile, due to the availability of massive training data, powerful computational facilities and diverse application domains, training *deep learning* models is becoming a feasible task, which has dominated both academia and industry in these years.

In this part, we will introduce several popular *deep learning* models briefly, which include *deep autoencoder* [14, 53], *deep recurrent neural network* (*RNN*) [14], and *deep convolutional neural network* (*CNN*) [14, 26, 27]. Here, we need to indicate that the *deep autoencoder* model is actually an unsupervised *deep learning* models, and the *deep RNN* and *deep CNN* are both supervised *deep learning* models. The training of *deep learning* models is mostly based on the *error back propagation* algorithm, and we will not cover the *deep learning* model learning materials in this part any more.

jwzhanggy@gmail.com

2.6.2.1 Deep Autoencoder

Deep autoencoder is an unsupervised neural network model, which can learn a low-dimensional representation of the input data via a series of non-linear mappings. *Deep autoencoder* model involves two main steps: encoder and decoder. The encoder step projects the original input to the objective low-dimensional feature space, while the decoder step recovers the latent feature representation to a reconstruction space. In the *deep autoencoder* model, we generally need to ensure that the original feature representation of data instances should be as similar to the reconstructed feature representation as possible.

The architecture of the *deep autoencoder* model is shown in Fig. 2.17a. Formally, let $\mathbf{x}_i$ represent the original input feature representation of a data instance, $\mathbf{y}_i^1, \mathbf{y}_i^2, \ldots, \mathbf{y}_i^o$ be the latent feature representation of the instance at hidden layers $1, 2, \ldots, o$ in the encoder step, and $\mathbf{z}_i \in \mathbb{R}^d$ be the representation in the low-dimensional object space. Formally, the relationship between these vector variables can be represented with the following equations:

$$\begin{cases} \mathbf{y}_i^1 & = \sigma(\mathbf{W}^1 \mathbf{x}_i + \mathbf{b}^1), \\ \mathbf{y}_i^k & = \sigma(\mathbf{W}^k \mathbf{y}_i^{k-1} + \mathbf{b}^k), \forall k \in \{2, 3, \ldots, o\}, \\ \mathbf{z}_i & = \sigma(\mathbf{W}^{o+1} \mathbf{y}_i^o + \mathbf{b}^{o+1}). \end{cases} \tag{2.71}$$

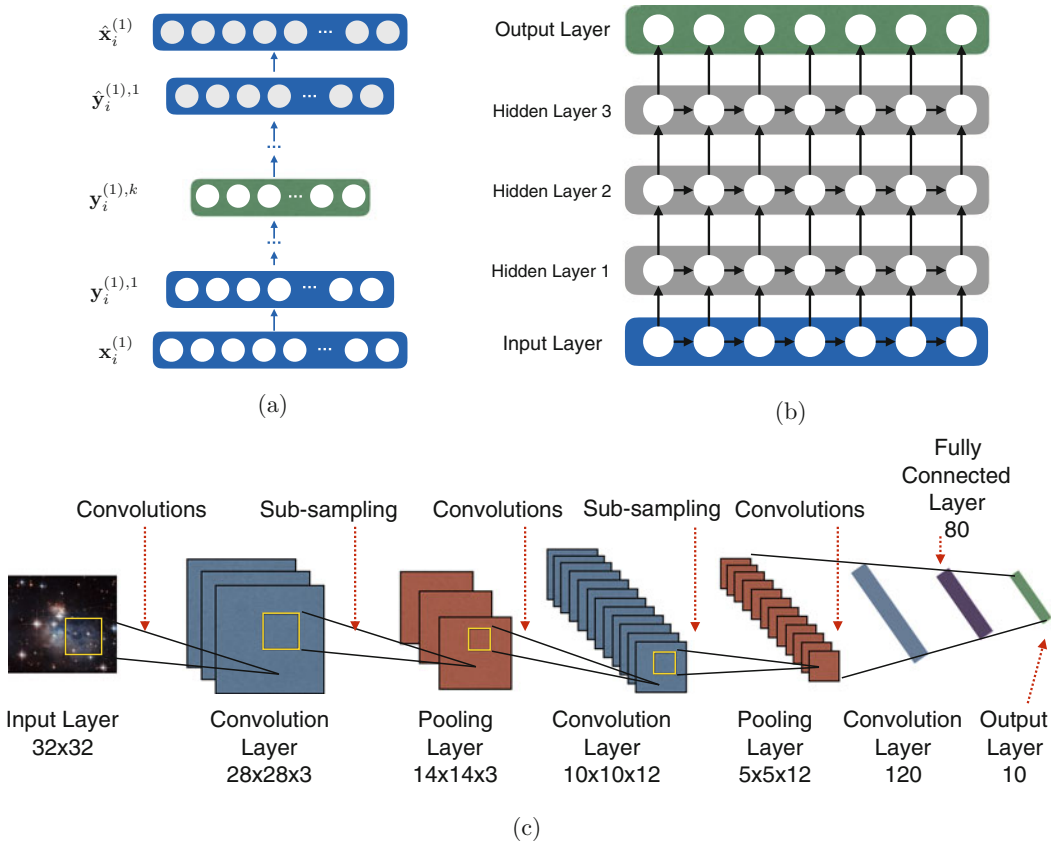

Fig. 2.17 Examples of deep neural network models: **(a)** deep autoencoder model, **(b)** RNN model, **(c)** CNN model

