

# 2 Data Preprocessing

**Today's real-world databases** are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size (often several gigabytes or more) and their likely origin from multiple, heterogeneous sources. Low-quality data will lead to low-quality mining results. *"How can the data be preprocessed in order to help improve the quality of the data and, consequently, of the mining results? How can the data be preprocessed so as to improve the efficiency and ease of the mining process?"*

There are a number of data preprocessing techniques. *Data cleaning* can be applied to remove noise and correct inconsistencies in the data. *Data integration* merges data from multiple sources into a coherent data store, such as a data warehouse. *Data transformations*, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements. *Data reduction* can reduce the data size by aggregating, eliminating redundant features, or clustering, for instance. These 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. Data processing techniques, when applied before mining, can substantially improve the overall quality of the patterns mined and/or the time required for the actual mining.

In this chapter, we introduce the basic concepts of data preprocessing in Section 2.1. Section 2.2 presents *descriptive data summarization*, which serves as a foundation for data preprocessing. Descriptive data summarization helps us study the general characteristics of the data and identify the presence of noise or outliers, which is useful for successful data cleaning and data integration. The methods for data preprocessing are organized into the following categories: *data cleaning* (Section 2.3), *data integration and transformation* (Section 2.4), and *data reduction* (Section 2.5). Concept hierarchies can be used in an alternative form of data reduction where we replace low-level data (such as raw values for *age*) with higher-level concepts (such as *youth*, *middle-aged*, or *senior*). This form of data reduction is the topic of Section 2.6, wherein we discuss the automatic generation of concept hierarchies from numerical data using data discretization techniques. The automatic generation of concept hierarchies from categorical data is also described.

## 2.1 Why Preprocess the Data?

Imagine that you are a manager at *Allelectronics* and have been charged with analyzing the company's data with respect to the sales at your branch. 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 to be included in your analysis, such as *item*, *price*, and *units\_sold*. Alas! You notice that several of the attributes for various tuples have no recorded value. 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), **noisy** (containing errors, or *outlier* 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!

Incomplete, noisy, and inconsistent data are commonplace properties of large real-world databases and data warehouses. 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 it was 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 history or modifications to the data may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

There are many possible reasons for noisy data (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. 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 used, or inconsistent formats for input fields, such as *date*. Duplicate tuples also require data cleaning.

**Data cleaning** routines work to “clean” the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. If users believe the data are dirty, they are unlikely to trust the results of any data mining that has been applied to it. Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output. Although most mining routines have some procedures for dealing with incomplete or noisy data, they are not always robust. Instead, they may concentrate on avoiding overfitting the data to the function being modeled. Therefore, a useful preprocessing step is to run your data through some data cleaning routines. Section 2.3 discusses methods for cleaning up your data.

Getting back to your task at *Allelectronics*, suppose that you would like to include data from multiple sources in your analysis. This would involve integrating multiple

