CHAPTER

2 Data, measurements, and data preprocessing

To conduct successful data mining, the first important thing is to get familiar with your data. You may want to know the following: What are the types of *attributes* or fields that make up your data? What kind of values does each attribute have? How are the values distributed? How can we measure the similarity of some data objects with respect to others? Gaining such insights into the data will help with the subsequent analysis. Moreover, real-world data are typically noisy, enormous in volume (often several gigabytes or more), and may originate from a hodgepodge of heterogeneous sources. How can we measure the quality of data? How can we clean and integrate data from multiple heterogeneous sources? How can we normalize, compress, or transform the data? How can we reduce the dimensionality of data to help subsequent analysis? These are the tasks of this chapter.

We begin in Section 2.1 by studying the various attribute types. These include nominal attributes, binary attributes, ordinal attributes, and numeric attributes. Basic *statistical descriptions* can be used to learn more about each attribute's values, as described in Section 2.2. Given a *temperature* attribute, for example, we can determine its *mean* (average value), *median* (middle value), and *mode* (most common value). These are *measures of central tendency*, which give us an idea of the "middle" or center of a distribution. Knowing such basic statistics regarding each attribute makes it easier to fill in missing values, smooth noisy values, and spot outliers during data preprocessing. Knowledge of the attributes and attribute values can also help in fixing inconsistencies incurred during data integration. Plotting the measures of central tendency shows us if the data are symmetric or skewed. Quantile plots, histograms, and scatter plots are other graphic displays of basic statistical descriptions. These can all be useful during data preprocessing and can provide insight into areas for mining.

We may also want to examine how *similar* (or *dissimilar*) data objects are. For example, suppose we have a database where the data objects are patients, described by their symptoms. We may want to find the similarity or dissimilarity between individual patients. Such information can allow us to find clusters of like patients within the data set. The similarity (or dissimilarity) between objects may also be used to detect outliers in the data, or to perform nearest-neighbor classification. There are many measures for assessing similarity and dissimilarity. In general, such measures are referred to as *proximity measures*. Think of the proximity of two objects as a function of the *distance* between their attribute values, although proximity can also be calculated based on probabilities rather than actual distance. Measures of data proximity are described in Section 2.3.

Finally, we will discuss data preprocessing, which is to address today's real-world challenges: data sets are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size and their likely origin from multiple, heterogeneous sources. Low-quality data will lead to low-quality mining results. Huge efforts need to be paid to preprocess the data to enhance the quality of data for effective mining. Section 2.4 is on *data cleaning* and *data integration*. The former is to remove noise and correct inconsistencies in data, whereas the latter is to merge data from multiple sources into a

Data Mining. https://doi.org/10.1016/B978-0-12-811760-6.00012-6

coherent data store such as a data warehouse. Section 2.5 is on *data transformation*, which transforms or consolidates data into forms appropriate for mining. That is, it can make the resulting mining process be more efficient, and the patterns found be easier to understand. Various strategies for data transformation have been developed. For example, *data normalization* scales the attribute data to fall within a smaller range, like 0.0 to 1.0; *data discretization* replaces the raw values of a numeric attribute by interval labels or conceptual labels; and *data reduction* techniques (e.g., *compression* and *sampling*) transform the input data to a reduced representation and can improve the accuracy and efficiency of mining algorithms involving distance measurements. Last, Section 2.6 is on *dimensionality reduction*, which is the process of reducing the number of random variables or attributes under consideration. Please note that various data preprocessing techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a *date* field to a common format.

2.1 Data types

Data sets are made up of data objects. A **data object** represents an entity—in a sales database, the objects may be customers, store items, and sales; in a medical database, the objects may be patients; in a university database, the objects may be students, professors, and courses. Data objects are typically described by attributes. Data objects can also be referred to as *samples, examples, instances, data points*, or *objects*. If the data objects are stored in a database, they are *data tuples*. That is, the rows of a database correspond to the data objects, and the columns correspond to the attributes. In this section, we define attributes and look at the various attribute types.

What is an attribute? An **attribute** is a data field, representing a characteristic or feature of a data object. The nouns *attribute*, *dimension*, *feature*, and *variable* are often used interchangeably in the literature. The term *dimension* is commonly used in data warehousing. Machine learning literature tends to use the term *feature*, whereas statisticians prefer the term *variable*. Data mining and database professionals commonly use the term *attribute*, and we do here as well. Attributes describing a customer object can include, for example, *customer_ID*, *name*, and *address*. Observed values for a given attribute are known as *observations*. A set of attributes used to describe a given object is called an *attribute vector* (or *feature vector*). The distribution of data involving one attribute (or variable) is called *univariate*. A *bivariate* distribution involves two attributes, and so on.

The **type** of an attribute is determined by the set of possible values—nominal, binary, ordinal, or numeric—the attribute can have. In the following subsections, we introduce each type.

2.1.1 Nominal attributes

Nominal means "relating to names." The values of a **nominal attribute** are symbols or *names of things*. Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as **categorical**. The values do not have any meaningful order. In computer science, the values are also known as *enumerations*.

Example 2.1. Nominal attributes. Suppose that *hair_color* and *marital_status* are two attributes describing *person* objects. In our application, possible values for *hair_color* are *black*, *brown*, *blond*, *red*, *auburn*, *gray*, and *white*. The attribute *marital_status* can take on the values *single, married, divorced*, and *widowed*. Both *hair_color* and *marital_status* are nominal attributes. Another example of a nominal attribute is *occupation*, with the values *teacher, dentist, programmer, farmer*, and so on. □

Although we said that the values of a nominal attribute are symbols or "names of things," it is possible to represent such symbols or "names" with numbers. With *hair_color*, for instance, we can assign a code of 0 for *black*, 1 for *brown*, and so on. Another example is *customer_ID*, with possible values that are all numeric. However, in such cases, the numbers are not intended to be used quantitatively. That is, mathematical operations on values of nominal attributes are not meaningful. It makes no sense to subtract one customer ID number from another, unlike, say, subtracting an age value from another (where *age* is a numeric attribute). Even though a nominal attribute may have integers as values, it is not considered a numeric attribute because the integers are not meant to be used quantitatively. We will say more on numeric attributes in Section 2.1.4.

Because nominal attribute values do not have any meaningful order about them and are not quantitative, it makes no sense to find the mean (average) value or median (middle) value for such an attribute, given a set of objects. One thing that is of interest, however, is the attribute's most commonly occurring value. This value, known as the *mode*, is one of the measures of central tendency. You will learn about measures of central tendency in Section 2.2.

2.1.2 Binary attributes

A **binary attribute** is a nominal attribute with only two categories or states: 0 or 1, where 0 typically means that the attribute is absent, and 1 means that it is present. Binary attributes are referred to as **Boolean** if the two states correspond to *true* and *false*.

Example 2.2. Binary attributes. Given the attribute *smoker* describing a *patient* object, 1 indicates that the patient smokes, whereas 0 indicates that the patient does not. Similarly, suppose the patient undergoes a medical test that has two possible outcomes. The attribute *medical_test* is binary, where a value of 1 means the result of the test for the patient is positive, whereas 0 means the result is negative. □

A binary attribute is **symmetric** if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be the attribute *gender* having the states *male* and *female*.

A binary attribute is **asymmetric** if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a medical test for HIV. By convention, we code the most important outcome, which is usually the rarer one, by 1 (e.g., *HIV positive*) and the other by 0 (e.g., *HIV negative*).

Computing similarities between objects involving symmetric and asymmetric binary attributes will be discussed in a later section of this chapter.

2.1.3 Ordinal attributes

An **ordinal attribute** is an attribute with possible values that have a meaningful order or *ranking* among them, but the magnitude between successive values is not known.

Example 2.3. Ordinal attributes. Suppose that *drink_size* corresponds to the size of drinks available at a fast-food restaurant. This nominal attribute has three possible values: *small, medium*, and *large*.

The values have a meaningful sequence (which corresponds to increasing drink size); however, we cannot tell from the values *how much* bigger, say, a large is than a medium. Other examples of ordinal attributes include *grade* (e.g., $A+$, A, $A-$, $B+$, and so on) and *professional_rank*. Professional ranks can be enumerated in a sequential order: for example, *assistant*, *associate*, and *full* for professors, and *private, private second class, private first class, specialist, corporal, sergeant, …* for army ranks.

Ordinal attributes are useful for registering subjective assessments of qualities that cannot be measured objectively; thus ordinal attributes are often used in surveys for ratings. In one survey, participants were asked to rate how satisfied they were as customers. Customer satisfaction had the following ordinal categories: *1: very dissatisfied, 2: dissatisfied, 3: neutral, 4: satisfied*, and *5: very satisfied.* □

Ordinal attributes may also be obtained from the discretization of numeric quantities by splitting the value range into a finite number of ordered categories as described in a later section on data reduction.

The central tendency of an ordinal attribute can be represented by its mode and its median (the middle value in an ordered sequence), but the mean cannot be defined.

Note that nominal, binary, and ordinal attributes are *qualitative*. That is, they *describe* a feature of an object without giving an actual size or quantity. The values of such qualitative attributes are typically words representing categories. If integers are used, they represent computer codes for the categories, as opposed to measurable quantities (e.g., 0 for *small* drink size, 1 for *medium*, and 2 for *large*). In the following subsection we look at numeric attributes, which provide *quantitative* measurements of an object.

2.1.4 Numeric attributes

A **numeric attribute** is *quantitative*; that is, it is a measurable quantity, represented in integer or real values. Numeric attributes can be *interval-scaled* or *ratio-scaled*.

Interval-scaled attributes

Interval-scaled attributes are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, 0, or negative. Thus, in addition to providing a ranking of values, such attributes allow us to compare and quantify the *difference* between values.

Example 2.4. Interval-scaled attributes. A *temperature* attribute is interval-scaled. Suppose that we have the outdoor *temperature* values for a number of different days, where each day is an object. By ordering the values, we obtain a ranking of the objects with respect to *temperature*. In addition, we can quantify the difference between values. For example, a temperature of 20 °C is five degrees higher than a temperature of 15 °C. Calendar dates are another example. For instance, the years 2012 and 2020 are eight years apart. □

Temperatures in Celsius and Fahrenheit do not have a true zero-point, that is, neither 0 °C nor 0 °F indicates "no temperature." (On the Celsius scale, for example, the unit of measurement is 1/100 of the difference between the melting temperature and the boiling temperature of water in atmospheric pressure.) Although we can compute the *difference* between temperature values, we cannot talk of one temperature value as being a *multiple* of another. Without a true zero, we cannot say, for instance, that 10 °C is twice as warm as 5 °C. That is, we cannot speak of the values in terms of ratios. Similarly, there is no true zero-point for calendar dates. (The year 0 does not correspond to the beginning of time.) This brings us to ratio-scaled attributes, for which a true zero-point exists.

Because interval-scaled attributes are numeric, we can compute their mean value, in addition to the median and mode measures of central tendency.

Ratio-scaled attributes

A **ratio-scaled attribute** is a numeric attribute with an inherent zero-point. That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value. In addition, the values are ordered, and we can also compute the difference between values, as well as the mean, median, and mode.

Example 2.5. Ratio-scaled attributes. Unlike temperatures in Celsius and Fahrenheit, the Kelvin (K) temperature scale has what is considered a true zero-point ($0\,\mathrm{K} = -273.15\,^{\circ}\mathrm{C}$): It is the point at which all thermal motion ceases in the classical description of thermodynamics. Other examples of ratio-scaled attributes include *count* attributes such as *years_of_experience* (e.g., the objects are employees) and *number_of_words* (e.g., the objects are documents). Additional examples include attributes to measure weight, height, and speed, and monetary quantities (e.g., you are 100 times richer with $100 than with $1). □

2.1.5 Discrete vs. continuous attributes

In our presentation, we have organized attributes into nominal, binary, ordinal, and numeric types. There are many ways to organize attribute types. The types are not mutually exclusive.

Classification algorithms developed from the field of machine learning often consider attributes as being either *discrete* or *continuous*. Each type may be processed differently. A **discrete attribute** has a finite or countably infinite set of values, which may or may not be represented as integers. The attributes *hair_color*, *smoker*, *medical_test*, and *drink_size* each have a finite number of values, and so are discrete. Note that discrete attributes may have numeric values, such as 0 and 1 for binary attributes or, the values 0 to 110 for the attribute *age*. An attribute is *countably infinite* if the set of possible values is infinite but the values can be put in a one-to-one correspondence with natural numbers. For example, the attribute *customer_ID* is countably infinite. The number of customers can grow to infinity, but in reality, the actual set of values is countable (where the values can be put in one-to-one correspondence with the set of integers). Zip codes are another example.

If an attribute is not discrete, it is **continuous**. The terms *numeric attribute* and *continuous attribute* are often used interchangeably in the literature. (This can be confusing because, in the classic sense, continuous values are real numbers, whereas numeric values can be either integers or real numbers.) In practice, real values are represented using a finite number of digits. Continuous attributes are typically represented as floating-point variables.

2.2 Statistics of data

For data preprocessing to be successful, it is essential to have an overall picture of your data. Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.

This section discusses three areas of basic statistical descriptions. We start with *measures of central tendency* (Section 2.2.1), which measure the location of the middle or center of a data distribution.

Intuitively speaking, given an attribute, where do most of its values fall? In particular, we discuss the mean, median, mode, and midrange.

In addition to assessing the central tendency of our data set, we also would like to have an idea of the *dispersion of the data*. That is, how are the data spread out? The most common data dispersion measures are the *range*, *quartiles* (e.g., Q_1, which is the first quartile, i.e., the 25th percentile), and *interquartile range*; the *five-number summary* and *boxplots*; and the *variance* and *standard deviation* of the data. These measures are useful for identifying outliers and are described in Section 2.2.2.

To facilitate the description of relations among multiple variables, the concepts of *co-variance* and *correlation coefficient* for numerical data and χ^2 *correlation test* for nominal data are introduced in Section 2.2.3.

Finally, we can use many graphic displays of basic statistical descriptions to visually inspect our data (Section 2.2.4). Most statistical or graphical data presentation software packages include bar charts, pie charts, and line graphs. Other popular displays of data summaries and distributions include *quantile plots*, *quantile-quantile plots*, *histograms*, and *scatter plots*.

2.2.1 Measuring the central tendency

In this section, we look at various ways to measure the central tendency of data. Suppose that we have some attribute X, like *salary*, which has been recorded for a set of objects. Let $x_1, x_2, \ldots, x_N$ be the set of N observed values or *observations* for X. Here, these values may also be referred to as the data set (for X). If we were to plot the observations for *salary*, where would most of the values fall? This gives us an idea of the central tendency of the data. Measures of central tendency include the mean, median, mode, and midrange.

The most common and effective numeric measure of the "center" of a set of data is the *(arithmetic) mean*. Let $x_1, x_2, \ldots, x_N$ be a set of N values or *observations*, such as for some numeric attribute X, like *salary*. The **mean** of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}. \tag{2.1}$$

This corresponds to the built-in aggregate function, *average* (`avg()` in SQL), provided in relational database systems.

Example 2.6. Mean. Suppose we have the following values for *salary* (in thousands of dollars), shown in ascending order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110. Using Eq. (2.1), we have

$$\begin{aligned}\bar{x} &= \frac{30+36+47+50+52+52+56+60+63+70+70+110}{12}\\ &= \frac{696}{12} = 58.\end{aligned}$$

Thus, the mean salary is $58,000. □

Sometimes, each value x_i in a set may be associated with a weight w_i for $i = 1, \ldots, N$. The weights reflect the significance, importance, or occurrence frequency attached to their respective values. In this case, we can compute

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_N x_N}{w_1 + w_2 + \cdots + w_N}. \tag{2.2}$$

This is called the **weighted arithmetic mean** or the **weighted average**.

Although the mean is the single most useful quantity for describing a data set, it is not always the best way of measuring the center of the data. A major problem with the mean is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers. Similarly, the mean score of a class in an exam could be pulled down quite a bit by a few very low scores. To offset the effect caused by a small number of extreme values, we can instead use the **trimmed mean**, which is the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for *salary* and remove the top and bottom 2% before computing the mean. We should avoid trimming too large a portion (such as 20%) at both ends, as this can result in the loss of valuable information.

For skewed (asymmetric) data, a better measure of the center of data is the **median**, which is the middle value in a set of ordered data values. It is the value that separates the higher half of a data set from the lower half.

In probability and statistics, the median generally applies to numeric data; however, we may extend the concept to ordinal data. Suppose that a given data set of N values for an attribute X is sorted in ascending order. If N is odd, then the median is the *middle value* of the ordered set. If N is even, then the median is not unique; it is the two middlemost values and any value in between. If X is a numeric attribute in this case, by convention, the median is taken as the average of the two middlemost values.

Example 2.7. Median. Let's find the median of the data from Example 2.6. The data are already sorted in ascending order. There is an even number of observations (i.e., 12); therefore, the median is not unique. It can be any value within the two middlemost values of 52 and 56 (that is, within the sixth and seventh values in the list). By convention, we assign the average of the two middlemost values as the median; that is, $\frac{52+56}{2} = \frac{108}{2} = 54$. Thus, the median is \$54,000.

Suppose that we had only the first 11 values in the list. Given an odd number of values, the median is the middlemost value. This is the sixth value in this list, which has a value of \$52,000. □

The median is expensive to compute when we have a large number of observations. For numeric attributes, however, we can easily *approximate* the value. Assume that data are grouped in intervals according to their x_i data values and that the frequency (i.e., number of data values) of each interval is known. For example, employees may be grouped according to their annual salary in intervals such as \$10,001–20,000, \$20,001–50,000, and so on. (A similar, concrete example can be seen in the data table of Exercise 2.3.) Let the interval that contains the median frequency be the *median interval*. We can approximate the median of the entire data set (e.g., the median salary) by interpolation using the formula

$$median \approx L_1 + \left(\frac{N/2 - \left(\sum freq\right)_l}{freq_{median}}\right) \times width, \tag{2.3}$$

where L_1 is the lower boundary of the median interval, N is the number of values in the entire data set, $\left(\sum freq\right)_l$ is the sum of the frequencies of all of the intervals that are lower than the median interval, $freq_{median}$ is the frequency of the median interval, and *width* is the width of the median interval.

Mode is another measure of central tendency. The **mode** for a set of data is the value that occurs most frequently compared to all neighboring values in the set. Therefore, it can be determined for qualitative and quantitative attributes. It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. Data sets with one, two, or three modes are respectively called **unimodal**, **bimodal**, and **trimodal**. In general, a data set with two or more modes is **multimodal**.

Example 2.8. Mode. The data from Example 2.6 are bimodal. The two modes are \$52,000 and \$70,000. □

For unimodal numeric data that are moderately skewed (asymmetrical), we have the following empirical relation:

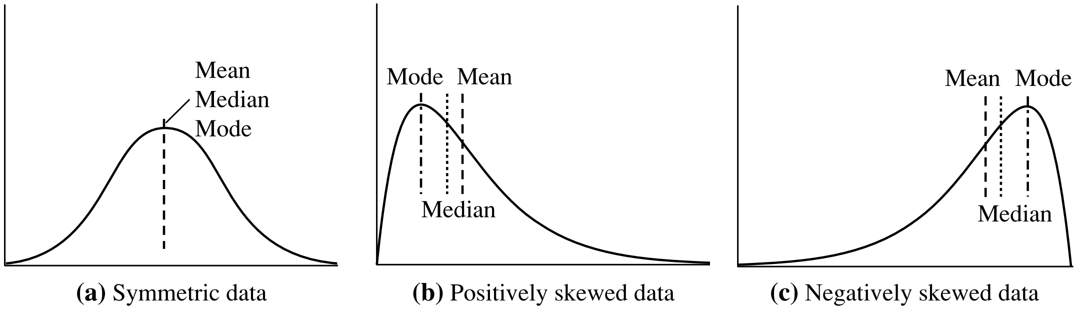
$$mean - mode \approx 3 \times (mean - median). \tag{2.4}$$

This implies that the mode for unimodal frequency curves that are moderately skewed can easily be approximated if the mean and median values are known.