jwzhanggy@gmail.com

Meanwhile, in the decoder step, the input will be the latent feature vector $\mathbf{z}_i$ (i.e., the output of the encoder step) instead, and the final output will be the reconstructed vector $\hat{\mathbf{x}}_i$. The latent feature vectors at each hidden layers in the decoder step can be represented as $\hat{\mathbf{y}}_i^o, \ldots, \hat{\mathbf{y}}_i^2, \hat{\mathbf{y}}_i^1$. The relationship among these vector variables can be denoted as follows:

$$\begin{cases} \hat{\mathbf{y}}_i^o & = \sigma(\hat{\mathbf{W}}^{o+1}\mathbf{z}_i + \hat{\mathbf{b}}^{o+1}), \\ \hat{\mathbf{y}}_i^{k-1} & = \sigma(\hat{\mathbf{W}}^k\hat{\mathbf{y}}_i^k + \hat{\mathbf{b}}^k), \forall k \in \{2, 3, \ldots, o\}, \\ \hat{\mathbf{x}}_i & = \sigma(\hat{\mathbf{W}}^1\hat{\mathbf{y}}_i^1 + \hat{\mathbf{b}}^1). \end{cases} \tag{2.72}$$

The objective of the *deep autoencoder* model is to minimize the loss between the original feature vector $\mathbf{x}_i$ and the reconstructed feature vector $\hat{\mathbf{x}}_i$ of all the instances in the network. Formally, the objective function of the *deep autoencoder* model can be represented as

$$\arg_{\mathbf{W},\hat{\mathbf{W}},\mathbf{b},\hat{\mathbf{b}}} \min \sum_i \left\| \mathbf{x}_i - \hat{\mathbf{x}}_i \right\|_2^2, \tag{2.73}$$

where L_2 norm is used to define the loss function. In the objective function, terms $\mathbf{W}$, $\hat{\mathbf{W}}$ and $\mathbf{b}$, $\hat{\mathbf{b}}$ represent the variables involved in the encoder and decoder steps in *deep autoencoder*, respectively.

2.6.2.2 Deep Recurrent Neural Network

Recurrent neural network (*RNN*) is a class of artificial neural network where connections between units form a directed cycle. *RNN* has been successfully used for some special learning tasks, like language modeling, word embedding, handwritten recognition, and speech recognition. For these tasks, the inputs can usually be represented as an ordered sequence, and there exists a temporal correlation between the inputs. *RNN* can capture such a temporal correlation effectively.

In recent years, due to the significant boost of GPUs' computing power, the representation ability of *RNN* models has been greatly improved by involving more hidden layers into a deeper architecture, which is called the *deep recurrent neural network* [33]. In Fig. 2.17b, we show an example of a 3-hidden layer *deep RNN* model, which receives input from the input layer, and outputs the result to the output layer. In the *deep RNN* model, the states of the neurons depends on both the lower-layer neurons and the previous neuron (in the same layer). For the *deep RNN* model shown in Fig. 2.17b, given a training instance $((\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_T, \mathbf{y}_T))$, we can represent the hidden states of neurons corresponding to input $\mathbf{x}_t$ (where $t \in \{1, 2, \ldots, T\}$) in the three hidden layers and output layer as vectors $\mathbf{h}_t^1$, $\mathbf{h}_t^2$, $\mathbf{h}_t^3$ and $\mathbf{y}_t$, respectively. The dynamic correlations among these variables can be represented with the following equations formally:

$$\mathbf{h}_t^1 = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}^1; \boldsymbol{\theta}_h), \tag{2.74}$$

$$\mathbf{h}_t^2 = f_h(\mathbf{h}_t^1, \mathbf{h}_{t-1}^2; \boldsymbol{\theta}_h), \tag{2.75}$$

$$\mathbf{h}_t^3 = f_h(\mathbf{h}_t^2, \mathbf{h}_{t-1}^3; \boldsymbol{\theta}_h), \tag{2.76}$$

$$\hat{\mathbf{y}}_t = f_o(\mathbf{h}_t^3; \boldsymbol{\theta}_o), \tag{2.77}$$

where $f_h(\cdot; \boldsymbol{\theta}_h)$ and $f_o(\cdot; \boldsymbol{\theta}_o)$ denote the hidden state transition function and output function parameterized by variables $\boldsymbol{\theta}_h$ and $\boldsymbol{\theta}_o$, respectively. These functions can be defined in different ways, depending on the unit models used in depicting the neuron states, e.g., traditional M-P neuron or the

jwzhanggy@gmail.com

LSTM (i.e., long short-term memory) unit [19], GRU (i.e., gated recurrent unit) [8], and the recent GDU (gated diffusive unit) [56].