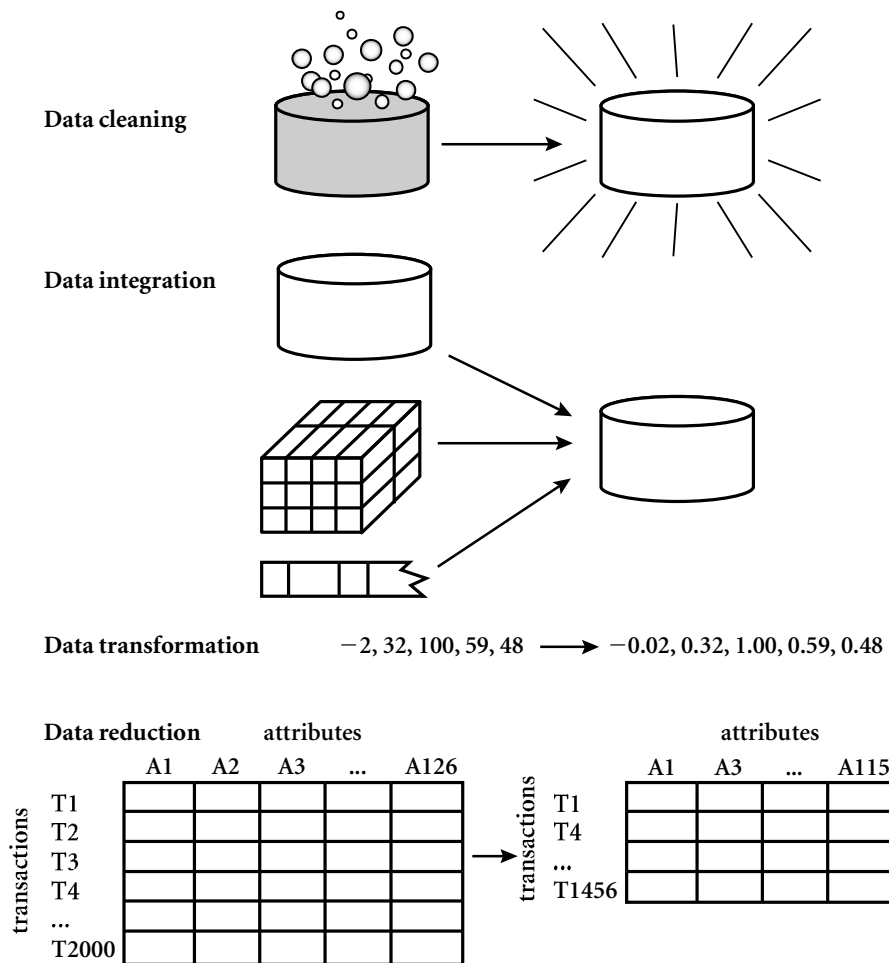
databases, data cubes, or files, that is, **data integration**. Yet some attributes representing a given concept may have different names in different databases, causing inconsistencies and redundancies. For example, the attribute for customer identification may be referred to as *customer\_id* in one data store and *cust\_id* in another. Naming inconsistencies may also occur for attribute values. For example, the same first name could be registered as “Bill” in one database, but “William” in another, and “B.” in the third. Furthermore, you suspect that some attributes may be inferred from others (e.g., annual revenue). Having a large amount of redundant data may slow down or confuse the knowledge discovery process. Clearly, in addition to data cleaning, steps must be taken to help avoid redundancies during data integration. Typically, data cleaning and data integration are performed as a preprocessing step when preparing the data for a data warehouse. Additional data cleaning can be performed to detect and remove redundancies that may have resulted from data integration.

Getting back to your data, you have decided, say, that you would like to use a distance-based mining algorithm for your analysis, such as neural networks, nearest-neighbor classifiers, or clustering.<sup>1</sup> Such methods provide better results if the data to be analyzed have been *normalized*, that is, scaled to a specific range such as [0.0, 1.0]. Your customer data, for example, contain the attributes *age* and *annual salary*. The *annual salary* attribute usually takes much larger values than *age*. Therefore, if the attributes are left unnormalized, the distance measurements taken on *annual salary* will generally outweigh distance measurements taken on *age*. Furthermore, it would be useful for your analysis to obtain aggregate information as to the sales per customer region—something that is not part of any precomputed data cube in your data warehouse. You soon realize that **data transformation** operations, such as normalization and aggregation, are additional data preprocessing procedures that would contribute toward the success of the mining process. Data integration and data transformation are discussed in Section 2.4.

“Hmmm,” you wonder, as you consider your data even further. “*The data set I have selected for analysis is HUGE, which is sure to slow down the mining process. Is there any way I can reduce the size of my data set, without jeopardizing the data mining results?*” **Data reduction** obtains a reduced representation of the data set that is much smaller in volume, yet produces the same (or almost the same) analytical results. There are a number of strategies for data reduction. These include *data aggregation* (e.g., building a data cube), *attribute subset selection* (e.g., removing irrelevant attributes through correlation analysis), *dimensionality reduction* (e.g., using encoding schemes such as minimum length encoding or wavelets), and *numerosity reduction* (e.g., “replacing” the data by alternative, smaller representations such as clusters or parametric models). Data reduction is the topic of Section 2.5. Data can also be “reduced” by *generalization* with the use of concept hierarchies, where low-level concepts, such as *city* for customer location, are replaced with higher-level concepts, such as *region* or *province\_or\_state*. A concept hierarchy organizes the concepts into varying levels of abstraction. *Data discretization* is

---

<sup>1</sup>Neural networks and nearest-neighbor classifiers are described in Chapter 6, and clustering is discussed in Chapter 7.