The **midrange** can also be used to assess the central tendency of a numeric data set. It is the average of the largest and smallest values in the set. This measure is easy to compute using the SQL aggregate functions, `max()` and `min()`.

Example 2.9. Midrange. The midrange of the data of Example 2.6 is $\frac{30{,}000+110{,}000}{2} = \$70{,}000$. □

In a unimodal frequency curve with perfect **symmetric** data distribution, the mean, median, and mode are all at the same center value, as shown in Fig. 2.1a.



(a) Symmetric data **(b)** Positively skewed data **(c)** Negatively skewed data

FIGURE 2.1

Mean, median, and mode of symmetric vs. positively and negatively skewed data.

Data in most real applications are not symmetric. They may instead be either **positively skewed**, where the mode occurs at a value that is smaller than the median (Fig. 2.1b), or **negatively skewed**, where the mode occurs at a value greater than the median (Fig. 2.1c).

2.2.2 Measuring the dispersion of data

We now look at measures to assess the dispersion or spread of numeric data. The measures include range, quantiles, quartiles, percentiles, and the interquartile range. The five-number summary, which can be displayed as a boxplot, is useful in identifying outliers. Variance and standard deviation also indicate the spread of a data distribution.

Range, quartiles, and interquartile range

To start off, let's study the *range*, *quantiles*, *quartiles*, *percentiles*, and the *interquartile range* as measures of data dispersion.

Let $x_1, x_2, \ldots, x_N$ be a set of observations for some numeric attribute, X. The **range** of the set is the difference between the largest (max()) and smallest (min()) values.

Suppose that the data for attribute X are sorted in ascending numeric order. Imagine that we can pick certain data points so as to split the data distribution into equal-size consecutive sets, as in Fig. 2.2. These data points are called *quantiles*. **Quantiles** are points taken at regular intervals of a data distribution, dividing it into essentially equal-size consecutive sets. (We say "essentially" because there may not be data values of X that divide the data into exactly equal-size subsets. For readability, we will refer to them as equal.) The kth *q-quantile* for a given data distribution is the value x such that at most k/q of the data values are less than x and at most $(q-k)/q$ of the data values are more than x, where k is an integer such that $0 < k < q$. There are $q-1$ q-quantiles.

The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median. The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as **quartiles**. The 100-quantiles are more commonly referred to as **percentiles**; they divide

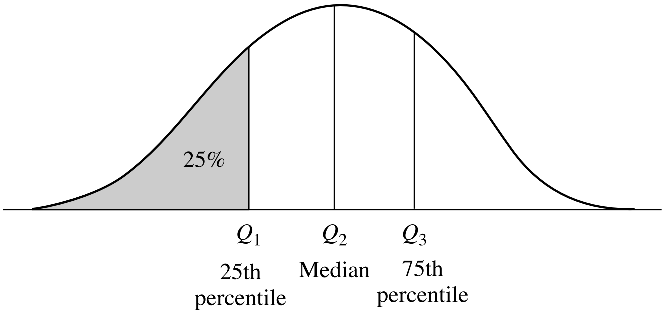


FIGURE 2.2

A plot of the data distribution for some attribute X. The quantiles plotted are quartiles. The three quartiles divide the distribution into four equal-size consecutive subsets. The second quartile corresponds to the median.

the data distribution into 100 equal-size consecutive sets. The median, quartiles, and percentiles are the most widely used forms of quantiles.

The quartiles give an indication of a distribution's center, spread, and shape. The **first quartile**, denoted by Q_1, is the 25th percentile. It cuts off the lowest 25% of the data. The **third quartile**, denoted by Q_3, is the 75th percentile—it cuts off the lowest 75% (or highest 25%) of the data. The second quartile is the 50th percentile. As the median, it gives the center of the data distribution.

The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range** (**IQR**) and is defined as

$$IQR = Q_3 - Q_1. \tag{2.5}$$

Example 2.10. Interquartile range. The quartiles are the three values that split the sorted data set into four equal parts. The data of Example 2.6 contain 12 observations, already sorted in ascending order. Since there are even number of elements on this list, the median of the list should be the mean of the center two elements, that is ($52,000 + $56,000)/2 = $54,000. Then the first quartile should be the mean of the 3rd and 4th elements, that is, ($47,000 + $50,000)/2 = $48,500, whereas the 3rd quartile should be the mean of the 9th and 10th elements, that is, ($63,000 + $70,000)/2 = $66,500. Thus the interquartile range is IQR = $66,500 − $48,500 = $18,000. □

Five-number summary, boxplots, and outliers

No single numeric measure of spread (e.g., IQR) is very useful for describing skewed distributions. Have a look at the symmetric and skewed data distributions of Fig. 2.1. In the symmetric distribution, the median (and other measures of central tendency) splits the data into equal-size halves. This does not occur for skewed distributions. Therefore it is more informative to also provide the two quartiles Q_1 and Q_3, along with the median. A common rule of thumb for identifying suspected **outliers** is to single out values falling at least 1.5 × IQR above the third quartile or below the first quartile.

Because Q_1, the median, and Q_3 together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the *five-number summary*. The **five-number summary** of a distribution consists of the median (Q_2), the quartiles Q_1 and Q_3, and the smallest and largest individual observations, written in the order of *Minimum*, Q_1, *Median*, Q_3, *Maximum*.

Boxplots are a popular way of visualizing a distribution. A boxplot incorporates the five-number summary as follows:

- Typically, the ends of the box are at the quartiles so that the box length is the interquartile range.
- The median is marked by a line within the box.
- Two lines (called *whiskers*) outside the box extend to the smallest (*Minimum*) and largest (*Maximum*) observations.

When dealing with a moderate number of observations, it is worthwhile to plot potential outliers individually. To do this in a boxplot, the whiskers are extended to the extreme low and high observations *only if* these values are less than 1.5 × IQR beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within 1.5 × IQR of the quartiles. The remaining cases are plotted individually. Boxplots can be used in the comparisons of several sets of compatible data.

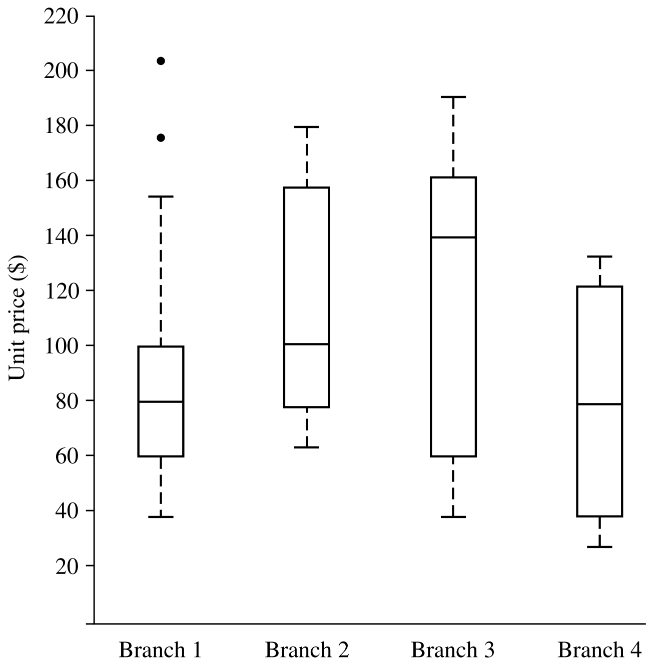


FIGURE 2.3

Boxplot for the unit price data for items sold at four branches of an online store during a given time period.

Example 2.11. Boxplot. Fig. 2.3 shows boxplots for unit price data for items sold at four branches of an online store during a given time period. For branch 1, we see that the median price of items sold is \$80, Q_1 is \$60, and Q_3 is \$100. Notice that two outlying observations for this branch were plotted individually, as their values of 175 and 202 are more than 1.5 times the *IQR* here of 40. □

Variance and standard deviation

Variance and standard deviation are measures of data dispersion. They indicate how spread out a data distribution is. A low standard deviation means that the data observations tend to be very close to the mean, whereas a high standard deviation indicates that the data are spread out over a large range of values.

The **variance** of N observations, $x_1, x_2, \ldots, x_N$ (when N is large), for a numeric attribute X is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N}x_i^2\right) - \bar{x}^2, \tag{2.6}$$

where $\bar{x}$ is the mean value of the observations, as defined in Eq. (2.1). The **standard deviation**, σ, of the observations is the square root of the variance, σ^2.

Example 2.12. Variance and standard deviation. In Example 2.6, we found $\bar{x} = \$58{,}000$ using Eq. (2.1) for the mean. To determine the variance and standard deviation of the data from that example, we set $N = 12$ and use Eq. (2.6) to obtain

$$\begin{aligned}\sigma^2 &= \frac{1}{12}(30^2 + 36^2 + 47^2 \ldots + 110^2) - 58^2 \\ &\approx 379.17 \\ \sigma &\approx \sqrt{379.17} \approx 19.47.\end{aligned}$$

□

The basic properties of the standard deviation, σ, as a measure of spread are as follows:

- σ measures spread about the mean and should be considered only when the mean is chosen as the measure of center.
- $\sigma = 0$ only when there is no spread, that is, when all observations have the same value. Otherwise, $\sigma > 0$.

Importantly, an observation is unlikely to be more than several standard deviations away from the mean. Mathematically, using Chebyshev's inequality, it can be shown that at least $\left(1 - \frac{1}{k^2}\right) \times 100\%$ of the observations are no more than k standard deviations from the mean. Therefore, the standard deviation is a good indicator of the spread of a data set.

The computation of the variance and standard deviation is scalable in large data sets.

2.2.3 Covariance and correlation analysis

Covariance of numeric data

In probability theory and statistics, correlation and covariance are two similar measures for assessing how much two attributes change together. Consider two numeric attributes A and B and a set of n real-valued observations $\{(a_1, b_1), \ldots, (a_n, b_n)\}$. The mean values of A and B, respectively, are also known as the **expected values** on A and B, that is,

$$E(A) = \bar{A} = \frac{\sum_{i=1}^{n} a_i}{n}$$

and

$$E(B) = \bar{B} = \frac{\sum_{i=1}^{n} b_i}{n}.$$

The **covariance** between A and B is defined as

$$Cov(A, B) = E((A - \bar{A})(B - \bar{B})) = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{n}. \tag{2.7}$$

Mathematically, it can also be shown that

$$Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B}. \tag{2.8}$$

This equation may simplify calculations.

For two attributes A and B that tend to change together, if a value a_i of A is larger than $\bar{A}$ (the expected value of A), then the corresponding value of b_i of attribute B is likely to be larger than $\bar{B}$ (the expected value of B). Therefore the covariance between A and B is *positive*. On the other hand, if one of the attributes tends to be above its expected value when the other attribute is below its expected value, then the covariance of A and B is *negative*.

If A and B are *independent* (i.e., they do not have correlation), then $E(A \cdot B) = E(A) \cdot E(B)$. Therefore the covariance is $Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B} = E(A) \cdot E(B) - \bar{A}\bar{B} = 0$. However, the converse is not true. Some pairs of random variables (attributes) may have a covariance of 0 but are not independent. Only under some additional assumptions (e.g., the data follow multivariate normal distributions) does a covariance of 0 imply independence.

Example 2.13. Covariance analysis of numeric attributes. Consider Table 2.1, which presents a simplified example of stock prices observed at five time points for *AllElectronics* and *HighTech*, a high-tech company. If the stocks are affected by the same industry trends, will their prices rise or fall together?

$$E(AllElectronics) = \frac{6+5+4+3+2}{5} = \frac{20}{5} = \$4$$

and

$$E(HighTech) = \frac{20+10+14+5+5}{5} = \frac{54}{5} = \$10.80.$$

Thus, using Eq. (2.7), we compute

$$\begin{aligned} Cov(AllElectroncis, HighTech) &= \frac{6 \times 20 + 5 \times 10 + 4 \times 14 + 3 \times 5 + 2 \times 5}{5} - 4 \times 10.80 \\ &= 50.2 - 43.2 = 7. \end{aligned}$$

Therefore, given the positive covariance we can say that stock prices for both companies rise together. □

Variance is a special case of covariance, where the two attributes are identical (i.e., the covariance of an attribute with itself).

Table 2.1 Stock prices for *AllElectronics* and *HighTech*.

Time point	AllElectronics	HighTech
t1	6	20
t2	5	10
t3	4	14
t4	3	5
t5	2	5

Correlation coefficient for numeric data

For numeric attributes, we can evaluate the correlation between two attributes, A and B, by computing the **correlation coefficient** (also known as **Pearson's product moment coefficient**, named after its inventer, Karl Pearson). This is

$$r_{A,B} = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{n\sigma_A\sigma_B} = \frac{\sum_{i=1}^{n}(a_i b_i) - n\bar{A}\bar{B}}{n\sigma_A\sigma_B}, \tag{2.9}$$

where n is the number of tuples, a_i and b_i are the respective values of A and B in tuple i, $\bar{A}$ and $\bar{B}$ are the respective mean values of A and B, σ_A and σ_B are the respective standard deviations of A and B (as defined in Section 2.2.2), and $\Sigma(a_i b_i)$ is the sum of the AB cross-product (i.e., for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \le r_{A,B} \le +1$. If $r_{A,B}$ is greater than 0, then A and B are *positively correlated*, meaning that the values of A increase as the values of B increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that A (or B) may be removed as a redundancy.

If the resulting value is equal to 0, then A and B are *independent*, and there is no correlation between them. If the resulting value is less than 0, then A and B are *negatively correlated*, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes (Section 2.2.3). For example, Fig. 2.8's scatter plots, respectively, show positively correlated data and negatively correlated data, whereas Fig. 2.9 displays uncorrelated data.

Note that correlation does not imply causality. That is, if A and B are correlated, this does not necessarily imply that A causes B or that B causes A. For example, in analyzing a demographic database, we may find that attributes representing the number of hospitals and the number of car thefts in a region are correlated. This does not mean that one causes the other. Both are actually causally linked to a third attribute, namely, *population*.

χ^2 correlation test for nominal data

For nominal data, a correlation relationship between two attributes, A and B, can be discovered by a χ^2 (**chi-square**) test. Suppose A has c distinct values, namely, $a_1, a_2, \ldots a_c$, and B has r distinct values, namely, $b_1, b_2, \ldots b_r$. The data tuples described by A and B can be shown as a **contingency table**, with the c values of A making up the columns and the r values of B making up the rows. Let (A_i, B_j) denote the joint event that attribute A takes on value a_i and attribute B takes on value b_j, that is, where $(A = a_i, B = b_j)$. Each and every possible (A_i, B_j) joint event has its own cell (or slot) in the table. The χ^2 value (also known as the *Pearson* χ^2 *statistic*) is computed as

$$\chi^2 = \sum_{i=1}^{c}\sum_{j=1}^{r}\frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{2.10}$$

where o_{ij} is the *observed frequency* (i.e., actual count) of the joint event (A_i, B_j) and e_{ij} is the *expected frequency* of (A_i, B_j), which can be computed as

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{n}, \tag{2.11}$$

where n is the number of data tuples, $count(A = a_i)$ is the number of tuples having value a_i for A, and $count(B = b_j)$ is the number of tuples having value b_j for B. The sum in Eq. (2.10) is computed over all of the $r \times c$ cells. Note that the cells that contribute the most to the χ^2 value are those for which the actual count is very different from that expected.

The χ^2 statistic tests the hypothesis that A and B are *independent*, that is, there is no correlation between them. The test is based on a significance level, with $(r-1) \times (c-1)$ degrees of freedom. We illustrate the use of this statistic in Example 2.14. If the hypothesis can be rejected, then we say that A and B are statistically correlated.

Example 2.14. Correlation analysis of nominal attributes using χ^2. Suppose that a group of 1500 people was surveyed. The gender of each person was noted. Each person was polled as to whether his or her preferred type of reading material was fiction or nonfiction. Thus, we have two attributes, *gender* and *preferred_reading*. The observed frequency (or count) of each possible joint event is summarized in the contingency table shown in Table 2.2, where the numbers in parentheses are the expected frequencies. The expected frequencies are calculated based on the data distribution for both attributes using Eq. (2.11).

Using Eq. (2.11), we can verify the expected frequencies for each cell. For example, the expected frequency for the cell (*male, fiction*) is

$$e_{11} = \frac{count(male) \times count(fiction)}{n} = \frac{300 \times 450}{1500} = 90,$$

and so on. Notice that in any row, the sum of the expected frequencies must equal the total observed frequency for that row, and the sum of the expected frequencies in any column must also equal the total observed frequency for that column.

Using Eq. (2.10) for χ^2 computation, we get

$$\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840}$$
$$= 284.44 + 121.90 + 71.11 + 30.48 = 507.93.$$

For this 2×2 table, the degrees of freedom are $(2-1) \times (2-1) = 1$. For 1 degree of freedom, the χ^2 value needed to reject the hypothesis at the 0.001 significance level is 10.828 (taken from the table of upper percentage points of the χ^2 distribution, typically available from any textbook on statistics). Since our computed value is above this, we can reject the hypothesis that *gender* and *preferred_reading*

Table 2.2 Example 2.1's 2 × 2 contingency table data.

	Male	Female	Total
fiction	250 (90)	200 (360)	450
non_fiction	50 (210)	1000 (840)	1050
Total	300	1200	1500

Note: Are gender and preferred_reading correlated?

are independent and conclude that the two attributes are (strongly) correlated for the given group of people. □

2.2.4 Graphic displays of basic statistics of data

In this section, we study graphic displays of basic statistical descriptions. These include *quantile plots*, *quantile-quantile plots*, *histograms*, and *scatter plots*. Such graphs are helpful for the visual inspection of data, which is useful for data preprocessing. The first three of these show univariate distributions (i.e., data for one attribute), whereas scatter plots show bivariate distributions (i.e., involving two attributes).

Quantile plot

A **quantile plot** is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute (allowing a user to assess both the overall behavior and unusual occurrences). Second, it plots quantile information (see Section 2.2.2). Let x_i, for $i = 1$ to N, be the data sorted in ascending order so that x_1 is the smallest observation and x_N is the largest for some ordinal or numeric attribute X. Each observation, x_i, is paired with a percentage, f_i, which indicates that approximately $f_i \times 100\%$ of the data are below the value, x_i. We say "approximately" because there may not be a value with exactly a fraction, f_i, of the data below x_i. Note that the 0.25 quantile corresponds to quartile Q_1, the 0.50 quantile is the median, and the 0.75 quantile is Q_3.

Let

$$f_i = \frac{i - 0.5}{N}. \tag{2.12}$$

These numbers increase in equal steps of $1/N$, ranging from $\frac{1}{2N}$ (which is slightly above 0) to $1 - \frac{1}{2N}$ (which is slightly below 1). On a quantile plot, x_i is graphed against f_i. This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare their Q_1, median, Q_3, and other f_i values at a glance.

Example 2.15. Quantile plot. Fig. 2.4 shows a quantile plot for the *unit price* data of Table 2.3. □

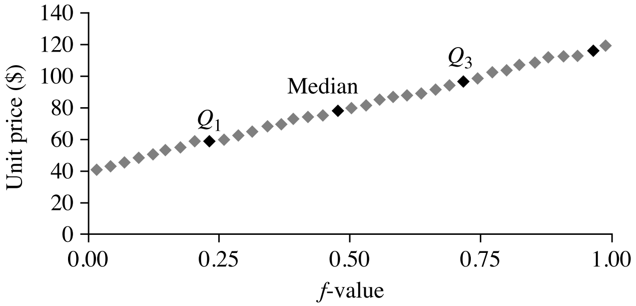


FIGURE 2.4

A quantile plot for the unit price data of Table 2.3.