For the provided training instance $((\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_T, \mathbf{y}_T))$, the loss introduced by the prediction result compared against the ground truth can be denoted as

$$J(\boldsymbol{\theta}_h, \boldsymbol{\theta}_o) = \sum_{t=1}^{T} d(\hat{\mathbf{y}}_t, \mathbf{y}_t), \tag{2.78}$$

where $d(\cdot, \cdot)$ denotes the difference between the provided variables, e.g., Euclidean distance or cross-entropy.

Given a set of n training instances $\mathcal{T} = \{((\mathbf{x}_1^i, \mathbf{y}_1^i), (\mathbf{x}_2^i, \mathbf{y}_2^i), \ldots, (\mathbf{x}_T^i, \mathbf{y}_T^i))\}_{i=1}^n$, by minimizing the training loss, we will be able to learn the variables $\boldsymbol{\theta}_h$ and $\boldsymbol{\theta}_o$ of the *deep RNN* model. The learning process of the *deep RNN* models with the classic M-P neuron may usually suffer from the gradient vanishing/exploding problem a lot. To overcome such a problem, some new unit neuron models have been proposed, including LSTM, GRU, and GDU. More detailed descriptions about these unit models are available in [8, 19, 56].

2.6.2.3 Deep Convolutional Neural Network

The deep *convolutional neural network* (*CNN*) model [27] is a type of feed-forward artificial neural network, in which the connectivity pattern between the neurons is inspired by the organization of the animal visual cortex. *CNN* has been shown to be effective in a lot of applications, especially in image and computer vision related tasks. The concrete applications of *CNN* include image classification, image semantic segmentation, and object detection in images. In recent years, some research works also propose to apply *CNN* for the textual representation learning and classification tasks [24]. In this part, we will use image classification problem as an example to illustrate the *CNN* model. As shown in Fig. 2.17c, given an image input, the image classification task aims at determining the labels for the image.

According to the architecture, the *CNN* model is formed by a stack of distinct layers that transform the input image into the output labels. Depending on their functions, these layers can be categorized into the *input layer*, *convolutional layer*, *pooling layer*, *ReLU layer*, *fully connected layer*, and *output layer*, which will be introduced as follows, respectively:

- **Input Layer**: In the input layer, the neurons receive the image data input, and represent it as a stack of matrices (e.g., a high-order tensor). For instance, as shown in Fig. 2.17c, for an input image of size 32×32 by pixels, we can represent it as a 3-way tensor of dimensions $32 \times 32 \times 3$ if the image is represented in the RGB format.
- **Convolutional Layer**: In the convolutional layer, convolution kernels will applied to extract patterns from the current image representations. For instance, if matrix $\mathbf{K} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ is used as the convolution kernel, by applying it to extract features from the images, we will be able to identify the horizontal edges from the image, where the pixels above and below the edge differ a lot. In Fig. 2.17c, a 5×5 kernel is applied to the input images, which brings about a representation of dimensions $28 \times 28 \times 3$.
- **Pooling Layer**: Another important layer in the *CNN* model is the pooling layer, which performs non-linear down-sampling of the feature representation from the convolution layers. There are several different non-linear functions to implement pooling, among which *max pooling* and *mean*

jwzhanggy@gmail.com

pooling are the most common pooling techniques. Pooling partitions the feature representation into a set of non-overlapping rectangles, and for each such sub-region, it outputs the maximum number (if *max-pooling* is applied). The intuition of pooling is that once a feature has been found, its exact location isn't as important as its rough location relative to other features. Pooling greatly reduces the size of the representation, which can reduce the amount of parameters and computation in the model and hence also control overfitting. Pooling is usually periodically inserted in between successive convolution layers in the *CNN* architecture. For instance, in Fig. 2.17c, we perform the pooling operation on a 2×2 sub-region in the feature representation layer of dimensions $28 \times 28 \times 3$, which will lead to a pooling layer of dimensions $14 \times 14 \times 3$.

- **ReLU Layer**: In *CNN*, when applying the *sigmoid function* as the *activation function*, it will suffer from the *gradient vanish* problem [1, 34] a lot (just like deep *RNN*), which may make the gradient based learning method (e.g., SGD) fail to work. In real-world practice, a non-linear function

$$f(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{otherwise.} \end{cases} \tag{2.79}$$

is usually used as the *activation function* instead. Neurons using such a kind of *activation function* is called the *Rectified Linear Unit* (*ReLU*), and the layers involving the *ReLU* as the units are called the *ReLU Layer*. The introduction of *ReLU* to replace sigmoid is an important change in *CNN*, which significantly reduces the difficulty in learning *CNN* variables and also greatly improves its performance.
- **Fully Connected Layer**: Via a series of *convolution layers* and *pooling layers*, the input image will be transformed into a vector representation. The classification task will be performed by a *fully connected layer* and an *output layer*, which together will compose a *single-hidden layer feed-forward neural network* actually. For instance, in Fig. 2.17c, with convolution and sampling operations, we can obtain a feature vector of dimension 120, which together with a *fully connected layer* (of dimension 84) and the output layer (of dimension 10) will perform the classification task finally.
- **Output Layer**: The classification result will be achieved from the *output layer*, which involves 10 neurons in the example in Fig. 2.17c.