**Figure 2.1** Forms of data preprocessing.

a form of data reduction that is very useful for the automatic generation of concept hierarchies from numerical data. This is described in Section 2.6, along with the automatic generation of concept hierarchies for categorical data.

Figure 2.1 summarizes the data preprocessing steps described here. Note that the above categorization is not mutually exclusive. For example, the removal of redundant data may be seen as a form of data cleaning, as well as data reduction.

In summary, real-world data tend to be dirty, incomplete, and inconsistent. Data preprocessing techniques can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process. Data preprocessing is an

important step in the knowledge discovery process, because quality decisions must be based on quality data. Detecting data anomalies, rectifying them early, and reducing the data to be analyzed can lead to huge payoffs for decision making.

## 2.2 Descriptive Data Summarization

For data preprocessing to be successful, it is essential to have an overall picture of your data. Descriptive data summarization techniques can be used to identify the typical properties of your data and highlight which data values should be treated as noise or outliers. Thus, we first introduce the basic concepts of descriptive data summarization before getting into the concrete workings of data preprocessing techniques.

For many data preprocessing tasks, users would like to learn about data characteristics regarding both central tendency and dispersion of the data. Measures of central tendency include *mean*, *median*, *mode*, and *midrange*, while measures of data dispersion include *quartiles*, *interquartile range (IQR)*, and *variance*. These descriptive statistics are of great help in understanding the distribution of the data. Such measures have been studied extensively in the statistical literature. From the data mining point of view, we need to examine how they can be computed efficiently in large databases. In particular, it is necessary to introduce the notions of *distributive measure*, *algebraic measure*, and *holistic measure*. Knowing what kind of measure we are dealing with can help us choose an efficient implementation for it.

### 2.2.1 Measuring the Central Tendency

In this section, we look at various ways to measure the central tendency of data. The most common and most effective numerical measure of the “center” of a set of data is the (*arithmetic*) *mean*. Let  $x_1, x_2, \dots, x_N$  be a set of  $N$  values or observations, such as for some attribute, like *salary*. The **mean** of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N} = \frac{x_1 + x_2 + \dots + x_N}{N}. \quad (2.1)$$

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

A **distributive measure** is a measure (i.e., function) that can be computed for a given data set by partitioning the data into smaller subsets, computing the measure for each subset, and then merging the results in order to arrive at the measure’s value for the original (entire) data set. Both **sum()** and **count()** are distributive measures because they can be computed in this manner. Other examples include **max()** and **min()**. An **algebraic measure** is a measure that can be computed by applying an algebraic function to one or more distributive measures. Hence, *average* (or **mean()**) is an algebraic measure because it can be computed by **sum()/count()**. When computing

data cubes<sup>2</sup>, `sum()` and `count()` are typically saved in precomputation. Thus, the derivation of *average* for data cubes is straightforward.

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

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

This is called the **weighted arithmetic mean** or the **weighted average**. Note that the weighted average is another example of an algebraic measure.

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 average 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.

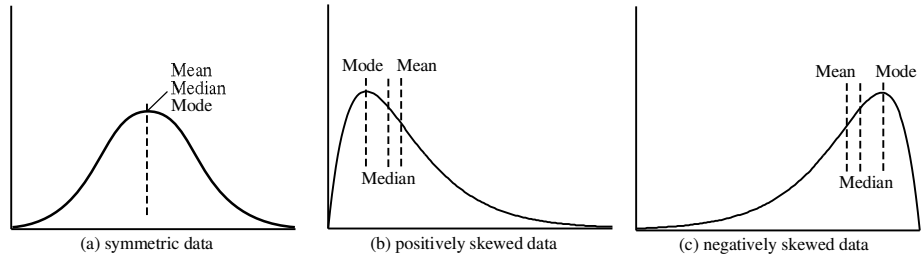
For skewed (asymmetric) data, a better measure of the center of data is the *median*. Suppose that a given data set of  $N$  distinct values is sorted in numerical order. If  $N$  is odd, then the **median** is the *middle value* of the ordered set; otherwise (i.e., if  $N$  is even), the median is the average of the middle two values.