Table 2.3 A set of unit price data for items sold at a branch of the online store.

Unit price ($)	Count of items sold
40	275
43	300
47	250
⋮	⋮
74	360
75	515
78	540
⋮	⋮
115	320
117	270
120	350

Quantile-quantile plot

A **quantile-quantile plot**, or **q-q plot**, graphs the quantiles of one univariate distribution against the corresponding quantiles of another. It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another.

Suppose that we have two sets of observations for the attribute or variable *unit price*, taken from two different branch locations. Let $x_1, \ldots, x_N$ be the data from the first branch, and $y_1, \ldots, y_M$ be the data from the second, where each data set is sorted in ascending order. If $M = N$ (i.e., the number of points in each set is the same), then we simply plot y_i against x_i, where y_i and x_i are both $(i - 0.5)/N$ quantiles of their respective data sets. If $M < N$ (i.e., the second branch has fewer observations than the first), there can be only M points on the q-q plot. Here, y_i is the $(i - 0.5)/M$ quantile of the y data, which is plotted against the $(i - 0.5)/M$ quantile of the x data. This computation typically involves interpolation.

Example 2.16. Quantile-quantile plot. Fig. 2.5 shows a quantile-quantile plot for *unit price* data of items sold at two branches of the online store during a given time period. Each point corresponds to the same quantile for each data set and shows the unit price of items sold at branch 1 vs. branch 2 for that quantile. (To aid comparison, the straight line represents the case where, for each given quantile, the unit price at each branch is the same. The darker points correspond to the data for Q_1, the median, and Q_3, respectively.)

We see, for example, that at Q_1, the unit price of items sold at branch 1 was slightly less than that at branch 2. In other words, 25% of items sold at branch 1 were less than or equal to $60, whereas 25% of items sold at branch 2 were less than or equal to $64. At the 50th percentile (marked by the median, which is also Q_2), we see that 50% of items sold at branch 1 were less than $78, whereas 50% of items at branch 2 were less than $85. In general, we note that there is a shift in the distribution of branch 1 with respect to branch 2 in that the unit prices of items sold at branch 1 tend to be lower than those at branch 2. □

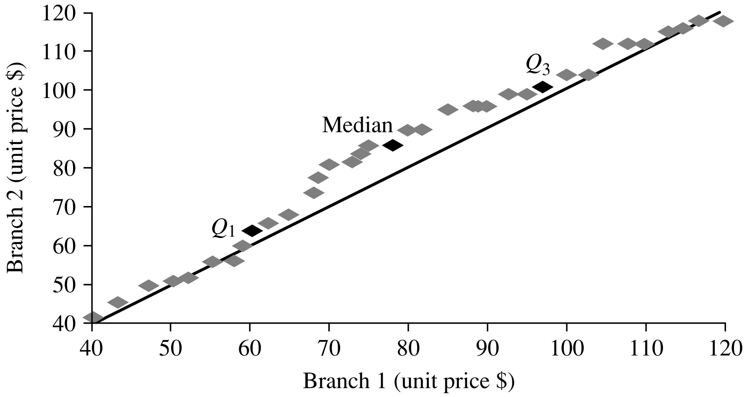


FIGURE 2.5

A q-q plot for unit price data from two branches of the online store.

Histograms

Histograms (or **frequency histograms**) are at least a century old and are widely used. “Histos” means pole or mast, and “gram” means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute, X. According to the number of poles desired in the chart, the range of values for X is partitioned into a set of disjoint consecutive subranges. The subranges, referred to as *buckets* or *bins*, are disjoint subsets of the data distribution for X. The range of a bucket is known as the **width**. Typically, the buckets are of equal width. For example, a *price* attribute with a value range of $1–$200 (rounded up to the nearest dollar) can be partitioned into subranges 1–20, 21–40, 41–60, and so on. For each subrange, a bar is drawn with a height that represents the total count of items observed within the subrange.

Please note that histogram is different from another popularly used graph representation called **bar chart**. Bar chart uses a set of bars (often separated with space) with X representing a set of categorical data, such as *automobile_model* or *item_type*, and the height of the bar (column) indicates the size of the group defined by the categories. On the other hand, histogram plots quantitative data with a range of X values grouped into bins or intervals. Histograms are used to show distributions (along X axis) while bar charts are used to compare categories. It is always appropriate to talk about the skewness of a histogram; that is, the tendency of the observations to fall more on the low end or the high end of the X axis. However, bar chart’s X axis does not have a low end or a high end; because the labels on the X axis are categorical—not quantitative. Thus, bars can be reordered in bar charts but not in histograms.

Example 2.17. Histogram. Fig. 2.6 shows a histogram for a data set on research award distribution for a region, where buckets (or bins) are defined by equal-width ranges representing $1000 increments, and the frequency is the number of research awards in the corresponding buckets. □

Although histograms are widely used, they may not be as effective as the quantile plot, q-q plot, and boxplot methods in comparing groups of univariate observations.

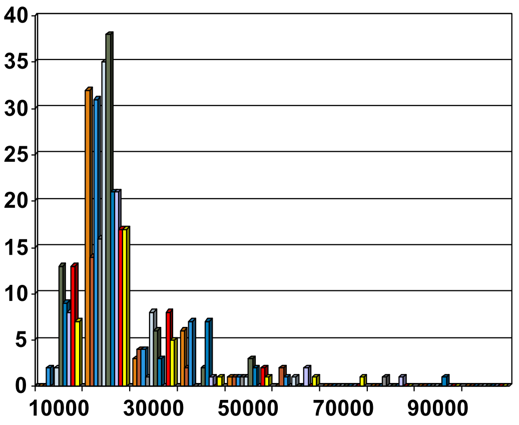


FIGURE 2.6

A histogram on research award distribution for a region.

Scatter plots and data correlation

A **scatter plot** is one of the most effective graphical methods for determining whether there appears to be a relationship, pattern, or trend between two numeric attributes. To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane. Fig. 2.7 shows a scatter plot for the set of data in Table 2.3.

The scatter plot is a useful method for providing a first look at bivariate data to see clusters of points and outliers, or to explore the possibility of correlation relationships. Two attributes, X and Y, are **correlated** if the knowledge of one attribute enables to predict the other with some accuracy. Correlations can be positive, negative, or null (uncorrelated). Fig. 2.8 shows examples of positive and negative correlations between two attributes.

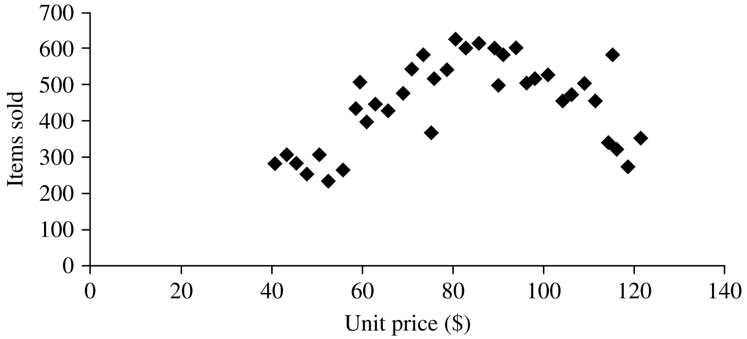


FIGURE 2.7

A scatter plot for Table 2.3 data set.

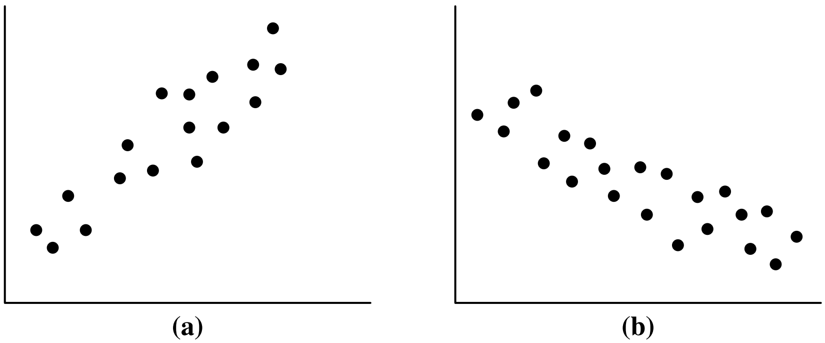


FIGURE 2.8

Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.

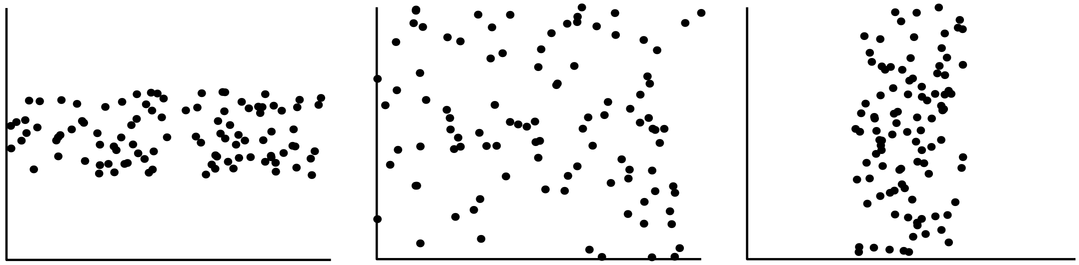


FIGURE 2.9

Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.

If the plotted points pattern slopes from lower left to upper right, this means that the values of X increase as the values of Y increase, suggesting a *positive correlation* (Fig. 2.8a). If the pattern of plotted points slopes from upper left to lower right, the values of X increase as the values of Y decrease, suggesting a *negative correlation* (Fig. 2.8b). A line of best fit can be drawn to study the correlation between the variables. Statistical tests for correlation are introduced in Appendix A.

Fig. 2.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets. Scatter plots can also be extended to n attributes, resulting in a *scatter-plot matrix*.

In summary, basic data descriptions (e.g., measures of central tendency and measures of dispersion) and graphic statistical displays (e.g., quantile plots, histograms, and scatter plots) provide valuable insight into the overall behavior of your data. By helping to identify noise and outliers, they are especially useful for data cleaning.

2.3 Similarity and distance measures

In data mining applications, such as clustering, outlier analysis, and nearest-neighbor classification, we need ways to assess how alike or unalike objects are in comparison to one another. For example, a store may want to search for clusters of *customer* objects, resulting in groups of customers with similar characteristics (e.g., similar income, area of residence, and age). Such information can then be used for marketing. A **cluster** is a collection of data objects such that the objects within a cluster are *similar* to one another and *dissimilar* to the objects in other clusters. Outlier analysis also employs clustering-based techniques to identify potential outliers as objects that are highly dissimilar to others. Knowledge of object similarities can also be used in nearest-neighbor classification schemes where a given object (e.g., a *patient*) is assigned a class label (relating to, say, a *diagnosis*) based on its similarity toward other objects in the model.

This section presents similarity and dissimilarity measures, which are referred to as measures of *proximity*. Similarity and dissimilarity are related. A similarity measure for two objects, i and j, will typically return value 0 if the objects are completely unalike. The higher the similarity value, the greater the similarity between objects. (Typically, a value of 1 indicates complete similarity, that is, the objects are identical.) A dissimilarity measure works the opposite way. It returns a value of 0 if the objects are the same (and therefore, far from being dissimilar). The higher the dissimilarity value, the more dissimilar the two objects are.

In Section 2.3.1 we present two data structures that are commonly used in the above types of applications: the *data matrix* (used to store the data objects) and the *dissimilarity matrix* (used to store dissimilarity values for pairs of objects). We also switch to a different notation for data objects than previously used in this chapter since now we are dealing with objects described by more than one attribute. We then discuss how object dissimilarity can be computed for objects described by *nominal* attributes (Section 2.3.2), by *binary* attributes (Section 2.3.3), by *numeric* attributes (Section 2.3.4), by *ordinal* attributes (Section 2.3.5), or by combinations of these attribute types (Section 2.3.6). Section 2.3.7 provides similarity measures for very long and sparse data vectors, such as term-frequency vectors representing documents in information retrieval. Finally, Section 2.3.8 discusses how to measure the difference between two probability distributions over the same variable x, and introduces a measure, called the *Kullback-Leibler divergence*, or simply, the *KL divergence*, which has been popularly used in the data mining literature.

Knowing how to compute dissimilarity is useful in studying attributes and will also be referenced in later topics on clustering (Chapters 8 and 9), outlier analysis (Chapter 11), and nearest-neighbor classification (Chapter 6).

2.3.1 Data matrix vs. dissimilarity matrix

In Section 2.2, we looked at ways of studying the central tendency, dispersion, and spread of observed values for some attribute X. Our objects there were one-dimensional, that is, described by a single attribute. In this section, we talk about objects described by *multiple* attributes. Therefore we need a change in notation. Suppose that we have n objects (e.g., persons, items, or courses) described by p attributes (also called *measurements* or *features*, such as age, height, weight, or gender). The objects are $x_1 = (x_{11}, x_{12}, \ldots, x_{1p})$, $x_2 = (x_{21}, x_{22}, \ldots, x_{2p})$, and so on, where x_{ij} is the value for object x_i of the jth attribute. For brevity, we hereafter refer to object x_i as object i. The objects may be tuples in a relational database and are also referred to as *data samples* or *feature vectors*.

Main memory-based clustering and nearest-neighbor algorithms typically operate on either of the following two data structures:

- **Data matrix** (or *object-by-attribute structure*): This structure stores the n data objects in the form of a relational table or an n-by-p matrix (n objects $\times p$ attributes):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}. \tag{2.13}$$

  Each row corresponds to an object. As part of our notation, we may use f to index through the p attributes.

- **Dissimilarity matrix** (or *object-by-object structure*): This structure stores a collection of proximities that are available for all pairs of n objects. It is often represented by an n-by-n table:

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}, \tag{2.14}$$

  where $d(i, j)$ is the measured **dissimilarity** or "difference" between objects i and j. In general, $d(i, j)$ is a nonnegative number that is close to 0 when objects i and j are highly similar or "near" each other, and becomes larger the more they differ. Note that $d(i,i) = 0$; that is, the difference between an object and itself is 0. Furthermore, $d(i, j) = d(j,i)$. (For readability, we do not show the $d(j,i)$ entries since the matrix is symmetric.) Measures of dissimilarity are discussed throughout the remainder of this chapter.

Measures of similarity can often be expressed as a function of measures of dissimilarity. For example, for nominal data,

$$sim(i, j) = 1 - d(i, j), \tag{2.15}$$

where $sim(i, j)$ is the similarity between objects i and j. Throughout the rest of this chapter, we will also comment on measures of similarity.

A data matrix is made up of two entities or "things," namely rows (for objects) and columns (for attributes). Therefore, the data matrix is often called a **two-mode** matrix. The dissimilarity matrix contains one kind of entity (dissimilarities) and so is called a **one-mode** matrix. Many clustering and nearest-neighbor algorithms operate on a dissimilarity matrix. Data in the form of a data matrix can be transformed into a dissimilarity matrix before applying such algorithms.

2.3.2 Proximity measures for nominal attributes

A nominal attribute can take on two or more states (Section 2.1.1). For example, *map_color* is a nominal attribute that may have, say, five states: *red*, *yellow*, *green*, *pink*, and *blue*.

Let the number of states of a nominal attribute be M. The states can be denoted by letters, symbols, or a set of integers, such as $1, 2, \ldots, M$. Notice that such integers are used just for data handling and do not represent any specific ordering.

"How is dissimilarity computed between objects described by nominal attributes?" The dissimilarity between two objects i and j can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p}, \tag{2.16}$$

where m is the number of *matches* (i.e., the number of attributes for which i and j are in the same state), and p is the total number of attributes describing the objects. Weights can be assigned to increase the effect of m or to assign greater weight to the matches in attributes having a larger number of states.

Example 2.18. Dissimilarity between nominal attributes. Suppose that we have the sample data of Table 2.4, except that only the *object-identifier* and the attribute *test-1* are available, where *test-1* is nominal. (We will use *test-2* and *test-3* in later examples.) Let's compute the dissimilarity matrix Eq. (2.14), that is,

$$\begin{bmatrix} 0 & & & \\ d(2,1) & 0 & & \\ d(3,1) & d(3,2) & 0 & \\ d(4,1) & d(4,2) & d(4,3) & 0 \end{bmatrix}.$$

Since here we have one nominal attribute, *test-1*, we set $p = 1$ in Eq. (2.16) so that $d(i, j)$ evaluates to 0 if objects i and j match, and 1 if the objects differ. Thus, we get

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

From this, we see that all objects are dissimilar except objects 1 and 4 (i.e., $d(4, 1) = 0$). □

Alternatively, similarity can be computed as

$$sim(i, j) = 1 - d(i, j) = \frac{m}{p}. \tag{2.17}$$

Table 2.4 A sample data table containing attributes of mixed types.

Object Identifier	Test-1 (nominal)	Test-2 (ordinal)	Test-3 (numeric)
1	code A	excellent	45
2	code B	fair	22
3	code C	good	64
4	code A	excellent	28

Proximity between objects described by nominal attributes can be computed using an alternative encoding scheme. Nominal attributes can be encoded using asymmetric binary attributes by creating a new binary attribute for each of the M states. For an object with a given state value, the binary attribute representing that state is set to 1, whereas the remaining binary attributes are set to 0. For example, to encode the nominal attribute *map_color*, a binary attribute can be created for each of the five colors previously listed. For an object having the color *yellow*, the *yellow* attribute is set to 1, whereas the remaining four attributes are set to 0. Proximity measures for this form of encoding can be calculated using the methods discussed in the next subsection.

2.3.3 Proximity measures for binary attributes

Let's look at dissimilarity and similarity measures for objects described by either *symmetric* or *asymmetric binary attributes*.

Recall that a binary attribute has only one of two states, 0 and 1, where 0 means that the attribute is absent, and 1 means that it is present (Section 2.1.2). Given the attribute *smoker* describing a patient, for instance, 1 indicates that the patient smokes, whereas 0 indicates that the patient does not. Treating binary attributes as if they are other numeric attributes can be misleading. Therefore methods specific to binary data are necessary for computing dissimilarity.

"So, how can we compute the dissimilarity between two binary attributes?" One approach involves computing a dissimilarity matrix from the given binary data. If all binary attributes are thought of as having the same weight, we have the 2 × 2 contingency table of Table 2.5, where q is the number of attributes that equal 1 for both objects i and j, r is the number of attributes that equal 1 for object i but equal 0 for object j, s is the number of attributes that equal 0 for object i but equal 1 for object j, and t is the number of attributes that equal 0 for both objects i and j. The total number of attributes is p, where $p = q + r + s + t$.

Recall that for symmetric binary attributes, each state is equally valuable. Dissimilarity that is based on symmetric binary attributes is called **symmetric binary dissimilarity**. If objects i and j are described by symmetric binary attributes, then the dissimilarity between i and j is

$$d(i, j) = \frac{r + s}{q + r + s + t}. \tag{2.18}$$

For asymmetric binary attributes, the two states are not equally important, such as the *positive* (1) and *negative* (0) outcomes of a disease test. Given two asymmetric binary attributes, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore such binary attributes are often considered "monary" (having one state). The dissimilarity

Table 2.5 Contingency table for binary attributes.