2.7 Evaluation Metrics

We have introduced the classification, regression, clustering, and deep learning tasks together with their several well-known algorithms already in the previous sections, and also talked about how to build the modes with the available data set. By this context so far, we may have several questions in mind: (1) how is quality of the built models, (2) how to compare the performance of different models. To answer these two questions, we will introduce some frequently used evaluation metrics in this part for the classification, regression, and clustering task, respectively.

2.7.1 Classification Evaluation Metrics

Here, we take the binary classification task as an example. Let $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$ ($y_i \in \{+1, -1\}$) be the true labels of n data instances, and $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n]^\top$ ($\hat{y}_i \in \{+1, -1\}$) be the labels predicted by a classification model. Based on vectors $\mathbf{y}$ and $\hat{\mathbf{y}}$, we can introduce a new concept named

jwzhanggy@gmail.com

Table 2.2 Confusion matrix

	Predicted positive	Predicted negative
Actual positive	TP	FN
Actual negative	FP	TN

confusion matrix, as shown in Table 2.2. In the table, depending on the true and predicted labels, the data instances can be divided into 4 main categories as follows:

- **TP (True Positive)**: the number of correctly classified positive instances;
- **FN (False Negative)**: the number of incorrectly classified positive instances;
- **FP (False Positive)**: the number of incorrectly classified negative instances;
- **TN (True Negative)**: the number of correctly classified negative instances.

Based on the confusion matrix, we can define four different classification evaluation metrics as follows:

Accuracy:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}, \tag{2.80}$$

Precision:

$$Precision = \frac{TP}{TP + FP}, \tag{2.81}$$

Recall:

$$Recall = \frac{TP}{TP + FN}, \tag{2.82}$$

F_β**-Score**:

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}, \tag{2.83}$$

where F_1-Score (with $\beta = 1$) is normally used in practice.

Among these 4 metrics, *Accuracy* considers both TP and TN simultaneously in the computation, which works well for the *class balanced* scenarios (i.e., the amount of positive and negative data instances is close) but may suffer from a serious problem in evaluating the learning performance of classification models in the *class imbalanced* scenario.

Example 2.6 Let's assume we have one million patient data records. Among these patient records, only 100 records indicate that the patient have the Alzheimer disease (AD), which is also the objective label that we want to predict. If we treat the patient records with AD as the positive instance, and those without AD as the negative instance, then the dataset will form a *class imbalanced scenario*.

Given two models with the following performance, where *Model 1* is well-trained but *Model 2* simply predicts all the data instance to be *negative*.

- **Mode 1**: with TP = 90, FP = 10, FN = 900, TN = 999,000;
- **Model 2**: with TP = 0, FP = 100, FN = 0, TN = 999,900.

jwzhanggy@gmail.com

By comparing the performance, *Model 1* is definitely much more useful than *Model 2* in practice. However, due to the *class imbalance* problem, we may have different evaluation for these two models with *accuracy*. According to the results, we observe that *Model 1* can identify most (90%) of the AD patients from the data but also mis-identify 900 normal people as AD patients. *Model 1* can achieve an *accuracy* about $\frac{90+999,000}{1,000,000} = 99.909\%$. Meanwhile, for *Model 2*, by predicting all the patient records to be negative, it cannot identify the AD patients at all. However, it can still achieve an *accuracy* at $\frac{999,900}{1,000,000} = 99.99\%$, which is even larger than the *accuracy* of *Model 1*.

According to the above example, the *accuracy* metric will fail to work when handling the *class imbalanced* data set. However, *Precision*, *Recall*, and F_1 metrics can still work well in such a scenario. We will leave the computation of *Precision*, *Recall*, and F_1 for this example as an exercise for the readers.

Besides these four metrics, two other curve based evaluation metrics can also be defined based on the confusion matrix, which are called the *Precision-Recall curve* and *ROC curve* (receiver operating characteristic curve). Given the n data instances together with their true and predicted labels, we can obtain a $(precision, recall)$ pair at each of the data instances $\mathbf{x}_i$ based on the cumulative prediction results from the beginning to the current position. The *Precision-Recall curve* is a plot obtained based on such $(precision, recall)$ pairs. Meanwhile, as to the *ROC curve*, it is introduced based on two new concepts named *true positive rate* (TPR) and *false positive rate* (FPR):

$$TPR = \frac{TP}{TP + FN}, \tag{2.84}$$

$$FPR = \frac{FP}{TN + FP}. \tag{2.85}$$

Based on the predicted and true labels of these n data instances, a series of (TPR, FPR) pairs can be calculated, which will plot the *ROC curve*. In practice, a larger area under the curves generally indicates a better performance of the models. Formally, the area under the *ROC curve* is defined as the *ROC AUC* metric (AUC is short for *area under curve*), and the area under the *Precision-Recall curve* is called the *PR AUC*.