A **holistic measure** is a measure that must be computed on the entire data set as a whole. It cannot be computed by partitioning the given data into subsets and merging the values obtained for the measure in each subset. The median is an example of a holistic measure. Holistic measures are much more expensive to compute than distributive measures such as those listed above.

We can, however, easily *approximate* the median value of a data set. 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, people may be grouped according to their annual salary in intervals such as 10–20K, 20–30K, and so on. 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:

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

<sup>2</sup>Data cube computation is described in detail in Chapters 3 and 4.



**Figure 2.2** Mean, median, and mode of symmetric versus positively and negatively skewed data.

where  $L_1$  is the lower boundary of the median interval,  $N$  is the number of values in the entire data set,  $(\sum freq)_l$  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.

Another measure of central tendency is the *mode*. The **mode** for a set of data is the value that occurs most frequently in the set. 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**. At the other extreme, if each data value occurs only once, then there is no mode.

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

$$mean - mode = 3 \times (mean - median). \quad (2.4)$$

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

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 Figure 2.2(a). However, 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 (Figure 2.2(b)), or negatively skewed, where the mode occurs at a value greater than the median (Figure 2.2(c)).

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

## 2.2.2 Measuring the Dispersion of Data

The degree to which numerical data tend to spread is called the **dispersion**, or **variance** of the data. The most common measures of data dispersion are *range*, the *five-number summary* (based on *quartiles*), the *interquartile range*, and the *standard deviation*. Boxplots

can be plotted based on the five-number summary and are a useful tool for identifying outliers.

## Range, Quartiles, Outliers, and Boxplots

Let  $x_1, x_2, \dots, x_N$  be a set of observations for some attribute. The **range** of the set is the difference between the largest ( $\max()$ ) and smallest ( $\min()$ ) values. For the remainder of this section, let's assume that the data are sorted in increasing numerical order.

The  $k$ th **percentile** of a set of data in numerical order is the value  $x_i$  having the property that  $k$  percent of the data entries lie at or below  $x_i$ . The *median* (discussed in the previous subsection) is the 50th percentile.

The most commonly used percentiles other than the median are **quartiles**. The **first quartile**, denoted by  $Q_1$ , is the 25th percentile; the **third quartile**, denoted by  $Q_3$ , is the 75th percentile. The quartiles, including the median, give some indication of the center, spread, and shape of a 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

$$IQR = Q_3 - Q_1. \quad (2.5)$$

Based on reasoning similar to that in our analysis of the median in Section 2.2.1, we can conclude that  $Q_1$  and  $Q_3$  are holistic measures, as is  $IQR$ .

No single numerical measure of spread, such as  $IQR$ , is very useful for describing skewed distributions. The spreads of two sides of a skewed distribution are unequal (Figure 2.2). 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  $1.5 \times IQR$  above the third quartile or below the first quartile.

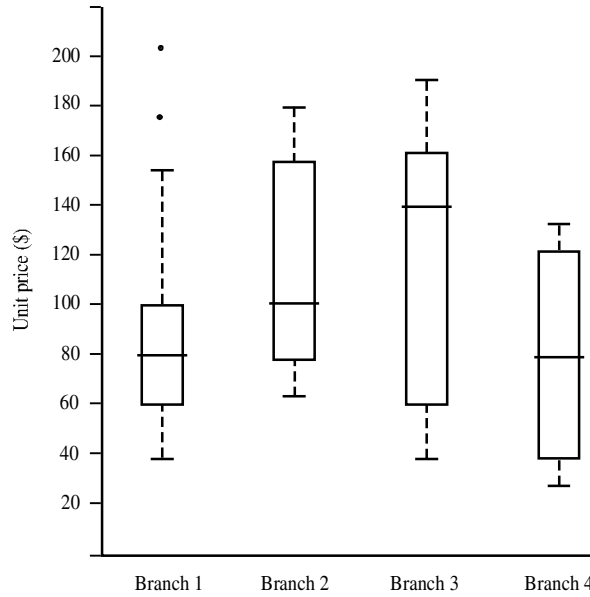
Because  $Q_1$ , the median, and  $Q_3$  together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the *five-number summary*. The **five-number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





**Figure 2.3** Boxplot for the unit price data for items sold at four branches of *AllElectronics* during a given time period.

the extreme low and high observations *only if* these values are less than  $1.5 \times IQR$  beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within  $1.5 \times IQR$  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 shows boxplots for unit price data for items sold at four branches of *AllElectronics* during a given time period. For branch 1, we see that the median price of items sold is \$80,  $Q_1$  is \$60,  $Q_3$  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. The efficient computation of boxplots, or even *approximate boxplots* (based on approximates of the five-number summary), remains a challenging issue for the mining of large data sets.