		Object j		
		1	0	sum
Object i	1	q	r	$q+r$
	0	s	t	$s+t$
	sum	$q+s$	$r+t$	p

based on these attributes is called **asymmetric binary dissimilarity**, where the number of negative matches, t, is considered unimportant and is thus ignored in the following computation:

$$d(i, j) = \frac{r+s}{q+r+s}. \tag{2.19}$$

Complementarily, we can measure the difference between two binary attributes based on the notion of similarity instead of dissimilarity. For example, the **asymmetric binary similarity** between the objects i and j can be computed as

$$sim(i, j) = \frac{q}{q+r+s} = 1 - d(i, j). \tag{2.20}$$

The coefficient $sim(i, j)$ of Eq. (2.20) is called the **Jaccard coefficient** and is popularly referenced in the literature.

When both symmetric and asymmetric binary attributes occur in the same data set, the approach for mixed attributes described in Section 2.3.6 can be applied.

Example 2.19. Dissimilarity between binary attributes. Suppose that a patient record table (Table 2.6) contains the attributes *name, gender, fever, cough, test-1, test-2, test-3*, and *test-4*, where *name* is an object identifier, *gender* is a symmetric binary attribute, and the remaining attributes are asymmetric binary. □

For asymmetric binary attribute values, let the values Y (*yes*) and P (*positive*) be set to 1, and the value N (*no* or *negative*) be set to 0. Suppose that the distance between objects (patients) is computed based only on the asymmetric binary attributes. According to Eq. (2.19), the distance between each pair of the three patients—Jack, Mary, and Jim—is

$$d(Jack, Jim) = \frac{1+1}{1+1+1} = 0.67,$$
$$d(Jack, Mary) = \frac{0+1}{2+0+1} = 0.33,$$
$$d(Jim, Mary) = \frac{1+2}{1+1+2} = 0.75.$$

These measurements suggest that Jim and Mary are unlikely to have a similar disease because they have the highest dissimilarity value among the three pairs. Of the three patients, Jack and Mary are the most likely to have a similar disease.

Table 2.6 Relational table where patients are described by binary attributes.

Name	Gender	Fever	Cough	Test-1	Test-2	Test-3	Test-4
Jack	M	Y	N	P	N	N	N
Jim	M	Y	Y	N	N	N	N
Mary	F	Y	N	P	N	P	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

2.3.4 Dissimilarity of numeric data: Minkowski distance

In this section, we describe distance measures that are commonly used for computing the dissimilarity of objects described by numeric attributes. These measures include the *Euclidean, Manhattan*, and *Minkowski distances*.

In some cases, the data are normalized before applying distance calculations. This involves transforming the data to fall within a smaller or common range, such as [−1.0, 1.0] or [0.0, 1.0]. Consider a *height* attribute, for example, which could be measured in either meters or inches. In general, expressing an attribute in smaller units will lead to a larger range for that attribute and thus tend to give such attributes greater effect or "weight." Normalizing the data attempts to give all attributes an equal weight. It may or may not be useful in a particular application. Methods for normalizing data are discussed in detail in Section 2.5 on data transformation.

The most popular distance measure is **Euclidean distance** (i.e., straight line or "as the crow flies"). Let $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ be two objects described by p numeric attributes. The Euclidean distance between objects i and j is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}. \tag{2.21}$$

Another well-known measure is the **Manhattan (or city block) distance**, named so because it is the distance in blocks between any two points in a city (such as 2 blocks down and 3 blocks over for a total of 5 blocks). It is defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|. \tag{2.22}$$

Both the Euclidean and the Manhattan distance satisfy the following mathematical properties:

Nonnegativity: $d(i, j) \geq 0$: Distance is a nonnegative number.
Identity of indiscernibles: $d(i, i) = 0$: The distance of an object to itself is 0.
Symmetry: $d(i, j) = d(j, i)$: Distance is a symmetric function.
Triangle inequality: $d(i, j) \leq d(i, k) + d(k, j)$: Going directly from object i to object j in space is no more than making a detour over any other object k.

A measure that satisfies these conditions is known as **metric**. Please note that the nonnegativity property is implied by the other three properties.

Example 2.20. Euclidean distance and Manhattan distance. Let $x_1 = (1, 2)$ and $x_2 = (3, 5)$ represent two objects as shown in Fig. 2.10. The Euclidean distance between the two is $\sqrt{2^2 + 3^2} = 3.61$. The Manhattan distance between the two is $2 + 3 = 5$. □

Minkowski distance is a generalization of the Euclidean and Manhattan distances. It is defined as

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}, \tag{2.23}$$

where h is a real number such that $h \geq 1$. (Such a distance is also called L_p **norm** in some literature, where the symbol p refers to our notation of h. We have kept p as the number of attributes to be consistent with the rest of this chapter.) It represents the Manhattan distance when $h = 1$ (i.e., L_1 norm) and Euclidean distance when $h = 2$ (i.e., L_2 norm).

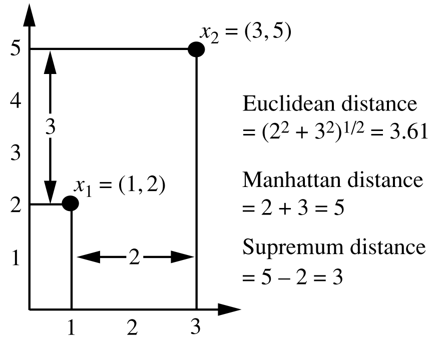


FIGURE 2.10

Euclidean, Manhattan, and supremum distances between two objects.

The **supremum distance** (also referred to as $\boldsymbol{L_{max}}$, $\boldsymbol{L_\infty}$ **norm**, and the **Chebyshev distance**) is a generalization of the Minkowski distance for $h \to \infty$. To compute it, we find the attribute f that gives the maximum difference in values between the two objects. This difference is the supremum distance, defined more formally as:

$$d(i, j) = \lim_{h\to\infty} \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^h \right)^{\frac{1}{h}} = \max_f^p |x_{if} - x_{jf}|. \tag{2.24}$$

The L_∞ norm is also known as the *uniform norm*.

Example 2.21. Supremum distance. Let's use the same two objects, $x_1 = (1, 2)$ and $x_2 = (3, 5)$, as in Fig. 2.10. The second attribute gives the greatest difference between the values for the objects. That is, $max\{(|3 - 1|, |5 - 2|\} = 3$. This is the supremum distance between the two objects. □

If each attribute is assigned a weight according to its perceived importance, the **weighted Euclidean distance** can be computed as

$$d(i, j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_m|x_{ip} - x_{jp}|^2}. \tag{2.25}$$

Weighting can also be applied to other distance measures as well.

2.3.5 Proximity measures for ordinal attributes

The values of an ordinal attribute have a meaningful order or ranking about them, yet the magnitude between successive values is unknown (Section 2.1.3). An example includes the sequence *small*, *medium*, *large* for a *size* attribute. Ordinal attributes may also be obtained from the discretization of numeric attributes by splitting the value range into a finite number of categories. These categories are organized into ranks. That is, the range of a numeric attribute can be mapped to an ordinal attribute f having M_f

states. For example, the range of the interval-scaled attribute *temperature* (in Celsius) can be organized into the following states: −30 to −10, −10 to 10, and 10 to 30, representing the categories *cold temperature*, *moderate temperature*, and *warm temperature*, respectively. Let M_f represent the number of possible states that an ordinal attribute can have. These ordered states define the ranking $1, \ldots, M_f$.

"How are ordinal attributes handled?" The treatment of ordinal attributes is quite similar to that of numeric attributes when computing dissimilarity between objects. Suppose that f is an attribute from a set of ordinal attributes describing n objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the ith object is x_{if}, and f has M_f ordered states, representing the ranking $1, \ldots, M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \ldots, M_f\}$.
2. Since each ordinal attribute can have a different number of states, it is often necessary to map the range of each attribute onto [0.0, 1.0] so that each attribute has equal weight. We perform such data normalization by replacing the rank r_{if} of the ith object in the fth attribute by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \tag{2.26}$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 2.3.4 for numeric attributes, using z_{if} to represent the f value for the ith object.

Example 2.22. Dissimilarity between ordinal attributes. Suppose that we have the sample data shown earlier in Table 2.4, except that this time only the *object-identifier* and the continuous ordinal attribute, *test-2*, are available. There are three states for *test-2*: *fair*, *good*, and *excellent*, that is, $M_f = 3$. For step 1, if we replace each value for *test-2* by its rank, the four objects are assigned the ranks 3, 1, 2, and 3, respectively. Step 2 normalizes the ranking by mapping rank 1 to 0.0, rank 2 to 0.5, and rank 3 to 1.0. For step 3, we can use, say, the Euclidean distance defined in Eq. (2.21), which results in the following dissimilarity matrix:

$$\begin{bmatrix} 0 & & & \\ 1.0 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}.$$

Therefore objects 1 and 2 are the most dissimilar, as are objects 2 and 4 (i.e., $d(2, 1) = 1.0$ and $d(4, 2) = 1.0$). This makes intuitive sense since objects 1 and 4 are both *excellent*. Object 2 is *fair*, which is at the opposite end of the range of values for *test-2*. □

Similarity values for ordinal attributes can be interpreted from dissimilarity as $sim(i, j) = 1 - d(i, j)$.

2.3.6 Dissimilarity for attributes of mixed types

Sections 2.3.2 through 2.3.5 discussed how to compute the dissimilarity between objects described by attributes of the same type, where these types may be either *nominal*, *symmetric binary, asymmetric binary*, *numeric*, or *ordinal*. However, in many real databases, objects are described by a *mixture* of attribute types. In general, a database can contain all of these attribute types.

"So, how can we compute the dissimilarity between objects of mixed attribute types?" One approach is to group each type of attributes together, performing separate data mining (e.g., clustering) analysis for each type. This is feasible if these analyses derive compatible results. However, in real applications, it is unlikely that a separate analysis per attribute type will generate compatible results.

A more preferable approach is to process all attribute types together, performing a single analysis. One such technique combines the different attributes into a single dissimilarity matrix, bringing all of the meaningful attributes onto a common scale of the interval [0.0, 1.0].

Suppose that the data set contains p attributes of mixed types. The dissimilarity $d(i, j)$ between objects i and j is defined as

$$d(i, j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}, \tag{2.27}$$

where the indicator $\delta_{ij}^{(f)} = 0$ if either (1) x_{if} or x_{jf} is missing (i.e., there is no measurement of attribute f for object i or object j), or (2) $x_{if} = x_{jf} = 0$ and attribute f is asymmetric binary; otherwise, $\delta_{ij}^{(f)} = 1$. The contribution of attribute f to the dissimilarity between i and j (i.e., $d_{ij}^{(f)}$) is computed dependent on its type:

- If f is numeric: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_f - min_f}$, where max_f and min_f are the maximum and minimum values of attribute f, respectively;
- If f is nominal or binary: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$; otherwise, $d_{ij}^{(f)} = 1$; and
- If f is ordinal: compute the ranks r_{if} and $z_{if} = \frac{r_{if}-1}{M_f - 1}$, and treat z_{if} as numeric.

These steps are identical to what we have already seen for each of the individual attribute types. The only difference is for numeric attributes, where we normalize so that the values map to the interval [0.0, 1.0]. Thus the dissimilarity between objects can be computed even when the attributes describing the objects are of different types.

Example 2.23. Dissimilarity between attributes of mixed types. Let's compute a dissimilarity matrix for the objects in Table 2.4. Now we will consider *all* of the attributes, which are of different types. In Examples 2.18 and 2.22, we worked out the dissimilarity matrices for each of the individual attributes. The procedures we followed for *test-1* (which is nominal) and *test-2* (which is ordinal) are the same as outlined earlier for processing attributes of mixed types. Therefore we can use the dissimilarity matrices obtained for *test-1* and *test-2* later when we compute Eq. (2.27). First, however, we need to compute the dissimilarity matrix for the third attribute, *test-3* (which is numeric). That is, we must compute $d_{ij}^{(3)}$. Following the case for numeric attributes, we let $max_h x_h = 64$ and $min_h x_h = 22$. The difference between the two is used in Eq. (2.27) to normalize the values of the dissimilarity matrix. The resulting dissimilarity matrix for *test-3* is

$$\begin{bmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1.00 & 0 & \\ 0.40 & 0.14 & 0.86 & 0 \end{bmatrix}.$$

We can now use the dissimilarity matrices for the three attributes in our computation of Eq. (2.27). The indicator $\delta_{ij}^{(f)} = 1$ for each of the three attributes, f. We get, for example, $d(3,1) = \frac{1(1)+1(0.50)+1(0.45)}{3} = 0.65$. The resulting dissimilarity matrix obtained for the data described by the three attributes of mixed types is:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{bmatrix}.$$

From Table 2.4, we can intuitively guess that objects 1 and 4 are the most similar, based on their values for *test-1* and *test-2*. This is confirmed by the dissimilarity matrix, where $d(4,1)$ is the lowest value for any pair of different objects. Similarly, the matrix indicates that objects 1 and 2 are the least similar. □

2.3.7 Cosine similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus each document is an object represented by what is called a *term-frequency vector*. For example, in Table 2.7, we see that *Document1* contains five instances of the word *team*, whereas *hockey* occurs three times. The word *coach* is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

Term-frequency vectors are typically very long and **sparse** (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, and biological data analysis. The traditional distance measures that we have studied in this chapter do not work well for such sparse numeric data. For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar. We need a measure that will focus on the words that the two documents *do* have in common, and the occurrence frequency of such words. In other words, we need a measure for numeric data that ignores zero-matches.

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors for

Table 2.7 Document vector or term-frequency vector.

Document	Team	Coach	Hockey	Baseball	Soccer	Penalty	Score	Win	Loss	Season
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

comparison. Using the cosine measure as a similarity function, we have

$$sim(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}, \tag{2.28}$$

where $||\boldsymbol{x}||$ is the Euclidean norm of vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$, defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}$. Conceptually, it is the length of the vector. Similarly, $||\boldsymbol{y}||$ is the Euclidean norm of vector $\boldsymbol{y}$. The measure computes the cosine of the angle between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors. Note that because the cosine similarity measure does not obey all of the properties of Section 2.3.4 defining metric measures, it is referred to as a *nonmetric measure*.

Example 2.24. Cosine similarity between two term-frequency vectors. Suppose that $\boldsymbol{x}$ and $\boldsymbol{y}$ are the first two term-frequency vectors in Table 2.7. That is, $\boldsymbol{x} = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0)$ and $\boldsymbol{y} = (3, 0, 2, 0, 1, 1, 0, 1, 0, 1)$. How similar are $\boldsymbol{x}$ and $\boldsymbol{y}$? Using Eq. (2.28) to compute the cosine similarity between the two vectors, we get:

$$\begin{aligned}
\boldsymbol{x}\cdot\boldsymbol{y} &= 5\times3+0\times0+3\times2+0\times0+2\times1+0\times1+0\times0+2\times1\\
&\quad +0\times0+0\times1 = 25\\
||\boldsymbol{x}|| &= \sqrt{5^2+0^2+3^2+0^2+2^2+0^2+0^2+2^2+0^2+0^2} = 6.48\\
||\boldsymbol{y}|| &= \sqrt{3^2+0^2+2^2+0^2+1^2+1^2+0^2+1^2+0^2+1^2} = 4.12\\
sim(\boldsymbol{x},\boldsymbol{y}) &= 0.94.
\end{aligned}$$

Therefore if we were using the cosine similarity measure to compare these documents, they would be considered quite similar. □

When attributes are binary-valued, the cosine similarity function can be interpreted in terms of shared features or attributes. Suppose an object $\boldsymbol{x}$ possesses the ith attribute if $x_i = 1$. Then $\boldsymbol{x}\cdot\boldsymbol{y}$ is the number of attributes possessed (i.e., shared) by both $\boldsymbol{x}$ and $\boldsymbol{y}$, and $||\boldsymbol{x}||$ and $||\boldsymbol{y}||$ are the *geometric mean* of the number of attributes possessed by $\boldsymbol{x}$ and that by $\boldsymbol{y}$ respectively. Thus, $sim(\boldsymbol{x},\boldsymbol{y})$ is a measure of relative possession of common attributes.

A simple variation of cosine similarity for the preceding scenario is

$$sim(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\boldsymbol{x}\cdot\boldsymbol{x}+\boldsymbol{y}\cdot\boldsymbol{y}-\boldsymbol{x}\cdot\boldsymbol{y}}, \tag{2.29}$$

which is the ratio of the number of attributes shared by $\boldsymbol{x}$ and $\boldsymbol{y}$ to the number of attributes possessed by $\boldsymbol{x}$ or $\boldsymbol{y}$. This function, known as the **Tanimoto coefficient** or **Tanimoto distance**, is frequently used in information retrieval and biology taxonomy.

2.3.8 Measuring similar distributions: the Kullback-Leibler divergence

Finally, we introduce *Kullback-Leibler divergence*, or simply, the *KL divergence*, a measure that has been popularly used in the data mining literature to measure the difference between two probability distributions over the same variable x. This concept was originated in probability theory and information theory.

The KL divergence, which is closely related to *relative entropy*, *information divergence*, and *information for discrimination*, is a nonsymmetric measure of the difference between two probability distributions $p(x)$ and $q(x)$. Specifically, the KL divergence of $q(x)$ from $p(x)$, denoted $D_{KL}(p(x)||q(x))$, is a measure of the information loss when $q(x)$ is used to approximate $p(x)$.

Let $p(x)$ and $q(x)$ be two probability distributions of a discrete random variable x. That is, both $p(x)$ and $q(x)$ sum up to 1, and $p(x) > 0$ and $q(x) > 0$ for any x in X. $D_{KL}(p(x)||q(x))$ is defined in Eq. (2.30).

$$D_{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)} \tag{2.30}$$

The KL divergence measures the expected number of extra bits required to code samples from $p(x)$ when using a code based on $q(x)$ rather than using a code based on $p(x)$. Typically $p(x)$ represents the "true" distribution of data, observations, or a precisely calculated theoretical distribution. The measure $q(x)$ typically represents a theory, model, description, or approximation of $p(x)$.

The continuous version of the KL divergence is

$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx. \tag{2.31}$$

Although the KL divergence measures the "distance" between two distributions, it is not a distance measure. This is because that the KL divergence is not a metric measure. It is not symmetric: the KL from $p(x)$ to $q(x)$ is generally not the same as the KL from $q(x)$ to $p(x)$. Furthermore, it need not satisfy triangular inequality. Nevertheless, $D_{KL}(p(x)||q(x))$ is a nonnegative measure. $D_{KL}(p(x)||q(x)) \geq 0$ and $D_{KL}(p(x)||q(x)) = 0$ if and only if $p(x) = q(x)$.

Notice that attention should be paid when computing the KL divergence. We know $lim_{p(x) \to 0}$ $p(x) \log p(x) = 0$. However, when $p(x) \neq 0$ but $q(x) = 0$, $D_{KL}(p(x)||q(x))$ is defined as ∞. This means that if one event e is possible (i.e., $p(e) > 0$), and the other predicts it is absolutely impossible (i.e., $q(e) = 0$), then the two distributions are absolutely different. However, in practice, two distributions P and Q are derived from observations and sample counting, that is, from frequency distributions. It is unreasonable to predict in the derived probability distribution that an event is completely impossible since we must take into account the possibility of unseen events. A *smoothing* method can be used to derive the probability distribution from an observed frequency distribution, as illustrated in the following example.

Example 2.25. Computing the KL divergence by smoothing. Suppose there are two sample distributions P and Q as follows: $P : (a : 3/5, b : 1/5, c : 1/5)$ and $Q : (a : 5/9, b : 3/9, d : 1/9)$. To compute the KL divergence $D_{KL}(P||Q)$, we introduce a small constant ϵ, for example $\epsilon = 10^{-3}$, and define a smoothed version of P and Q, P' and Q', as follows.