2.7.2 Regression Evaluation Metrics

For the regression tasks, the predicted and true labels are actually real values, instead of pre-defined class labels in classification tasks. For the regression tasks, given the predicted and true label vectors $\hat{\mathbf{y}}$ and $\mathbf{y}$ of data instances, some frequently used evaluation metrics include

Explained Variance Regression Score

$$EV(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{Var(\mathbf{y} - \hat{\mathbf{y}})}{Var(\mathbf{y})}, \tag{2.86}$$

where $Var(\cdot)$ denotes the variance of the vector elements.

jwzhanggy@gmail.com

Mean Absolute Error

$$MAE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{2.87}$$

Mean Square Error

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{2.88}$$

Median Absolute Error

$$MedAE(\mathbf{y}, \hat{\mathbf{y}}) = median(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \ldots, |y_n - \hat{y}_n|). \tag{2.89}$$

R^2 Score

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{2.90}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

2.7.3 Clustering Evaluation Metrics

For the clustering tasks, the output results are actually not real classes but just identifier of clusters. Given a data set $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{n}$ involving n data instances, we can denote the ground truth clusters and inferred clustering result as $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ and $\hat{\mathcal{C}} = \{\hat{\mathcal{C}}_1, \ldots, \hat{\mathcal{C}}_k\}$, respectively. Furthermore, for all the data instances in $\mathcal{D}$, we can represent their inferred cluster labels and the real cluster labels as $\hat{\mathbf{y}}$ and $\mathbf{y}$, respectively. For any pair of data instances $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$, based on the predicted and true cluster labels, we can divide the instance pairs into four categories (S: Same; D: Different):

SS: Set $SS = \{(\mathbf{x}_i, \mathbf{x}_j)|y_i = y_j, \hat{y}_i = \hat{y}_j, i \neq j\}$, and we denote $|SS| = a$.
SD: Set $SD = \{(\mathbf{x}_i, \mathbf{x}_j)|y_i = y_j, \hat{y}_i \neq \hat{y}_j, i \neq j\}$, and we denote $|SD| = b$.
DS: Set $DS = \{(\mathbf{x}_i, \mathbf{x}_j)|y_i \neq y_j, \hat{y}_i = \hat{y}_j, i \neq j\}$, and we denote $|DS| = c$.
DD: Set $DD = \{(\mathbf{x}_i, \mathbf{x}_j)|y_i \neq y_j, \hat{y}_i \neq \hat{y}_j, i \neq j\}$, and we denote $|DD| = d$.

Here, set SS contains the data instance pairs that are in the same cluster in both the prediction results and the ground truth; set SD contains the data instance pairs that are in the same cluster in the ground truth but in different clusters in the prediction result; set DS contains the data instance pairs that are in different clusters in the ground truth but in the same cluster in the prediction result; and set DD contains the data instance pairs that are in different clusters in both the prediction results and the ground truth. We have these set sizes sum up to be $a + b + c + d = n(n-1)$.

Based on these notations, metrics like *Jaccard's Coefficient*, *FM Index* (Fowlkes and Mallows Index), and *Rand Index* can be defined

jwzhanggy@gmail.com

Jaccard's Coefficient

$$JC(\mathbf{y}, \hat{\mathbf{y}}) = \frac{a}{a+b+c}. \tag{2.91}$$

FM Index

$$FMI(\mathbf{y}, \hat{\mathbf{y}}) = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}. \tag{2.92}$$

Rand Index

$$RI(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2(a+d)}{a+b+c+d} = \frac{2(a+d)}{n(n-1)}. \tag{2.93}$$

These above evaluation metrics can be used in the case where the clustering ground truth is available, i.e., $\mathcal{C}$ or $\mathbf{y}$ is known. In the case that no cluster ground truth can be obtained for the data instances, we will introduce another set of evaluation metrics based on the distance measures among the data instances. Formally, given a clustering result $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$, we introduce the following 4 concepts based on the instance distance measures

- **Average Distance**:

$$dist(\mathcal{C}_i) = \frac{2}{|\mathcal{C}_i|(|\mathcal{C}_i|-1)} \sum_{j,l \in \{1,2,\ldots,|\mathcal{C}_i|\}, j<l} dist(\mathbf{x}_j, \mathbf{x}_l), \tag{2.94}$$

- **Diameter**:

$$diam(\mathcal{C}_i) = \max_{j,l \in \{1,2,\ldots,|\mathcal{C}_i|\}, j<l} dist(\mathbf{x}_j, \mathbf{x}_l), \tag{2.95}$$

- **Inter-Cluster Distance**:

$$d_{\min}(\mathcal{C}_i, \mathcal{C}_j) = \min_{\mathbf{x}_i \in \mathcal{C}_i, \mathbf{x}_j \in \mathcal{C}_j} dist(\mathbf{x}_i, \mathbf{x}_j), \tag{2.96}$$

- **Inter-Cluster Center Distance**:

$$d_{cen}(\mathcal{C}_i, \mathcal{C}_j) = dist(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j). \tag{2.97}$$

Here $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ denote the centers of clusters $\mathcal{C}_i$ and $\mathcal{C}_j$, respectively. Based on these concepts, we will introduce two distance based clustering evaluation metrics: *DB Index* (Davies-Bouldin Index), and *Dunn Index*, which are frequently used in evaluating the clustering results quality.

DB Index

$$DBI = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{dist(\mathcal{C}_i) + dist(\mathcal{C}_j)}{d_{cen}(\mathcal{C}_i, \mathcal{C}_j)} \right), \tag{2.98}$$

jwzhanggy@gmail.com

Dunn Index

$$DI = \min_{i\in\{1,2,\ldots,k\}} \left\{ \min_{j\neq i} \left(\frac{d_{\min}(\mathcal{C}_i, \mathcal{C}_j)}{\max_{l\in\{1,2,\ldots,k\} diam(\mathcal{C}_l)}} \right) \right\}. \tag{2.99}$$

Besides these metrics introduced in this section, there also exist a large number of evaluation metrics proposed for different application scenarios, which will not be introduced here since they are out of the scope of this book. We may introduce some of them in the following chapters when talking about the specific social network fusion or knowledge discovery problems.

2.8 Summary

In this chapter, we provided an overview of data operations and machine learning tasks, which aims at endowing computers with the ability to "learn" and "adapt." In machine learning, experiences are usually represented as data, and the main objective of machine learning is to derive "models" from data that can capture the complicated hidden patterns. These new models can be fed to the new data, which can provide us with the inference results matching the captured patterns.

We have also introduced the basic knowledge about data, including the data attributes and data types. The data attributes can be categorized into various types, including *numerical* and *categorical*. *Record data*, *graph data*, and *ordered data* together will form the three main data types, that will be studied in this book. We have also talked about the data characteristics, including *quality*, *dimensionality*, and *sparsity*. Several basic data processing operations, e.g., *data cleaning and preprocessing*, *data aggregating and sampling*, *data dimensionality reduction and feature selection* as well as *data transformation*, have been introduced in this chapter as well.

We divided the supervised learning tasks into *classification tasks* and *regression tasks*, respectively. For the classification part, we have introduced its problem setting, training/testing set splitting approach, and two well-known models, i.e., *decision tree* and *SVM*. Meanwhile, for the regression part, we provide the description for the problem setting and three regression models in the linear form, i.e., *linear regression*, *Lasso*, and *Ridge*. We mainly focused on *clustering* when introducing the unsupervised learning tasks in this chapter. Three clustering algorithms were introduced in this chapter, including two hard clustering approaches: *K-Means* and *DBSCAN*, and one soft clustering approach: *Mixture-of-Gaussian*.

We also provided an introduction to the neural network research works and the recent deep learning models in this chapter. Starting from the *classic M-P neuron model* to *perceptron*, *multi-layer perceptron model*, and the *multi-layer feed-forward neural network model*, we provided an overview about the neural network development history. To train the models, we covered the well-known *error back-propagation* algorithm by taking the *single hidden-layer feed-forward neural network model* as an example. The latest development of deep learning models is also introduced in this chapter, including *deep autoencoder*, *deep RNN*, and *deep CNN* models, respectively.

This chapter was concluded with the descriptions of several evaluation metrics for measuring the performance of classification, regression, and clustering models, respectively.

jwzhanggy@gmail.com

2.9 Bibliography Notes

The data mining textbook [49] provides a comprehensive introduction about data types, data characteristics, and data processing operations. The readers may refer to the textbook, especially the chapters regarding the data introduction part, for more information regarding certain topics that you are interested in.

For the decision tree, the most famous representative models include ID3 [37], C4.5 [38], and CART [5], and the readers may check these three papers as a guidance when reading the decision tree section. In selecting the optimal attributes to construct the decision tree internal nodes, *information gain*, *information gain ratio*, and *Gini index* [39,47] are usually the most frequently used metrics. The tree branch pruning strategies mentioned in this chapter have a detailed introduction in [38], and the interested readers may check that article for more information.

SVM initially published in [9] has dominated the machine learning research area for a long time, which also serves as the foundation of *statistical learning* later on. SVM can achieve an outstanding performance on textual classification task [22]. Assisted with the kernel trick, SVM can be applied to handle the non-linearly separable data instances. Meanwhile, the selection of the kernels is still an open problem by this context so far.