The sample set observed in P, $SP = \{a, b, c\}$. Similarly, $SQ = \{a, b, d\}$. The union set is $SU = \{a, b, c, d\}$. By smoothing, the missing symbols can be added to each distribution accordingly, with the small probability ϵ. Thus we have $P' : (a : 3/5 - \epsilon/3, b : 1/5 - \epsilon/3, c : 1/5 - \epsilon/3, d : \epsilon)$ and $Q' : (a : 5/9 - \epsilon/3, b : 3/9 - \epsilon/3, c : \epsilon, d : 1/9 - \epsilon/3)$. $D_{KL}(P', Q')$ can be computed easily. □

2.3.9 Capturing hidden semantics in similarity measures

Similarity measure is a fundamental concept in data mining. We have introduced multiple measures for computing similarities among objects consisting of numerical attribute, symmetric and asymmetric binary attribute, ordinal attribute, and nominal attribute. We have also introduced how to compute document similarity using the vector space model, and how to compare two distributions using the notion of KL divergence. These notions and measures on object similarity will be used substantially in our subsequent studies on methods for pattern discovery, classification, clustering, and outlier analysis.

In real-life applications, we may encounter the notion of object similarity beyond what we have discussed in this chapter. Even for simple objects, similarities among objects are often closely related to their semantic meanings, which cannot be captured based on the above defined similarity measures. For example, people often consider *geometry* and *algebra* are more similar than *geometry* vs. *music* or *politics*, even all are subjects studied in schools. Moreover, documents that consist of similar frequency distributions of words (or similar *bags* of words) may express rather different meanings (e.g., considering "The cat bites a mouse" vs. "The mouse bites a cat"). This goes beyond what a *vector space model* (i.e., expressing words as a set of vectors in a high-dimensional vector space as shown in Section 2.3.7) can handle. Furthermore, objects can be composed of rather complex structures and connections. Similarity measures for graphs and networks may need to be introduced, which is beyond the notions of object similarity introduced here.

In the upcoming chapters, we will introduce additional similarity measures when encountered along with the problems and methods to be discussed. In particular, in Chapter 12, we will briefly introduce the notion of distributive representation and representation learning, where text embedding and deep learning will be used to compute such advanced notion of similarities.

2.4 Data quality, data cleaning, and data integration

In this section, we start with a discussion of data quality measures (Section 2.4.1). Then, we introduce common techniques for data cleaning (Section 2.4.2) and data integration (Section 2.4.3).

2.4.1 Data quality measures

Data have quality if they satisfy the requirements of the intended use. There are many factors comprising **data quality**, including *accuracy, completeness, consistency, timeliness, believability*, and *interpretability*.

Imagine that you are a manager at an online webstore and have been charged with analyzing the company's data with respect to your branch's sales. You immediately set out to perform this task. You carefully inspect the company's database and data warehouse, identifying and selecting the attributes or dimensions (e.g., *item*, *price*, and *units_sold*) to be included in your analysis. Alas! You notice that several of the attributes for various tuples have no recorded values. For your analysis, you would like to include information as to whether each item purchased was advertised as on sale, yet you discover that this information has not been recorded. Furthermore, users of your database system have reported errors, unusual values, and inconsistencies in the data recorded for some transactions. In other words, the data you wish to analyze by data mining techniques are *incomplete* (lacking attribute values or certain attributes of interest, or containing only aggregate data); *inaccurate* or *noisy* (containing errors, or values that deviate from the expected); and *inconsistent* (e.g., containing discrepancies in the department codes used to categorize items). Welcome to the real world!

This scenario illustrates three of the elements defining data quality: **accuracy**, **completeness**, and **consistency**. Inaccurate, incomplete, and inconsistent data are commonplace properties of large real-world databases and data warehouses. There are many possible reasons for inaccurate data (i.e., having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Users may purposely submit incorrect data values for mandatory fields when they do not wish to submit personal information (e.g., by choosing the default value "January 1" displayed for birthday). This is known as *disguised missing data*. Errors in data transmission can also occur. There may be technology limitations such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or data codes or inconsistent formats for input fields (e.g., *date*). Duplicate tuples also require data cleaning.

Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because they were not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding or because of equipment malfunctions. Data that were inconsistent with other recorded data may have been deleted. Furthermore, the recording of the data history or modifications may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

Recall that data quality depends on the intended use of the data. Two different users may have very different assessments of the quality of a given database. For example, a marketing analyst may need to access the database mentioned before for a list of customer addresses. Some of the addresses are outdated or incorrect, yet overall, 80% of the addresses are accurate. The marketing analyst considers this to be a large customer database for target marketing purposes and is pleased with the database's accuracy, although as sales manager, you found the data inaccurate.

Timeliness also affects data quality. Suppose that you are overseeing the distribution of monthly sales bonuses to the top sales representatives in a company. Several sales representatives, however, fail to submit their sales records on time at the month-end. There are also a number of corrections and adjustments that flow in after the month-end. For a period of time following each month, the data stored in the database are incomplete. However, once all of the data are received, it is correct. The fact that the month-end data are not updated in a timely fashion has a negative impact on the data quality.

Two other factors affecting data quality are believability and interpretability. **Believability** reflects how much the data are trusted by users, whereas **interpretability** reflects how easily the data are understood. Suppose that a database, at one point, had several errors, all of which have since been corrected. The past errors, however, had caused many problems for sales department users, and so they no longer trust the data. The data also use many accounting codes, which the sales department does not know how to interpret. Even though the database is now accurate, complete, consistent, and timely, sales department users may regard it as of low quality due to poor believability and interpretability.

2.4.2 Data cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. *Data cleaning* (or *data cleansing*) routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. In this section, you will study basic methods for data cleaning. First, we look at ways of handling missing values. Then, we explain data smoothing techniques. Finally, we discuss approaches to data cleaning as a process.

Missing values

Imagine that you need to analyze the sales and customer data of a company. You note that many tuples have no recorded value for several attributes such as customer *income*. How can you go about filling in the missing values for this attribute? Let's look at the following methods.

1. **Ignore the tuple**: This is usually done when the class label is missing (assuming the mining task involves classification). This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably. By ignoring the tuple, we do not make use of the remaining attributes' values in the tuple. Such data could have been useful to the task at hand.
2. **Fill in the missing value manually**: In general, this approach is time consuming and may not be feasible given a large data set with many missing values.
3. **Use a global constant to fill in the missing value**: Replace all missing attribute values by the same constant such as a label like *"Unknown"* or $-\infty$. If missing values are replaced by, say, *"Unknown*," then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common—that of *"Unknown."* Hence, although this method is simple, it is not foolproof.
4. **Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value**: Section 2.2 discussed measures of central tendency, which indicate the "middle" value of a data distribution. For normal (symmetric) data distributions, the mean can be used, whereas skewed data distribution should employ the median (Section 2.2). For example, suppose that the data distribution regarding the income of the customers is symmetric and that the mean income is $56,000. Use this value to replace the missing value for *income*.
5. **Use the attribute mean or median for all samples belonging to the same class as the given tuple**: For example, if classifying customers according to *credit_risk*, we may replace the missing value with the mean *income* value for customers in the same credit risk category as that of the given tuple. If the data distribution for a given class is skewed, the median value is a better choice.
6. **Use the most probable value to fill in the missing value**: This may be determined with regression, inference-based tools using a Bayesian formalism or decision tree induction. For example, using the other customer attributes in your data set, you may construct a decision tree to predict the missing values for *income*. Decision trees, regression, and Bayesian inference are described in detail in Chapters 6 and 7.

Methods 3 through 6 bias the data—the filled-in value may not be correct. Method 6, however, is a popular strategy. In comparison to the other methods, it uses the most information from the present data to predict missing values. By considering the values of other attributes in its estimation of the missing value for *income*, there is a greater chance that the relationships between *income* and the other attributes are preserved.

It is important to note that, in some cases, a missing value may not imply an error in the data! For example, when applying for a credit card, candidates may be asked to supply their driver's license number. Candidates who do not have a driver's license may naturally leave this field blank. Forms should allow respondents to specify values such as "not applicable." Software routines may also be used to uncover other null values (e.g., “don’t know,” “?”, or “none”). Ideally, each attribute should have one or more rules regarding the *null* condition. The rules may specify whether or not nulls are allowed and/or how such values should be handled or transformed. Fields may also be intentionally left blank if they are to be provided in a later step of the business process. Hence, although we can try our best to clean the data after it is seized, good database and data entry procedure design should help minimize the number of missing values or errors in the first place.

Noisy data

“What is noise?” **Noise** is a random error or variance in a measured variable. Given a numeric attribute such as, say, *price*, how can we “smooth” out the data to remove the noise? Let’s look at the following data smoothing techniques.

Binning: Binning methods smooth a sorted data value by consulting its “neighborhood,” that is, the values around it. The sorted values are distributed into a number of “buckets,” or *bins*. Because binning methods consult the neighborhood of values, they perform *local* smoothing. Fig. 2.11 illustrates some binning techniques. In this example, the data for *price* are first sorted and then partitioned into *equal-frequency* bins of size 3 (i.e., each bin contains three values). In **smoothing by bin means**, each value in a bin is replaced by the mean value of the bin. For example, the mean of the values 4, 8, and 15 in Bin 1 is 9. Therefore each original value in this bin is replaced by the value 9.

Similarly, **smoothing by bin medians** can be employed, in which each bin value is replaced by the bin median. In **smoothing by bin boundaries**, the minimum and maximum values in a given bin are identified as the *bin boundaries*. Each bin value is then replaced by the closest boundary value. In general, the larger the width, the greater the effect of the smoothing. Alternatively, bins may be

Sorted data for *price* (in dollars): 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equal-frequency) bins:

Bin 1: 4, 8, 15
Bin 2: 21, 21, 24
Bin 3: 25, 28, 34

Smoothing by bin means:

Bin 1: 9, 9, 9
Bin 2: 22, 22, 22
Bin 3: 29, 29, 29

Smoothing by bin boundaries:

Bin 1: 4, 4, 15
Bin 2: 21, 21, 24
Bin 3: 25, 25, 34

FIGURE 2.11

Data smoothing with different binning methods.

equal width, where the interval range of values in each bin is constant. Binning is also used as a discretization technique.

Regression: Data smoothing can also be done by regression, a technique that conforms data values to a function. *Linear regression* involves finding the "best" line to fit two attributes (or variables) so that one attribute can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two attributes are involved, and the data are fit to a multidimensional surface. Regression is further described in Chapter 6.

Outlier analysis: Outliers may be detected by clustering, for example, where similar values are organized into groups or "clusters." Intuitively, values that fall outside of the set of clusters may be considered as outliers (Fig. 2.12). Chapter 11 is dedicated to the topic of outlier analysis.

Many data smoothing methods are also used for data discretization (a form of data transformation) and data reduction. For example, the binning techniques described before reduce the number of distinct values per attribute. This acts as a form of data reduction for logic-based data mining methods, such as decision tree induction, which repeatedly makes value comparisons on sorted data. Concept hierarchies are a form of data discretization that can also be used for data smoothing. A concept hierarchy for *price*, for example, may map real *price* values into *inexpensive, moderately_priced*, and *expensive*, thereby reducing the number of data values to be handled by the mining process. Data discretization is discussed in Section 2.5.2. Some methods of classification have built-in data smoothing mechanisms. Classification is the topic of Chapters 6 and 7.

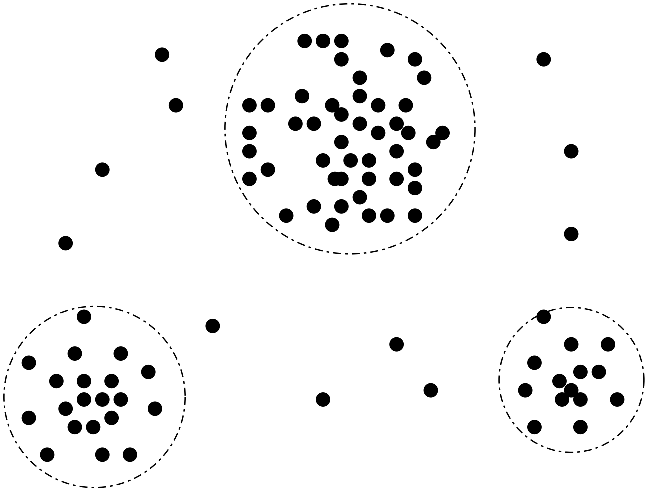


FIGURE 2.12

A 2-D customer data plot with respect to customer locations in a city, showing three data clusters. Outliers may be detected as values that fall outside of the cluster sets.

Data cleaning as a process

Missing values, noise, and inconsistencies contribute to inaccurate data. So far, we have looked at techniques for handling missing data and for smoothing data. *"But data cleaning is a big job. What about data cleaning as a process? How exactly does one proceed in tackling this task? Are there any tools out there to help?"*

The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and inconsistent use of codes. Other sources of discrepancies include errors in instrumentation devices that record data and system errors. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).[1]

"So, how can we proceed with discrepancy detection?" As a starting point, use any knowledge you may already have regarding properties of the data. Such knowledge or "data about data" is referred to as **metadata**. This is where we can make use of the knowledge we gained about our data in the earlier sections. For example, what are the data type and domain of each attribute? What are the acceptable values for each attribute? The basic statistical data descriptions discussed in Section 2.2 are useful here to grasp data trends and identify anomalies. For example, find the mean, median, and mode values. Are the data symmetric or skewed? What is the range of values? Do all values fall within the expected range? What is the standard deviation of each attribute? For Gaussian-like distributions, values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. Are there any known dependencies between attributes? In this step, you may write your own scripts and/or use some of the tools that we discuss further later. From this, you may find noise, outliers, and unusual values that need investigation.

As a data analyst, you should be on the lookout for the inconsistent use of codes and any inconsistent data representations (e.g., "2010/12/25" and "25/12/2010" for *date*). **Field overloading** is another error source that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., an unused bit of an attribute that has a value range that uses only, say, 31 out of 32 bits).

The data should also be examined regarding uniqueness, consecutiveness, and null conditions. A **uniqueness rule** says that each value of the given attribute must be different from all other values for that attribute. A **consecutiveness rule** says that there can be no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers). A **null condition rule** specifies the use of blanks, question marks, special characters, or other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled. As mentioned earlier, reasons for missing values may include: (1) the person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a *license_number* attribute left blank by nondrivers); (2) the data entry person does not know the correct value; or (3) the value is to be provided by a later step of the process. The null rule should specify how to record the null condition, for example, such as to store zero for numeric

[1] Data integration and the removal of redundant data that can result from such integration are further described in Section 2.4.3.

attributes, a blank for categorical attributes, or any other conventions that may be in use (e.g., entries like “don’t know” or “?” should be transformed to blank).

There are a number of different commercial tools that can aid in the discrepancy detection step. **Data scrubbing tools** use simple domain knowledge (e.g., knowledge of postal addresses and spell-checking) to detect errors and make corrections in the data. These tools rely on parsing and fuzzy matching techniques when cleaning data from multiple sources. **Data auditing tools** find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions. They are variants of data mining tools. For example, they may employ statistical analysis to find correlations, or clustering to identify outliers. They may also use the basic statistical data descriptions presented in Section 2.2.

Some data inconsistencies may be corrected manually using external references. For example, errors made at data entry may be corrected by performing a paper trace. Most errors, however, will require *data transformations*. That is, once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them.

Commercial tools can assist in the data transformation step. **Data migration tools** allow simple transformations to be specified such as to replace the string *“gender”* by *“sex.”* **ETL (extraction/transformation/loading) tools** allow users to specify transforms through a graphical user interface (GUI). These tools typically support only a restricted set of transformations so that often we may also choose to write custom scripts for this step of the data cleaning process.

The two-step process of discrepancy detection and data transformation (to correct discrepancies) iterates. This process, however, is error-prone and time-consuming. Some transformations may introduce more discrepancies. Some *nested discrepancies* may only be detected after others have been fixed. For example, a typo such as “20010” in a year field may only surface once all date values have been converted to a uniform format. Transformations are often done as a batch process while the user waits without feedback. Only after the transformation is complete can the user go back and check that no new anomalies have been mistakenly created. Typically, numerous iterations are required before the user is satisfied. Any tuples that cannot be automatically handled by a given transformation are typically written to a file without any explanation regarding the reasoning behind their failure. As a result, the entire data cleaning process also suffers from a lack of interactivity.

New approaches to data cleaning emphasize increased interactivity. Potter’s Wheel, for example, is a publicly available data cleaning tool that integrates discrepancy detection and transformation. Users gradually build a series of transformations by composing and debugging individual transformations, one step at a time, on a spreadsheet-like interface. The transformations can be specified graphically or by providing examples. Results are shown immediately on the records that are visible on the screen. The user can choose to undo the transformations, so that transformations that have introduced additional errors can be “erased.” The tool automatically performs discrepancy checking in the background on the latest transformed view of the data. Users can gradually develop and refine transformations as discrepancies are found, leading to more effective and efficient data cleaning. Section 2.5 will introduce some common data transformation techniques, including normalization, discretization, compression, and sampling.

Another approach to increasing interactivity in data cleaning is the development of declarative languages for the specification of data transformation operators. Such work focuses on defining powerful extensions to SQL and algorithms that enable users to express data cleaning specifications efficiently.

As we discover more about the data, it is important to keep updating the metadata to reflect this knowledge. This will help speed up data cleaning on future versions of the same data store.

2.4.3 Data integration

Data mining often requires data integration—the merging of data from multiple data stores. Careful integration can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent data mining process.

The semantic heterogeneity and structure of data pose great challenges in data integration. In this section, we first introduce the *entity identification problem*, which matches schema and objects from different sources. Then, we present *correlation tests* for spotting correlated numeric and nominal data. Finally, we introduce *tuple duplication* and the detection and resolution of *data value conflicts*.

Entity identification problem

It is likely that your data analysis task will involve *data integration*, which combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

There are a number of issues to consider during data integration. *Schema integration* and *object matching* can be tricky. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the **entity identification problem**. For example, how can a data analyst or a computer be sure that *customer_id* in one database and *cust_number* in another refer to the same attribute? Moreover, metadata may be used to help entity identification (e.g., data codes for *pay_type* in one database may be *"H"* and *"S"* but *1* and *2* in another). Examples of metadata for each attribute include the name, meaning, data type, range of values permitted for the attribute, and null rules for handling blank, zero, or null values (Section 2.4.2). Such metadata can be used to help avoid errors in schema integration. Hence, this step also relates to data cleaning, as described earlier.

When matching attributes from one database to another during integration, special attention must be paid to the *structure* of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. For example, in one system, a *discount* may be applied to the order, whereas in another system, it is applied to each individual line item within the order. If this is not caught before integration, items in the target system may be improperly discounted.

Redundancy and correlation analysis

Redundancy is another important issue in data integration. An attribute (such as *annual revenue*, for instance) may be redundant if it can be "derived" from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Some redundancies can be detected by **correlation analysis**. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For nominal data, we can use the χ^2 (*chi-square*) test. For numeric attributes, we can use the *correlation coefficient* and *covariance*, both of which assess how one attribute's values vary from those of another.

Tuple duplication

In addition to detecting redundancies between attributes, duplication should also be detected at the tuple level (e.g., where there are two or more identical tuples for a given unique data entry). The use of denormalized tables (often done to improve performance by avoiding joins) is another source of data redundancy. Inconsistencies often arise between various duplicates, due to inaccurate data entry or updating some but not all data occurrences. For example, if a purchase order database contains attributes for the purchaser's name and address instead of a key to this information in a purchaser database, discrepancies can occur, such as the same purchaser's name appearing with different addresses within the purchase order database.