The K-Means algorithm introduced in this chapter actually has so many different variants, like K-Medoids [23] which uses data instances as the prototypes and K-Modes [21] that can handle discrete attributes. To detect soft clustering results, K-Means can be extended to the Fuzzy C-Means [4], where each data instance can belong to multiple clusters. Some methods have been proposed for selecting the optimal cluster number K [35]. However, in the real practice, parameter K is normally selected by trying multiple different values and selecting the best one from them.

Neural network is a *black-box* model, whose learning performance is extremely challenging to explain. If the readers are interested in neural networks, you are suggested to read the textbook [18], which provides a systematic introduction about neural network models. Meanwhile, in recent years, due to the surge of deep learning, the latest deep learning book [14] has also become very popular among researchers. BP was initially proposed by Werbos in [54] and later re-introduced by Rumelhart in [43–45]. By this context so far, the BP algorithm is still used as the main learning algorithm for training neural network models.

2.10 Exercises

1. (Easy) According to the attribute type categorization provided in Fig. 2.2, please indicate the types of attributes used in Table 2.1.
2. (Easy) What's the "*curse of dimensionality*"? Please briefly explain the concept and indicate the potential problems caused by large data dimensionality, as well as the existing methods introduced to resolve such a problem.
3. (Easy) Please compute the *Precision*, *Recall*, and F_1 for both *Model 1* and *Model 2* in Example 2.6, whose performance statistics are provided as follows:
   - **Mode 1**: with TP = 90, FP = 10, FN = 900, TN = 999,000;
   - **Model 2**: with TP = 0, FP = 100, FN = 0, TN = 999,900.
4. (Easy) Compare the *pre-pruning* and *post-pruning* strategies used in *decision tree* mode training, and indicate their advantages and disadvantages.
5. (Medium) Please briefly summarize the *kernel trick* used in *SVM*, and explain why *kernel trick* is helpful for training *SVM*.

jwzhanggy@gmail.com

6. (Medium) Please compare the L_1-norm and L_2-norm used in *Lasso* and *Ridge*, respectively. Since they both can regularize the model variables, please try to provide their advantages and disadvantages.
7. (Medium) Please explain why the *single hidden-layer feed-forward neural network* model provided in Fig. 2.15c can address the *XOR* problem.
8. (Hard) Based on the technique introduced in Sect. 2.3.2.2, please try to construct a *decision tree* for the data records provided in Table 2.1.
9. (Hard) Please prove that there exists the closed form solution to the *Ridge* regression model, i.e., matrix $(\mathbf{X}^\top\mathbf{X} + \alpha\mathbf{I})$ in Eq. (2.39) is invertible.
10. (Hard) Please try to prove that Eq. (2.23) introduced in Sect. 2.3.3.2 holds, i.e., "*Maxmin no greater than Minmax*."

$$\max_{\boldsymbol{\alpha}} \min_{\mathbf{w},b} L(\mathbf{w}, b, \boldsymbol{\alpha}) \leq \min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}} L(\mathbf{w}, b, \boldsymbol{\alpha}). \tag{2.100}$$

References

1. Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
2. T. Bengtsson, P. Bickel, B. Li, Curse-of-dimensionality revisited: collapse of the particle filter in very large scale systems, in *Probability and Statistics: Essays in Honor of David A. Freedman*, vol. 2 (2008), pp. 316–334
3. S. Berchtold, C. Bohm, H. Kriegel, The pyramid-technique: towards breaking the curse of dimensionality, in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD '02)*, vol. 27, pp. 142–153 (1998)
4. J. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms* (Kluwer Academic Publishers, Norwell, 1981)
5. L. Breiman, J. Friedman, R. Olshen, C. Stone, *Classification and Regression Trees* (Wadsworth and Brooks, Monterey, 1984)
6. C. Brodley, P. Utgoff, Multivariate decision trees. Mach. Learn. **19**(1), 45–77 (1995)
7. O. Chapelle, B. Schlkopf, A. Zien, *Semi-supervised Learning*, 1st edn. (MIT Press, Cambridge, 2010)
8. J. Chung, Ç. Gülçehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555 (2014)
9. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
10. J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
11. B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, Least angle regression. Ann. Stat. **32**, 407–499 (2004)
12. M. Ester, H. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (AAAI Press, Menlo Park, 1996)
13. S. Fahlman, C. Lebiere, The cascade-correlation learning architecture, in *Advances in Neural Information Processing Systems 2* (Morgan-Kaufmann, Burlington, 1990)
14. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016). http://www.deeplearningbook.org
15. I. Guyon, A. Elisseeff, An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–11182 (2003)
16. J. Hartigan, M. Wong, A k-means clustering algorithm. JSTOR Appl. Stat. **28**(1), 100–108 (1979)
17. D. Hawkins, The problem of overfitting. J. Chem. Inf. Comput. Sci. **44**(1), 1–12 (2004)
18. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd edn. (Prentice Hall PTR, Upper Saddle River, 1998)
19. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
20. A. Hoerl, R. Kennard, Ridge regression: biased estimation for nonorthogonal problems. Technometrics **42**(1), 80–86 (2000)
21. Z. Huang, Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Min. Knowl. Discov. **2**(3), 283–304 (1998)