Data value conflict detection and resolution

Data integration also involves the *detection and resolution of data value conflicts*. For example, for the same real-world entity, attribute values from different sources may differ. This may be due to differences in representation, scaling, or encoding. For instance, a *weight* attribute may be stored in metric units in one system and British imperial units in another. For a hotel chain, the *price* of rooms in different cities may involve not only different currencies but also different services (e.g., free breakfast) and taxes. When exchanging information between schools, for example, each school may have its own curriculum and grading scheme. One university may adopt a quarter system, offer three courses on database systems, and assign grades from A+ to F, whereas another may adopt a semester system, offer two courses on databases, and assign grades from 1 to 10. It is difficult to work out precise course-to-grade transformation rules between the two universities, making information exchange difficult.

Attributes may also differ on the abstraction level, where an attribute in one system is recorded at, say, a lower abstraction level than the "same" attribute in another. For example, the *total_sales* in one database may refer to one branch of the company, whereas an attribute of the same name in another database may refer to the total sales for the stores in a given region.

The topic of discrepancy detection was described in Section 2.4.2 on data cleaning as a process.

2.5 Data transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Through appropriate data transformation, the resulting mining process may be more efficient, and the patterns found may be easier to understand. Various strategies for data transformation have been developed. In this section, we start with the introduction of data *normalization* (Section 2.5.1), where the attribute data are scaled so as to fall within a smaller range, such as −1.0 to 1.0 or 0.0 to 1.0. Then, we will learn data *discretization* (Section 2.5.2), which replaces the raw values of a numeric attribute (e.g., *age*) by interval labels (e.g., 0–10, 11–20, etc.) or conceptual labels (e.g., *youth, adult*, *senior*). Data *compression* (Section 2.5.3) and *sampling* (Section 2.5.4) are two data reduction techniques that transform the input data to a reduced representation that is much smaller in volume, yet closely maintains the integrity of the original data.

2.5.1 Normalization

The measurement unit used can affect the data analysis. For example, changing measurement units from meters to inches for *height*, or from kilograms to pounds for *weight*, may lead to very different results. In general, expressing an attribute in smaller units will lead to a larger range for that attribute and thus tend to give such an attribute greater effect or "weight." To help avoid dependence on the choice of measurement units, the data should be *normalized* or *standardized*. This involves transforming the data to fall within a smaller or common range such as [−1.0, 1.0] or [0.0, 1.0]. (The terms *standardize* and *normalize* are used interchangeably in data preprocessing, although in statistics, the latter term also has other connotations.)

Normalizing the data attempts to give all attributes an equal weight. Normalization is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbor classification and clustering. If using the neural network backpropagation algorithm for classification (Chapter 10), normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. For distance-based methods, normalization helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). It is also useful when given no prior knowledge of the data.

There are many methods for data normalization. We study *min-max normalization, z-score normalization,* and *normalization by decimal scaling*. For our discussion, let A be a numeric attribute with n observed values, $v_1, v_2, \ldots, v_n$.

Min-max normalization performs a linear transformation on the original data. Suppose that min_A and max_A are the minimum and maximum values of an attribute, A. Min-max normalization maps a value, v_i, of A to v_i' in the range $[new\_min_A, new\_max_A]$ by computing

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A. \tag{2.32}$$

Min-max normalization preserves the relationships among the original data values. It will encounter an "out-of-bounds" error if a future input case for normalization falls outside of the original data range for A.

Example 2.26. Min-max normalization. Suppose that the minimum and maximum values for the attribute *income* are \$12,000 and \$98,000, respectively. We would like to map *income* to the range [0.0, 1.0]. By min-max normalization, a value of \$73,600 for *income* is transformed to $\frac{73{,}600-12{,}000}{98{,}000-12{,}000}(1.0-0)+0=0.716$. □

In ***z*-score normalization** (or *zero-mean normalization*), the values for an attribute, A, are normalized based on the mean (i.e., average) and standard deviation of A. A value, v_i, of A is normalized to v_i' by computing

$$v_i' = \frac{v_i - \bar{A}}{\sigma_A}, \tag{2.33}$$

where $\bar{A}$ and σ_A are the mean and standard deviation, respectively, of attribute A. The mean and standard deviation were discussed in Section 2.2, where $\bar{A} = \frac{1}{n}(v_1 + v_2 + \cdots + v_n)$, and σ_A is computed as the square root of the variance of A (see Eq. (2.6)). This method of normalization is useful when the actual minimum and maximum of attribute A are unknown or when there are outliers that dominate the min-max normalization.

Example 2.27. z-score normalization. Suppose that the mean and standard deviation of the values for the attribute *income* are \$54,000 and \$16,000, respectively. With z-score normalization, a value of \$73,600 for *income* is transformed to $\frac{73{,}600-54{,}000}{16{,}000} = 1.225$. □

A variation of this z-score normalization replaces the standard deviation of Eq. (2.33) by the *mean absolute deviation* of A. The *mean absolute deviation* of A, denoted s_A, is

$$s_A = \frac{1}{n}(|v_1 - \bar{A}| + |v_2 - \bar{A}| + \cdots + |v_n - \bar{A}|). \tag{2.34}$$

Thus z-score normalization using the mean absolute deviation is

$$v_i' = \frac{v_i - \bar{A}}{s_A}. \tag{2.35}$$

The mean absolute deviation, s_A, is more robust to outliers than the standard deviation, σ_A. When computing the mean absolute deviation, the deviations from the mean (i.e., $|x_i - \bar{x}|$) are not squared; hence, the effect of outliers is somewhat reduced.

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A. The number of decimal points moved depends on the maximum absolute value of A. A value, v_i, of A is normalized to v_i' by computing

$$v_i' = \frac{v_i}{10^j}, \tag{2.36}$$

where j is the smallest integer such that $max(|v_i'|) < 1$.

Example 2.28. Decimal scaling. Suppose that the recorded values of A range from -986 to 917. The maximum absolute value of A is 986. To normalize by decimal scaling, we therefore divide each value by 1000 (i.e., $j = 3$) so that -986 normalizes to -0.986 and 917 normalizes to 0.917. □

Note that normalization can change the original data quite a bit, especially when using z-score normalization or decimal scaling. It is also necessary to save the normalization parameters (e.g., the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

2.5.2 Discretization

Data discretization is a common data transformation technique, where the raw values of a numeric attribute (e.g., *age*) are replaced by interval labels (e.g., 0–10, 11–20, etc.) or conceptual labels (e.g., *youth, adult*, *senior*). The labels, in turn, can be recursively organized into higher-level concepts, resulting in a *concept hierarchy* for the numeric attribute. Fig. 2.13 shows a concept hierarchy for the attribute *price*. More than one concept hierarchy can be defined for the same attribute to accommodate the needs of various users.

Discretization techniques can be categorized based on how the discretization is performed, such as whether it uses class information or which direction it proceeds (i.e., top-down vs. bottom-up). If the discretization process uses class information, then we say it is *supervised discretization*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few points (called *split points* or *cut*

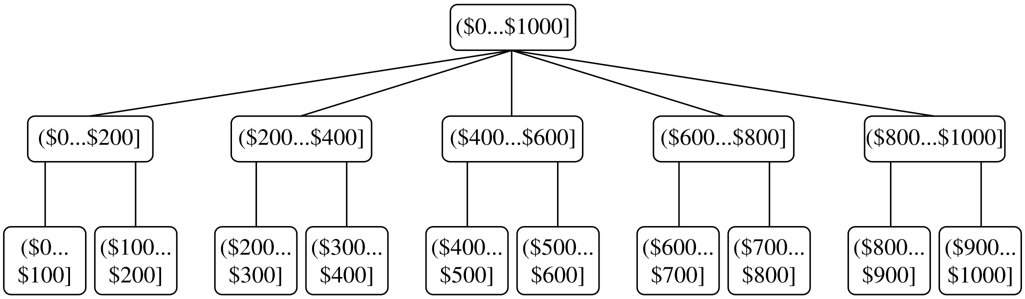


FIGURE 2.13

A concept hierarchy for the attribute *price*, where an interval ($$X \ldots \$Y$] denotes the range from $$X$ (exclusive) to $$Y$ (inclusive).

points) to split the entire attribute range and then repeats this recursively on the resulting intervals, it is called *top-down discretization* or *splitting*. This contrasts with *bottom-up discretization* or *merging*, which starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals.

We introduce two basic discretization techniques, including binning and histogram analysis. Other methods for discretization include cluster analysis, decision tree analysis, and correlation analysis. Each of these techniques can be used to generate concept hierarchies for numeric attributes.

Discretization by binning

Binning is a top-down splitting technique based on a specified number of bins. Section 2.4.2 discussed binning methods for data smoothing. These methods are also used as discretization methods for data reduction and concept hierarchy generation. For example, attribute values can be discretized by applying equal-width or equal-frequency binning and then replacing each bin value by the bin mean or median, as in *smoothing by bin means* or *smoothing by bin medians*, respectively. These techniques can be applied recursively to the resulting partitions to generate concept hierarchies.

Binning does not use class information and is therefore an unsupervised discretization technique. It is sensitive to the user-specified number of bins, as well as the presence of outliers.

Discretization by histogram analysis

Histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms were introduced in Section 2.2.4. A histogram partitions the values of an attribute, A, into disjoint ranges called *buckets* or *bins*. If each bucket represents only a single attribute-value/frequency pair, the buckets are called *singleton buckets*. Singleton buckets are useful for storing high-frequency outliers. Often, buckets instead represent continuous ranges for the given attribute.

Example 2.29. The following data are a list of prices for commonly sold items in the company (rounded to the nearest dollar). The numbers have been sorted: 1, 1, 5, 5, 5, 5, 5, 8, 8, 10, 10, 10, 10, 12, 14, 14,

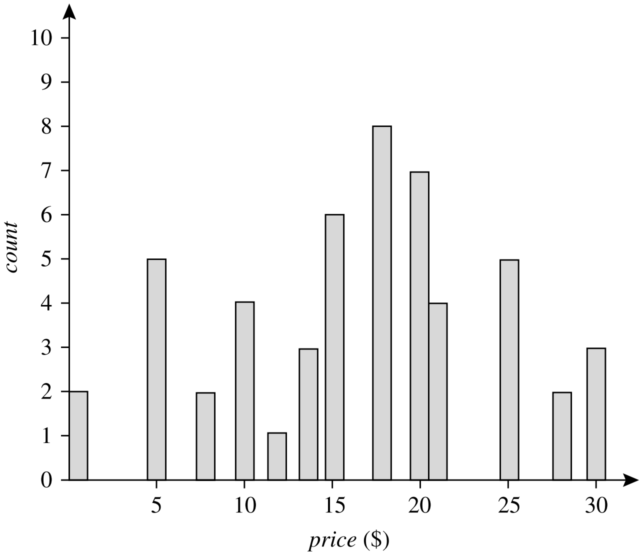


FIGURE 2.14

A histogram for *price* using singleton buckets—each bucket represents one price–value/frequency pair.

14, 15, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.

Fig. 2.14 shows a histogram for the data using singleton buckets. To further reduce the data, it is common to have each bucket denote a continuous value range for the given attribute. In Fig. 2.15, each bucket represents a different $10 range for *price*. □

"How are the buckets determined and the attribute values partitioned?" There are several partitioning rules, including the following:

- **Equal-width**: In an equal-width histogram, the width of each bucket range is uniform (e.g., the width of $10 for the buckets in Fig. 2.15).
- **Equal-frequency** (or equal-depth): In an equal-frequency histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (i.e., each bucket contains roughly the same number of contiguous data samples).

Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data. The histograms described before for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. These histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for high dimensionalities.

The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A *minimum interval size* can also be used per level to con-

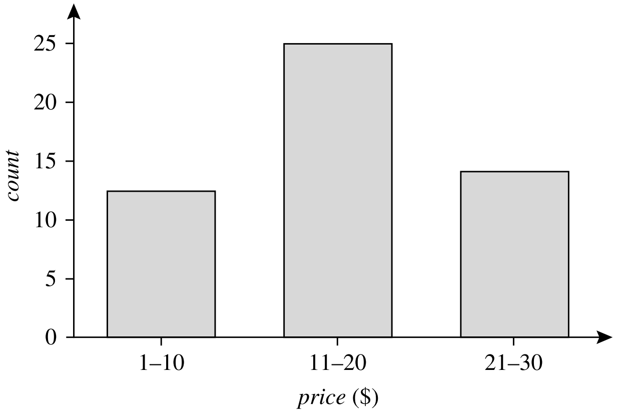


FIGURE 2.15

An equal-width histogram for *price*, where values are aggregated so that each bucket has a uniform width of $10.

trol the recursive procedure. This specifies the minimum width of a partition or the minimum number of values for each partition at each level.

2.5.3 Data compression

In data compression, transformations are applied so as to obtain a reduced or “compressed” representation of the original data. If the original data can be *reconstructed* from the compressed data without any information loss, the data reduction is called **lossless**. If, instead, we can reconstruct only an approximation of the original data, then the data reduction is called **lossy**. There are several lossless algorithms for string compression; however, they typically allow only limited data manipulation. Dimensionality reduction techniques (Section 2.6) can also be considered as forms of data compression.

The **discrete wavelet transform (DWT)** is a linear signal processing technique that, when applied to a data vector $\boldsymbol{x}$, transforms it to a numerically different vector, $\boldsymbol{x}'$, of **wavelet coefficients**. The two vectors are of the same length. When applying this technique to data reduction, we consider each tuple as an n-dimensional data vector, that is, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n database attributes.

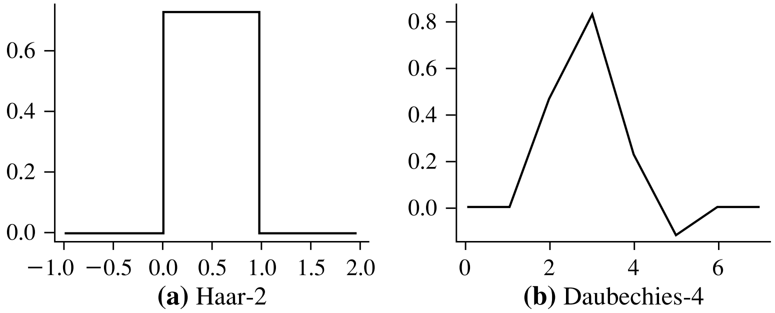
“How can this technique be useful for data reduction if the wavelet transformed data are of the same length as the original data?” The usefulness lies in the fact that the wavelet transformed data can be truncated. A compressed approximation of the data can be retained by storing only a small fraction of the strongest wavelet coefficients. For example, all wavelet coefficients larger than some user-specified threshold can be retained. All other coefficients are set to 0. The resulting data representation is therefore very sparse, so that operations that can take advantage of data sparsity are computationally very fast if performed in wavelet space. The technique also works to remove noise without smoothing out the main features of the data, making it effective for data cleaning as well. Given a set of coefficients, an approximation of the original data can be constructed by applying the *inverse* of the DWT used.

The DWT is closely related to the *discrete Fourier transform (DFT)*, a signal processing technique involving sines and cosines. In general, however, the DWT achieves better lossy compression. That is, if the same number of coefficients is retained for a DWT and a DFT of a given data vector, the DWT version will often provide a more accurate approximation of the original data. Hence, for an equivalent approximation, the DWT requires less space than the DFT. Unlike the DFT, wavelets are quite localized in space, contributing to the conservation of local detail.

There is only one DFT, yet there are several families of DWTs. Fig. 2.16 shows some wavelet families. Popular wavelet transforms include the Haar-2, Daubechies-4, and Daubechies-6. The general procedure for applying a discrete wavelet transform uses a hierarchical *pyramid algorithm* that halves the data at each iteration, resulting in fast computational speed. The method is as follows:

1. The length, L, of the input data vector must be an integer power of 2. This condition can be met by padding the data vector with zeros as necessary ($L \geq n$).
2. Each transform involves applying two functions. The first applies some data smoothing, such as sum or weighted average. The second performs a weighted difference, which acts to bring out the detailed features of the data.
3. The two functions are applied to pairs of data points in $\boldsymbol{X}$, that is, to all pairs of measurements (x_{2i}, x_{2i+1}). This results in two data sets of length $L/2$. In general, these represent a smoothed or low-frequency version of the input data and the high-frequency content of it, respectively.
4. The two functions are recursively applied to the data sets obtained in the previous iteration, until the resulting data sets obtained are of length 2.
5. Selected values from the data sets obtained in the previous iterations are designated as the wavelet coefficients of the transformed data.

Equivalently, a matrix multiplication can be applied to the input data in order to obtain the wavelet coefficients, where the matrix used depends on the given DWT. The matrix must be **orthonormal**, meaning that the columns are unit vectors and are mutually orthogonal, so that the matrix inverse is just its transpose. Although we do not have room to discuss it here, this property allows the reconstruction of the data from the smooth and smooth-difference data sets. Factoring the matrix used into a product



(a) Haar-2 **(b)** Daubechies-4

FIGURE 2.16

Examples of wavelet families. The number next to a wavelet name is the number of *vanishing moments* of the wavelet. This is a set of mathematical relationships that the coefficients must satisfy and is related to the number of coefficients.

of a few sparse matrices, the resulting "fast DWT" algorithm has a complexity of $O(n)$ for an input vector of length n.

Wavelet transforms can be applied to multidimensional data such as a data cube. This is done by first applying the transform to the first dimension, then to the second, and so on. The computational complexity involved is linear with respect to the number of cells in the cube. Wavelet transforms give good results on sparse or skewed data and on data with ordered attributes. Lossy compression by wavelets is reportedly better than JPEG compression, the current commercial standard. Wavelet transforms have many real-world applications, including the compression of fingerprint images, computer vision, analysis of time-series data, and data cleaning.

2.5.4 Sampling

Sampling can be used as a data reduction technique because it allows a large data set to be represented by a much smaller random data sample (or subset). Suppose that a large data set, D, contains N tuples. Let's look at the most common ways that we could sample D for data reduction.

- **Simple random sample without replacement (SRSWOR) of size** s: This is created by drawing s samples from D, and every time a sample is drawn, it is not to be placed back to the data set D.
- **Simple random sample with replacement (SRSWR) of size** s: This is similar to SRSWOR, except that each time a tuple is drawn from D, it is recorded and then *replaced*. That is, after a tuple is drawn, it is placed back in D so that it may be drawn again.
- **Cluster sample**: If the tuples in D are grouped into M mutually disjoint "clusters," then a sample of s clusters can be obtained, where $s < M$. For example, tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster. A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples. Other clustering criteria conveying rich semantics can also be explored. For example, in a spatial database, we may choose to define clusters geographically based on how closely different areas are located.
- **Stratified sample**: If D is divided into mutually disjoint parts called *strata,* a stratified sample of D is generated by obtaining a sample at each stratum. This helps ensure a representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where a stratum is created for each customer age group. In this way, the age group having the smallest number of customers will be sure to be represented.

An advantage of sampling for data reduction is that the cost of obtaining a sample *is proportional to the size of the sample*, s, as opposed to N, the data set size. Hence, sampling complexity is potentially *sublinear* to the size of the data. Other data reduction techniques can require at least one complete pass through D. For a fixed sample size, sampling complexity increases only linearly as the number of data dimensions, n, increases, whereas techniques using histograms, for example, could increase exponentially in n.

When applied to data reduction, sampling is most commonly used to estimate the answer to an aggregate query. It is possible (using the central limit theorem) to determine a sufficient sample size for estimating a given function within a specified degree of error. This sample size, s, may be extremely small in comparison to N. Sampling is a natural choice for the progressive refinement of a reduced data set. Such a set can be further refined by simply increasing the sample size.