jwzhanggy@gmail.com

22. T. Joachims, Text categorization with support vector machines: learning with many relevant features, in *European Conference on Machine Learning* (Springer, Berlin, 1998)
23. L. Kaufmann, P. Rousseeuw, *Clustering by Means of Medoids* (North Holland/Elsevier, Amsterdam, 1987)
24. Y. Kim, Convolutional neural networks for sentence classification, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Association for Computational Linguistics, Doha, 2014)
25. R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in *International Joint Conference on Artificial Intelligence (IJCA)* (Morgan Kaufmann Publishers Inc., San Francisco, 1995)
26. A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks, in *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12)* (Curran Associates Inc., Red Hook, 2012)
27. Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in *Proceedings of the IEEE* (IEEE, Piscataway, 1998)
28. J. Liu, S. Ji, J. Ye, SLEP: sparse learning with efficient projections. Technical report (2010)
29. W. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 115–133 (1943)
30. M. Minsky, S. Papert, *Perceptrons: Expanded Edition* (MIT Press, Cambridge, 1988)
31. S. Pan, Q. Yang, A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
32. N. Parikh, S. Boyd, Proximal algorithms. Found. Trends Optim. **1**(3), 123–231 (2014)
33. R. Pascanu, C. Gulcehre, K. Cho, Y. Bengio, How to construct deep recurrent neural networks. CoRR, abs/1312.6026 (2013)
34. R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML'13)* (2013)
35. D. Pelleg, A. Moore, X-means: extending k-means with efficient estimation of the number of clusters, in *Proceedings of the 17th International Conference on Machine Learning, Stanford* (2000)
36. J. Platt, Sequential minimal optimization: a fast algorithm for training support vector machines. Technical report. Adv. Kernel Methods Support Vector Learning 208 (1998)
37. J. Quinlan, Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
38. J. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco, 1993)
39. L. Raileanu, K. Stoffel. Theoretical comparison between the Gini index and information gain criteria. Ann. Math. Artif. Intell. **41**(1), 77–93 (2004)
40. C. Rasmussen, The infinite Gaussian mixture model, in *Advances in Neural Information Processing Systems 12* (MIT Press, Cambridge, 2000)
41. J. Rawlings, S. Pantula, D. Dickey, *Applied Regression Analysis*, 2nd edn. (Springer, Berlin, 1998)
42. F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386 (1958)
43. D. Rumelhart, G. Hinton, R. Williams, Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (MIT Press, Cambridge, 1986)
44. D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors, in *Neurocomputing: Foundations of Research* (MIT Press, Cambridge, 1988)
45. D. Rumelhart, R. Durbin, R. Golden, Y. Chauvin, Backpropagation: the basic theory, in *Developments in Connectionist Theory. Backpropagation: Theory, Architectures, and Applications* (Lawrence Erlbaum Associates, Inc., Hillsdale, 1995)
46. R. Salakhutdinov, G. Hinton, Deep Boltzmann machines, in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics* (2009)
47. C. Shannon, A mathematical theory of communication. SIGMOBILE Mob. Comput. Commun. Rev. **5**(1), 3–55 (2001)
48. D. Svozil, V. Kvasnicka, J. Pospichal, Introduction to multi-layer feed-forward neural networks. Chemom. Intell. Lab. Syst. **39**(1), 43–62 (1997)
49. P. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining (First Edition)* (Addison-Wesley Longman Publishing Co., Inc., Boston, 2005)
50. R. Tibshirani, The lasso method for variable selection in the cox model. Stat. Med. **16**, 385–395 (1997)
51. L. Van Der Maaten, E. Postma, J. Van den Herik, Dimensionality reduction: a comparative review. J. Mach. Learn. Res. **10**, 66–71 (2009)
52. M. Verleysen, D. François, The curse of dimensionality in data mining and time series prediction, in *Computational Intelligence and Bioinspired Systems. International Work-Conference on Artificial Neural Networks* (Springer, Berlin, 2005)
53. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**, 3371–3408 (2010)

jwzhanggy@gmail.com

54. P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, 1974
55. X. Yan, X. Su, *Linear Regression Analysis: Theory and Computing* (World Scientific Publishing Co., Inc., River Edge, 2009)
56. J. Zhang, L. Cui, Y. Fu, F. Gouza, Fake news detection with deep diffusive network model. CoRR, abs/1805.08751 (2018)
57. X. Zhu, Semi-supervised learning literature survey. Comput. Sci. **2**(3), 4 (2006)

jwzhanggy@gmail.com