2.6 Dimensionality reduction

Dimensionality reduction is the process of reducing the number of random variables or attributes or features under consideration. Dimensionality reduction methods include *principal components analysis* (PCA) (Section 2.6.1), which is a linear method that transforms or projects the original data onto a smaller space. *Attribute subset selection* is a method of dimensionality reduction in which irrelevant, weakly relevant, or redundant attributes or dimensions are detected and removed (Section 2.6.2). There are many nonlinear methods for dimensionality reduction (Section 2.6.3) such as kernel PCA and stochastic neighbor embedding.

2.6.1 Principal components analysis

In this subsection, we provide an intuitive introduction to principal components analysis as a method of dimensionality reduction. A detailed theoretical explanation is beyond the scope of this book. For additional references, please see the bibliographic notes at the end of this chapter.

Suppose that the data to be reduced consist of tuples or data vectors described by d attributes or dimensions. **Principal components analysis** (**PCA**; also called the Karhunen-Loeve, or K-L, method) searches for k d-dimensional orthonormal vectors that can best be used to represent the data, where $k \leq d$. The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. Unlike attribute subset selection (Section 2.6.2), which reduces the attribute set size by retaining a subset of the initial set of attributes, PCA "combines" the essence of attributes by creating an alternative, smaller set of variables. The initial data can then be projected onto this smaller set. PCA often reveals relationships that were not previously suspected and thereby allows interpretations that would not ordinarily result.

The basic procedure is as follows:

1. The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.
2. PCA computes k orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that are perpendicular with each other. These vectors are referred to as the *principal components*. The input data are a linear combination of the principal components.
3. The principal components are sorted in order of decreasing "significance" or strength. The principal components essentially serve as a new set of axes for the data, providing important information about variance. That is, the sorted axes are such that the first axis shows the most variance among the data, the second axis shows the next highest variance, and so on. For example, Fig. 2.17 shows the first two principal components, $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$, for the given set of data originally mapped to the axes X_1 and X_2. This information helps identify groups or patterns within the data.
4. Because the components are sorted in descending order of "significance," the data size can be reduced by eliminating the weaker components, that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data.

PCA can be applied to ordered and unordered attributes and can handle sparse data and skewed data. Multidimensional data of more than two dimensions can be handled by reducing the problem to two dimensions. Principal components may be used as inputs to multiple regression and cluster analysis.

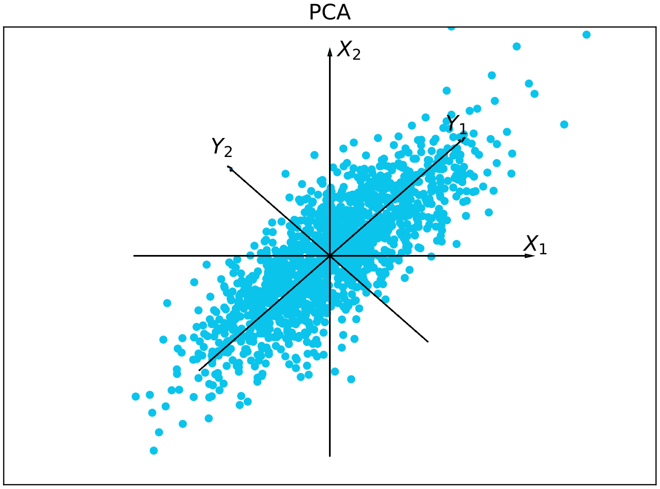


FIGURE 2.17

Principal components analysis. $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ are the first two principal components for the given data.

2.6.2 Attribute subset selection

Data sets for analysis may contain hundreds of attributes, many of which may be irrelevant to the mining task or redundant. For example, if the task is to classify customers based on whether or not they are likely to purchase a popular new music album when notified of a sale, attributes such as the customer's phone number are likely to be irrelevant, unlike attributes such as *age* or *music_taste*. Although it may be possible for a domain expert to pick out some of the useful attributes, this can be a difficult and time-consuming task, especially when the data's behavior is not well known. (Hence, a reason behind its analysis!) Leaving out relevant attributes or keeping irrelevant attributes may be detrimental, causing confusion for the mining algorithm employed. This can result in discovered patterns of poor quality. In addition, the added volume of irrelevant or redundant attributes can slow down the mining process.

Attribute subset selection[2] reduces the data set size by removing irrelevant or redundant attributes (or dimensions). This makes mining focused on the relevant dimensions. Mining on a reduced set of attributes has an additional benefit: It reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand.

"How can we find a 'good' subset of the original attributes?" For d attributes, there are 2^d possible subsets. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as d and the number of data classes increase. Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection. These methods are typically **greedy** in that, while searching through attribute space, they always make what looks to be the best choice at the time. Their strategy is to make a locally optimal choice in the hope that this will lead to a globally good

[2] In machine learning, attribute subset selection is known as *feature subset selection*.

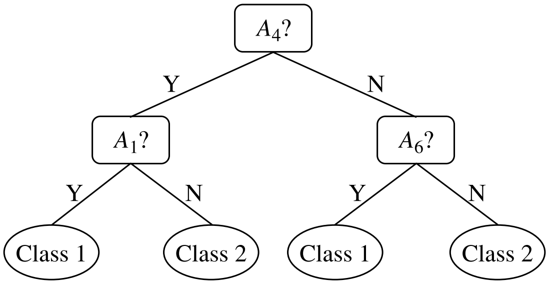
Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: $\{\}$ => $\{A_1\}$ => $\{A_1, A_4\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  A_4? Y N A_1? A_6? Y N Y N Class 1 Class 2 Class 1 Class 2 => Reduced attribute set: $\{A_1, A_4, A_6\}$

FIGURE 2.18

Greedy (heuristic) methods for attribute subset selection.

solution. Such greedy methods are effective in practice and may come close to estimating an optimal solution.

The "best" (and "worst") attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another. Many other attribute evaluation measures can be used such as the *information gain* measure used in building decision trees for classification.[3]

Basic heuristic methods of attribute subset selection include the following techniques, some of which are illustrated in Fig. 2.18.

1. **Stepwise forward selection**: The procedure starts with an empty set of attributes as the reduced set. The best of the original attributes is determined and added to the reduced set. At each subsequent iteration or step, the best of the remaining original attributes is added to the set.
2. **Stepwise backward elimination**: The procedure starts with the full set of attributes. At each step, it removes the worst attribute remaining in the set.
3. **Combination of forward selection and backward elimination**: The stepwise forward selection and backward elimination methods can be combined so that, at each step, the procedure selects the best attribute and removes the worst from among the remaining attributes.
4. **Decision tree induction**: Decision tree algorithms (e.g., ID3, C4.5, and CART) were originally intended for classification. Decision tree induction constructs a flowchart-like structure where each

[3] The information gain measure is described in detail in Chapter 6.

internal (nonleaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf) node denotes a class prediction. At each node, the algorithm chooses the "best" attribute to partition the data into individual classes.

When decision tree induction is used for attribute subset selection, a tree is constructed from the given data. All attributes that do not appear in the tree are assumed to be irrelevant. The set of attributes appearing in the tree form the reduced subset of attributes.

The stopping criteria for the methods may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

In some cases, we may want to create new attributes based on others. Such **attribute construction**[4] can help improve accuracy and understanding of structure in high-dimensional data. For example, we may wish to add the attribute *area* based on the attributes *height* and *width*. By combining attributes, attribute construction can discover missing information about the relationships between data attributes that can be useful for knowledge discovery.

2.6.3 Nonlinear dimensionality reduction methods

PCA is a linear method for dimensionality reduction in that each principal component is a linear combination of the original input attributes. This works well if the input data approximately follows a Gaussian distribution or forms a few linearly separable clusters. When the input data are linearly inseparable, however, PCA becomes ineffective. Luckily, there are many nonlinear methods we can resort to in this case.

General procedure

Suppose there are n data tuples $\boldsymbol{x}_i,\ (i = 1, ..., n)$, each of which is represented by a d-dimensional attribute vector. How can we reduce the dimensionality to k where $k \ll d$? In order words, we want to represent each of input data tuples by a k-dimensional attribute vector $\hat{\boldsymbol{x}}_i,\ (i = 1, ..., n)$. Since $k \ll d$, we call the k-dimensional attribute vector $\hat{\boldsymbol{x}}_i,\ (i = 1, ..., n)$ as low-dimensional representations of the original data tuples $\boldsymbol{x}_i,\ (i = 1, ..., n)$.

For many nonlinear dimensionality reduction methods, they often follow the following two steps (see Fig. 2.19 for an illustration). In the first step (*constructing proximity matrix*), we construct an $n \times n$ proximity matrix P whose entry $P(i, j)$ $(i, j = 1, ..., n)$ indicates the affinity or relevance between the two corresponding data tuples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. In the second step (*preserving proximity*), we learn the new, low-dimensional representations of the input data tuples in the k-dimensional space $\hat{\boldsymbol{x}}_i$ $(i = 1, ..., n)$ so that the proximity matrix P constructed in the first step is somewhat preserved.

Depending on how the proximity matrix is constructed (Step 1) and how to preserve the constructed proximity matrix (Step 2), a variety of nonlinear dimensionality reduction techniques have been developed. Let's look at two representative techniques below, including kernel PCA (KPCA) and stochastic hood embedding (SNE). A comparison of these two methods is summarized in Table 2.8.

[4] In the machine learning literature, attribute construction is known as *feature construction*.

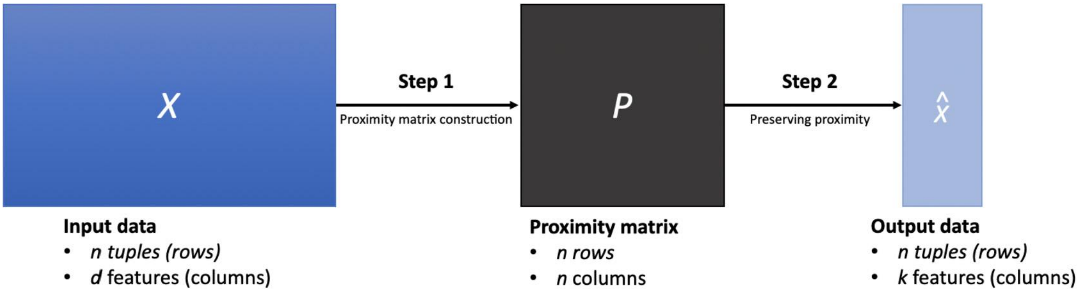


FIGURE 2.19

An illustration of nonlinear dimensionality reduction.

Table 2.8 Comparison of KPCA and SNE.

	Step 1: Proximity Construction	Step 2: Preserving Proximity
KPCA	$P(i,j) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$	$\min \sum_{i,j=1}^{n} (P(i,j) - \hat{P}(i,j))^2 = \|P - \hat{P}\|_{fro}^2$
SNE	$P(i,j) = \frac{e^{-d_{ij}^2}}{\sum_{l=1, l \neq i}^{n} e^{-d_{il}^2}}$	$\min \sum_{i=1}^{n} \mathrm{KL}(P_i \| \hat{P}_i)$

Kernel PCA

In kernel PCA (KPCA), we use a *kernel function* $\kappa(\cdot)$ to construct the proximity matrix called kernel matrix (Step 1): $P(i,j) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $(i, j = 1, ..., n)$. We will save the full details of kernel function $\kappa(\cdot)$ to the later chapters (e.g., Chapter 7). In the simplest term, a kernel function computes the similarity of a pair of input data tuples in some high-dimensional, often nonlinear, space.

Meanwhile, we can also estimate such proximity (i.e., similarity) based on the learned low-dimensional representations: $\hat{P}(i,j) = \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{x}}_j$, $(i, j = 1, ..., n)$ where $\cdot$ is the vector inner product. What would be the best (i.e., optimal) low-dimensional representations $\hat{\boldsymbol{x}}_i$, $(i = 1, ..., n)$? Intuitively we hope that the estimated proximity matrix $\hat{P}$ is as close as possible to the kernel matrix P. This leads to the following optimization problem (Step 2), which says that the best low-dimensional representations should be those that minimize $\sum_{i,j=1}^{n} (P(i,j) - \hat{P}(i,j))^2 = \|P - \hat{P}\|_{fro}^2$, where $\|\cdot\|_{fro}$ is the matrix Frobenius norm. We will not go into the mathematical details of how to solve this optimization problem. To make the long story short, it turns out the optimal low-dimensional representations $\hat{\boldsymbol{x}}_i$, $(i = 1, \ldots, n)$ can be obtained by the top-k eigenvectors and eigenvalues of the kernel matrix P. For a review of eigenvectors and eigenvalues, see Appendix A.

Typical choices for the kernel functions include (1) polynomial kernel: $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i \cdot \boldsymbol{x}_j)^p$ where p is the parameter, and (2) radial basis function (RBF) $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}}$, where σ is the parameter. If we choose a linear kernel: $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$, KPCA degenerates to the standard PCA.

Stochastic neighbor embedding

In stochastic neighbor embedding (SNE), we first construct the proximity matrix P as follows: $P(i,j) = \frac{e^{-d_{ij}^2}}{\sum_{l=1, l \neq i}^{n} e^{-d_{il}^2}}$, where $d_{ij}^2 = \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}$ and σ is the parameter. We can view $P(i,j)$ as the probability that data tuple $\boldsymbol{x}_j$ is the neighbor of data tuple $\boldsymbol{x}_i$: the closer the two data tuples are (i.e., smaller d_{ij}), the more likely $\boldsymbol{x}_j$ is the neighbor of $\boldsymbol{x}_i$.[5]

Suppose we have learned the low-dimensional representations $\hat{\boldsymbol{x}}_i$, $(i = 1, \ldots, n)$. We can obtain another estimated proximity matrix in the similar way: $\hat{P}(i, j) = \frac{e^{-\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2}}{\sum_{l=1, l \neq i}^{n} e^{-\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_l\|^2}}$. Again, the intuition is that if two data tuples share the similar low-dimensional representations (i.e., a small $\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|$), the estimated proximity between them is large (i.e., a high $\hat{P}(i, j)$). Now, in order to figure out the best low-dimensional representations $\hat{\boldsymbol{x}}_i$, $(i = 1, \ldots, n)$, we again seek those that make the estimated proximity $\hat{P}$ be as close as possible to the proximity matrix P: $P \approx \hat{P}$.

Different from KPCA, in this case, each row of both matrices P and $\hat{P}$ sums up to 1 and all the entries are nonnegative. In other words, each row of matrices P and $\hat{P}$ is a probability distribution that tells the probability that each data tuple is the neighbor of a give data tuple. Naturally we can use KL divergence (see Section 2.3.8) to measure the difference between them, and the optimal low-dimensional representations $\hat{\boldsymbol{x}}_i$, $(i = 1, \ldots, n)$ are those that minimize the overall KL divergences between all rows of P and that of $\hat{P}$: $\hat{\boldsymbol{x}}_i = \arg\min_{\hat{\boldsymbol{x}}_i, (i=1,\ldots n)} \sum_{i=1}^{n} D_{KL}(P_i || \hat{P}_i)$, where P_i and $\hat{P}_i$ are the ith rows of P and $\hat{P}$, respectively. Again, we will not go into the teeny weeny mathematical details of how to solve this optimization problem. Many off-the-shelf optimization packages can be used, such as gradient descent method.

A variant of SNE named t-SNE (t-distributed stochastic neighbor embedding) has been widely used to project the multi-dimensional representation produced by various deep learning models (Chapter 10) to a two- or three-dimensional space for the purpose of visualization.

Note that in the above introduction, we have omitted some implementation details of KPCA and SNE. For example, we need to ensure the data tuples are *centered* in KPCA; we often set $P(i, i) = 0$ in SNE; and a variant of SNE constructs a symmetric proximity matrix P. Interested readers can refer to the related papers in the bibliographic notes.

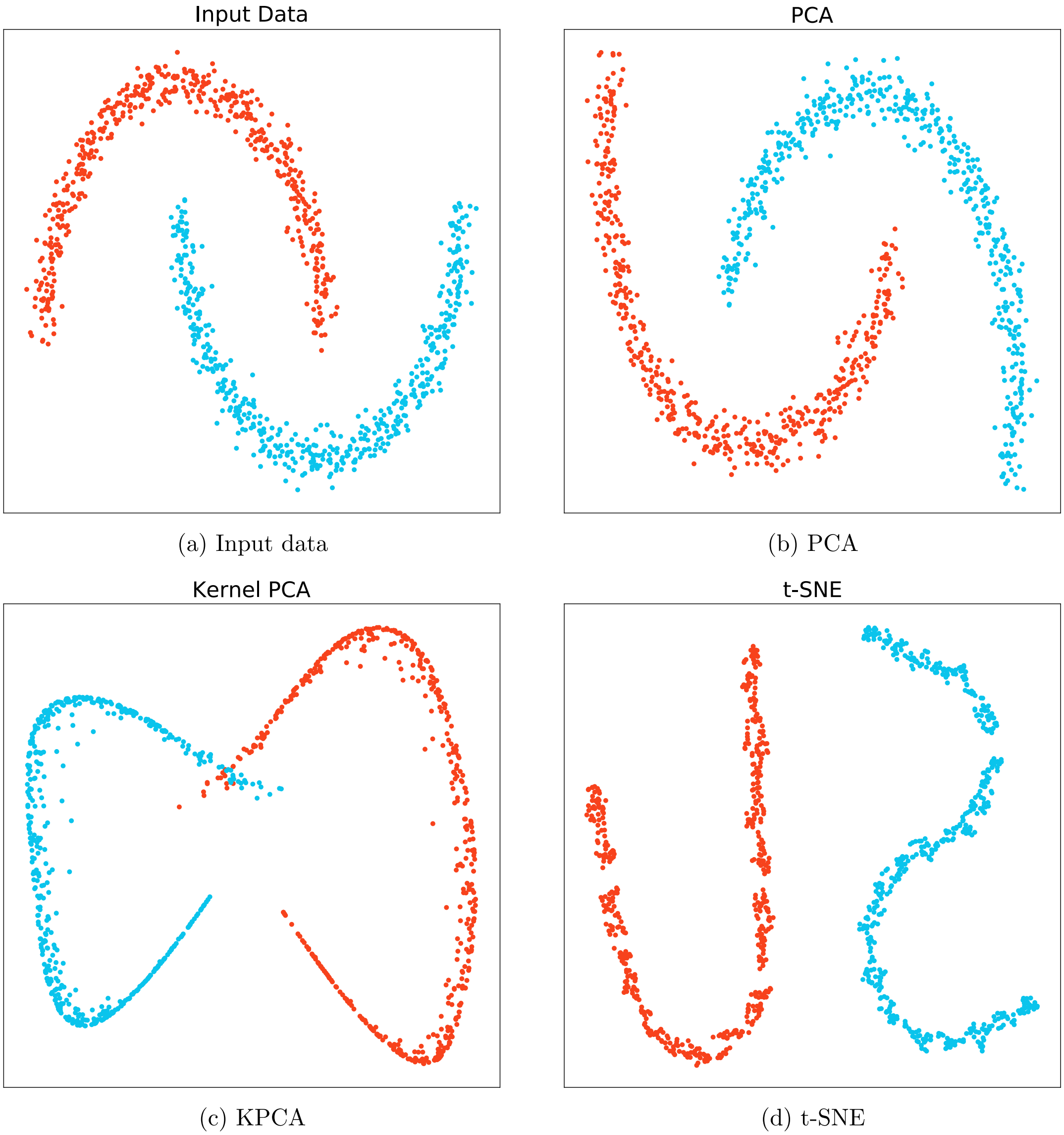
Let us look at an example.

Example 2.30. Given a collection of data tuples in 2-D space (Fig. 2.20(a)). The input data naturally form two clusters: one crescent shape facing up and one facing down. These two clusters are entangled with each other, and there is no way we can find a linear subspace (a linear line in this case) to separate them from each other. This means that no matter what kind of line we choose from the input space, if we project the original data tuples onto this line, the projected portions (i.e., the low-dimensional representation) will always be mixed with each other. This is what happens with PCA in Fig. 2.20(b), where we plot the projection of the input data onto the space spanned by two principal components. We can see that the two clusters are still mixed with each other, and the new representations by the principal components are essentially a linear rotation of the input data.

In contrast, using a nonlinear dimensionality reduction technique KPCA (Fig. 2.20(c)) or t-SNE (Fig. 2.20(d)), the two clusters are now better separated from each other in this new space.

Fig. 2.21 further shows the heatmaps of the similarity or proximity matrices in PCA (a), KPCA (b), and t-SNE (c), respectively. The two diagonal blocks indicate the proximity within the two clusters, respectively, and the two off-diagonal blocks indicate the proximity between the data from the two

[5] An interesting observation is that the proximity matrix P here is the row-normalized kernel matrix in KPCA with the RBF kernel.



(a) Input data

(b) PCA

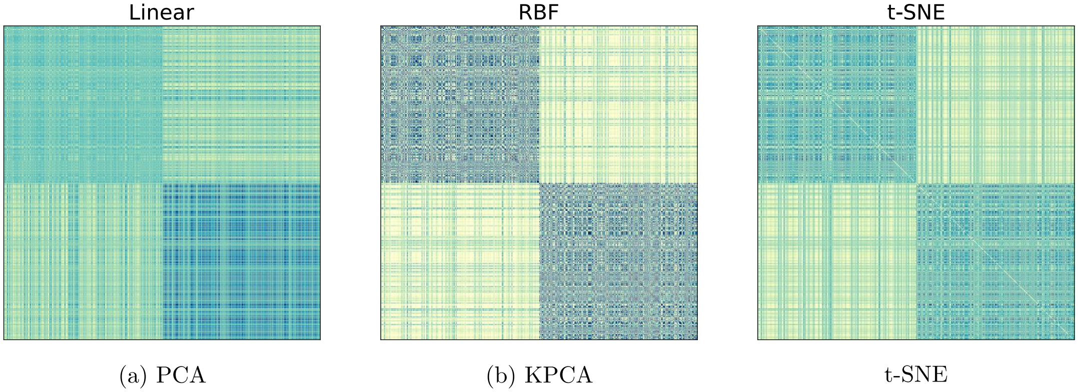
(c) KPCA

(d) t-SNE

FIGURE 2.20

An example of linear vs. nonlinear dimensionality reduction methods.

clusters. We can see that, in general, by nonlinear methods (KPCA and t-SNE), the proximity between data tuples from the same cluster is much higher than the proximity between data tuples from different clusters. This in turn leads to better dimensionality reduction results than linear methods (e.g., PCA). □



(a) PCA (b) KPCA t-SNE

FIGURE 2.21

The heatmaps of the similarity or proximity matrices in PCA (a), KPCA (b), and t-SNE (c), respectively. The two diagonal blocks correspond to the two clusters in Fig. 2.20.

We can view PCA as the following process. First, we find principal components and *project* the original data tuples into the subspace spanned by the principal components. Then, we use the projected data tuples together with the principal components to *reconstruct* the original data tuples. This is a linear process in that both the projection step and the reconstruction step are linear operations. Using a specific deep learning technique called *autoencoder*, which will be introduced in Chapter 10, we can make both projection and reconstruction steps to be nonlinear. The output from such a nonlinear projection step thus forms the low-dimensional representations of the input data tuples.

PCA, attribute subset selection, KPCA, and SNE can be used as a data preprocessing step. That is, we first apply one of these techniques on the input data tuples to produce their low-dimensional representations *before* seeing the specific data mining task (e.g., classification, clustering, and outlier detection). We can also perform dimensionality reduction *together with* a specific data mining task. The rationality is that the dimensionality reduction and the corresponding data mining task are likely to mutually complement with each other. For example, when combining attribute subset selection with the classification task (called embedded feature selection), the classification model will guide the attribute selection process, and the selected features will in turn help build a better classification model; when combining dimensionality reduction with the clustering task, the clustering structure is likely to be more evident in the new, low-dimensional space, and meanwhile such a clustering structure will help find better low-dimensional representations. We will introduce such dimensionality reduction techniques in the chapter on classification.

Dimensionality reduction, we introduced in this section, and data compression and sampling methods introduced in the previous section are common data reduction techniques. Another type of data reduction technique is called **numerosity reduction**, which uses parametric or nonparametric models to obtain smaller representations of the original data. Parametric models store only the model parameters instead of the actual data. Examples include regression and log-linear models. Nonparametric methods include histograms, clustering, sampling, and data cube aggregation.

2.7 Summary

- Data sets are made up of data objects. A **data object** represents an entity. Data objects are described by attributes. Attributes can be nominal, binary, ordinal, or numeric.
- The values of a **nominal** (or **categorical**) **attribute** are symbols or names of things, where each value represents some kind of category, code, or state.
- **Binary attributes** are nominal attributes with only two possible states (such as 1 and 0 or true and false). If the two states are equally important, the attribute is *symmetric*; otherwise it is *asymmetric*.
- An **ordinal attribute** is an attribute with possible values that have a meaningful order or ranking among them, but the magnitude between successive values is not known.
- A **numeric attribute** is *quantitative* (i.e., it is a measurable quantity) represented in integer or real values. Numeric attribute types can be *interval-scaled* or *ratio-scaled*. The values of an **interval-scaled attribute** are measured in fixed and equal units. **Ratio-scaled attributes** are numeric attributes with an inherent zero-point. Measurements are ratio-scaled in that we can speak of values as being an order of magnitude larger than the unit of measurement.
- **Basic statistical descriptions** provide the analytical foundation for data preprocessing. The basic statistical measures for data summarization include *mean, weighted mean, median*, and *mode* for measuring the central tendency of data; and *range, quantiles, quartiles, interquartile range, variance*, and *standard deviation* for measuring the dispersion of data. Graphical representations (e.g., *boxplots, quantile plots, quantile-quantile plots, histograms*, and *scatter plots*) facilitate visual inspection of the data and are thus useful for data preprocessing and mining.
- **Measures** of object **similarity** and **dissimilarity** are used in data mining applications such as clustering, outlier analysis, and nearest-neighbor classification. Such measures of *proximity* can be computed for each attribute type studied in this chapter, or for combinations of such attributes. Examples include the *Jaccard coefficient* for asymmetric binary attributes and *Euclidean*, *Manhattan*, *Minkowski*, and *supremum* distances for numeric attributes. For applications involving sparse numeric data vectors, such as term-frequency vectors, the *cosine measure* and the *Tanimoto coefficient* are often used in the assessment of similarity. To measure the difference between two probability distributions over the same variable x, *Kullback-Leibler divergence* (or the *KL divergence*) has been popularly used. $D_{KL}(p(x)||q(x))$ measures the expected number of extra bits required to code samples from $p(x)$ when using a code based on $q(x)$ rather than using a code based on $p(x)$.
- **Data quality** is defined in terms of *accuracy, completeness, consistency, timeliness, believability*, and *interpretability*. These qualities are assessed based on the intended use of the data.
- **Data cleaning** routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. Data cleaning is usually performed as an iterative two-step process consisting of discrepancy detection and data transformation.
- **Data integration** combines data from multiple sources to form a coherent data store. The resolution of semantic heterogeneity, metadata, correlation analysis, tuple duplication detection, and data conflict detection contribute to smooth data integration.
- **Data transformation** routines convert the data into appropriate forms for mining. For example, in **normalization**, attribute values are scaled; **data discretization** transforms numeric data by mapping values to interval or concept labels; and **data compression** and **data sampling**, as two typical data reduction techniques, transform the input data to a reduced representation.

- **Dimensionality reduction** reduces the number of random variables or attributes under consideration. Methods include *principal components analysis, attribute subset selection, kernel principal component analysis*, and *stochastic neighbor embedding*.

2.8 Exercises

2.1. Give three additional commonly used statistical measures that are not already illustrated in this chapter for the characterization of *data dispersion*, discuss how they can be computed efficiently in large databases.

2.2. Suppose that the data for analysis include the attribute *age*. The *age* values for the data tuples are (in ascending order) 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

a. What is the *mean* of the data? What is the *median*?

b. What is the *mode* of the data? Comment on the data's modality (i.e., bimodal, trimodal, etc.).

c. What is the *midrange* of the data?

d. Can you find (roughly) the first quartile (Q_1) and the third quartile (Q_3) of the data?

e. Give the *five-number summary* of the data.

f. Show a *boxplot* of the data.

2.3. Suppose that the values for a given set of data are grouped into intervals. The intervals and corresponding frequencies are as follows:

Age	Frequency
1–5	200
6–15	450
16–20	300
21–50	1500
51–80	700
81–110	44

Compute an *approximate median* value for the data.

2.4. How is a *quantile-quantile plot* different from a *quantile plot*?

2.5. In our text, we state that the **variance** of N observations, $x_1, x_2, \ldots, x_N$ (when N is large), for a numeric attribute X is defined as

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N}x_i^2\right) - \bar{x}^2, \qquad (2.37)$$

where $\bar{x}$ is the mean value of the observations, as defined in Eq. (2.1). This is actually the formula for calculating the variance for the whole population using all the data (hence called the *population variance*). If we are calculation the variance using only a sample of data (hence called *sample variance*), we will need to use the following formula:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2 - n\bar{x}^2\right), \qquad (2.38)$$

where n is size of the sample. With the sample size n, *sample standard deviation* can be defined similarly. Explain why there is such a minor difference at defining sample variance and population variance.

2.6. Reason why variance and standard deviation can be computed efficiently in very large data sets.

2.7. Suppose that a hospital tested the age and body fat data for 18 randomly selected adults with the following results:

age	23	23	27	27	39	41	47	49	50
%fat	9.5	26.5	7.8	17.8	31.4	25.9	27.4	27.2	31.2
age	52	54	54	56	57	58	58	60	61
%fat	34.6	42.5	28.8	33.4	30.2	34.1	32.9	41.2	35.7

a. Calculate the mean, median, and standard deviation of *age* and *%fat*

b. Draw the boxplots for *age* and *%fat*

c. Draw a *scatter plot* and a *q-q plot* based on these two variables

2.8. Briefly outline how to compute the dissimilarity between objects described by the following:

a. Nominal attributes

b. Asymmetric binary attributes

c. Numeric attributes

d. Term-frequency vectors

2.9. Given two objects represented by the tuples (22, 1, 42, 10) and (20, 0, 36, 8):

a. Compute the *Euclidean distance* between the two objects

b. Compute the *Manhattan distance* between the two objects

c. Compute the *Minkowski distance* between the two objects, using $h = 3$

d. Compute the *supremum distance* between the two objects

2.10. The *median* is one of the most important measures in data analysis. Propose several methods for median approximation. Analyze their respective complexity under different parameter settings and decide to what extent the real value can be approximated. Moreover, suggest a heuristic strategy to balance between accuracy and complexity, and then apply it to all methods you have given.

2.11. It is important to define or select similarity measures in data analysis. However, there is no commonly accepted subjective similarity measure. Results can vary depending on the similarity measures used. Nonetheless, seemingly different similarity measures may be equivalent after some transformation.

Suppose we have the following 2-D data set:

	A_1	A_2
x_1	1.5	1.7
x_2	2	1.9
x_3	1.6	1.8
x_4	1.2	1.5
x_5	1.5	1.0

a. Consider the data as 2-D data points. Given a new data point, $\boldsymbol{x} = (1.4, 1.6)$ as a query, rank the database points based on similarity with the query using Euclidean distance, Manhattan distance, supremum distance, and cosine similarity.

b. Normalize the data set to make the norm of each data point equal to 1. Use Euclidean distance on the transformed data to rank the data points.

2.12. *Data quality* can be assessed in terms of several issues, including accuracy, completeness, and consistency. For each of the above three issues, discuss how data quality assessment can depend on the *intended use* of the data, giving examples. Propose two other dimensions of data quality.

2.13. In real-world data, tuples with *missing values* for some attributes are a common occurrence. Describe various methods for handling this problem.

2.14. Given the following data (in the ascending order) for the attribute *age*: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

a. Use *smoothing by bin means* to smooth these data, using equal-frequency bins of size 3. Illustrate your steps. Comment on the effect of this technique for the given data.

b. How might you determine *outliers* in the data?

c. What other methods are there for *data smoothing*?

2.15. Discuss issues to consider during *data integration*.

2.16. What are the value ranges of the following *normalization methods*?

a. min-max normalization

b. z-score normalization

c. z-score normalization using the mean absolute deviation instead of standard deviation

d. normalization by decimal scaling

2.17. Use these methods to *normalize* the following group of data:

$$200, 300, 400, 600, 1000$$

a. min-max normalization by setting $new\_min = 0$ and $new\_max = 1$

b. z-score normalization

c. z-score normalization using the mean absolute deviation instead of standard deviation

d. normalization by decimal scaling

2.18. Using the data for *age* given in Exercise 2.14, answer the following:

a. Use min-max normalization to transform the value 35 for *age* onto the range [0.0, 1.0]

b. Use z-score normalization to transform the value 35 for *age*, where the standard deviation of *age* is 12.70 years

c. Use normalization by decimal scaling to transform the value 35 for *age*

d. Comment on which method you would prefer to use for the given data, giving reasons as to why

2.19. Using the data for *age* and *body fat* given in Exercise 2.7, answer the following:

a. Normalize the two attributes based on *z-score normalization*

b. Calculate the *correlation coefficient* (Pearson's product moment coefficient). Are these two attributes positively or negatively correlated? Compute their covariance.

2.20. Suppose a group of 12 *sales price* records has been sorted as follows:

$$5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215.$$

Partition them into three bins by each of the following methods:

a. Equal-frequency (equal-depth) partitioning

b. Equal-width partitioning

c. Clustering

2.21. Use a flowchart to summarize the following procedures for *attribute subset selection*:

a. Stepwise forward selection

b. Stepwise backward elimination

c. A combination of forward selection and backward elimination

2.22. Using the data for *age* given in Exercise 2.14,

a. Plot an equal-width histogram of width 10

b. Sketch examples of each of the following sampling techniques: SRSWOR, SRSWR, cluster sampling, and stratified sampling, using samples of size 5 and the strata "youth," "middle-aged," and "senior"

2.23. Robust data loading poses a challenge in database systems because the input data are often dirty. In many cases, an input record may miss multiple values; some records could be *contaminated*, with some data values out of range or of a different data type than expected. Work out an automated *data cleaning and loading* algorithm so that the erroneous data will be marked and contaminated data will not be mistakenly inserted into the database during data loading.

2.9 Bibliographic notes

Data description, statistical data measurements, and descriptive data characterization have been introduced in most statistics introductory textbooks. For statistics-based visualization of data using boxplots, quantile plots, quantile-quantile plots, scatter plots, and loess curves, see Cleveland [Cle93].

Similarity and distance measures among various variables have been introduced in many textbooks that study cluster analysis, including Hartigan [Har75]; Jain and Dubes [JD88]; Kaufman and Rousseeuw [KR90]; Arabie, Hubert, and de Soete [AHS96]. Methods for combining attributes of different types into a single dissimilarity matrix were introduced by Kaufman and Rousseeuw [KR90].

Data preprocessing is discussed in a number of textbooks, including Pyle [Pyl99], Loshin [Los01], Redman [Red01], and Dasu and Johnson [DJ03], and García, Luengo, and Herrera [GLH15], and Luengo et al. [LGGRG$^+$20].

For discussion regarding data quality, see Redman [Red01]; Wang, Storey, and Firth [WSF95]; Wand and Wang [WW96]; Ballou and Tayi [BT99]; and Olson [Ols03]. Potter's Wheel, an interactive data cleaning tool described in Section 2.4.2, is presented by Raman and Hellerstein [RH01]. An example of the development of declarative languages for the specification of data transformation operators is given by Galhardas et al. [GFS$^+$01]. The handling of missing attribute values is discussed by Friedman [Fri77]; Breiman, Friedman, Olshen, and Stone [BFOS84]; and Quinlan [Qui89]. Hua and Pei [HP07] present a heuristic approach to cleaning *disguised missing data*, where such data are captured when users falsely select default values on forms (e.g., "January 1" for *birthdate*) when they do not want to disclose personal information.

A method for the detection of outlier or "garbage" patterns in a handwritten character database is given in Guyon, Matic, and Vapnik [GMV96]. Binning and data normalization are treated in many texts, including Kennedy et al. [KLV$^+$98], Weiss and Indurkhya [WI98], and Pyle [Pyl99]. Systems that include attribute (or feature) construction include BACON by Langley, Simon, Bradshaw, and Zytkow [LSBZ87]; Stagger by Schlimmer [Sch86]; FRINGE by Pagallo [Pag89]; and AQ17-DCI by Bloedorn and Michalski [BM98a]. Attribute construction is also described in Liu and Motoda [LM98]. Dasu et al.

build a BELLMAN system and propose a set of interesting methods for building a data quality browser by mining database structures [DJMS02].

A survey of data reduction techniques can be found in Barbará et al. [BDF^{+}97]. For algorithms on data cubes and their precomputation, see Sarawagi and Stonebraker [SS94]; Agarwal et al. [AAD^{+}96]; Harinarayan, Rajaraman, and Ullman [HRU96]; Ross and Srivastava [RS97]; and Zhao, Deshpande, and Naughton [ZDN97]. Attribute subset selection (or *feature subset selection*) is described in many texts such as Neter, Kutner, Nachtsheim, and Wasserman [NKNW96]; Dash and Liu [DL97]; and Liu and Motoda [LM98]. A combination of forward selection and backward elimination method is proposed by Siedlecki and Sklansky [SS88]. A wrapper approach to attribute selection is described by Kohavi and John [KJ97]. Unsupervised attribute subset selection is described by Dash, Liu, and Yao [DLY97].

For a general introduction to histograms, see Barbará et al. [BDF^{+}97] and Devore and Peck [DP97]. For extensions of single-attribute histograms to multiple attributes, see Muralikrishna and DeWitt [MD88], and Poosala and Ioannidis [PI97].

There are many methods for assessing attribute relevance. Each has its own bias. The information gain measure is biased toward attributes with many values. Many alternatives have been proposed, such as gain ratio (Quinlan [Qui93]), which considers the probability of each attribute value. Other relevance measures include the Gini index (Breiman, Friedman, Olshen, and Stone [BFOS84]), the χ^2 contingency table statistic, and the uncertainty coefficient (Johnson and Wichern [JW92]). For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [BN92]. For additional methods, see Liu and Motoda [LM98], Dash and Liu [DL97], and Almuallim and Dietterich [AD91].

Liu et al. [LHTD02] perform a comprehensive survey of data discretization methods. Entropy-based discretization with the C4.5 algorithm is described by Quinlan [Qui93]. In Catlett [Cat91], the D-2 system binarizes a numeric feature recursively. ChiMerge by Kerber [Ker92] and Chi2 by Liu and Setiono [LS95] are methods for the automatic discretization of numeric attributes that both employ the χ^2 statistic.

For a description of wavelets for dimensionality reduction, see Press, Teukolosky, Vetterling, and Flannery [PTVF07]. A general account of wavelets can be found in Hubbard [Hub96]. For a list of wavelet software packages, see Bruce, Donoho, and Gao [BDG96]. Daubechies transforms are described by Daubechies [Dau92]. Routines for PCA are included in most statistical software packages such as SAS (*http://www.sas.com*). An introduction of KPCA can be found in [MSS^{+}98] by Mika, Schölkopf, and Smola. Stochastic neighbor embedding is proposed by Hinton and Roweis [HR02]. A comparative review on dimensionality reduction is by van der Maaten et al. [vdMPvdH09].