## Variance and Standard Deviation

The **variance** of  $N$  observations,  $x_1, x_2, \dots, x_N$ , is

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

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

The basic properties of the standard deviation,  $\sigma$ , as a measure of spread are

- $\sigma$  measures spread about the mean and should be used 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  $\sigma > 0$ .

The variance and standard deviation are algebraic measures because they can be computed from distributive measures. That is,  $N$  (which is `count()` in SQL),  $\sum x_i$  (which is the `sum()` of  $x_i$ ), and  $\sum x_i^2$  (which is the `sum()` of  $x_i^2$ ) can be computed in any partition and then merged to feed into the algebraic Equation (2.6). Thus the computation of the variance and standard deviation is scalable in large databases.

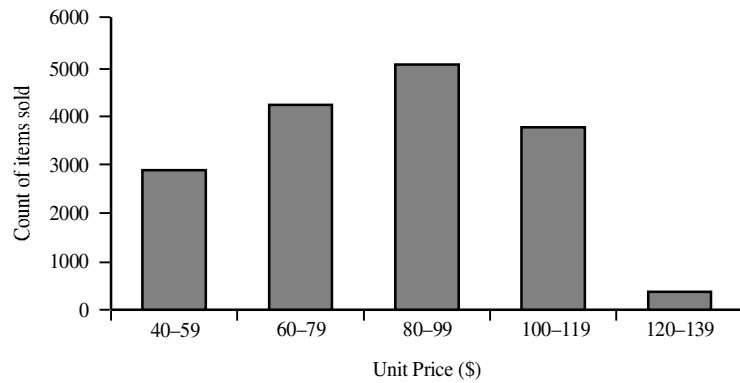
### 2.2.3 Graphic Displays of Basic Descriptive Data Summaries

Aside from the bar charts, pie charts, and line graphs used in most statistical or graphical data presentation software packages, there are other popular types of graphs for the display of data summaries and distributions. These include *histograms*, *quantile plots*, *q-q plots*, *scatter plots*, and *loess curves*. Such graphs are very helpful for the visual inspection of your data.

Plotting **histograms**, or **frequency histograms**, is a graphical method for summarizing the distribution of a given attribute. A histogram for an attribute  $A$  partitions the data distribution of  $A$  into disjoint subsets, or *buckets*. Typically, the width of each bucket is uniform. Each bucket is represented by a rectangle whose height is equal to the count or relative frequency of the values at the bucket. If  $A$  is categoric, such as *automobile\_model* or *item\_type*, then one rectangle is drawn for each known value of  $A$ , and the resulting graph is more commonly referred to as a **bar chart**. If  $A$  is numeric, the term *histogram* is preferred. Partitioning rules for constructing histograms for numerical attributes are discussed in Section 2.5.4. In an equal-width histogram, for example, each bucket represents an equal-width range of numerical attribute  $A$ .

Figure 2.4 shows a histogram for the data set of Table 2.1, where buckets are defined by equal-width ranges representing \$20 increments and the frequency is the count of items sold. Histograms are at least a century old and are a widely used univariate graphical method. However, they may not be as effective as the quantile plot, q-q plot, and boxplot methods for comparing groups of univariate observations.

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 the user to assess both the overall behavior and unusual occurrences). Second, it plots quantile information. The mechanism used in this step is slightly different from the percentile computation discussed in Section 2.2.2. Let  $x_i$ , for  $i = 1$  to  $N$ , be the data sorted in increasing order so that  $x_1$  is the smallest observation and  $x_N$  is the largest. Each observation,  $x_i$ , is paired with a percentage,  $f_i$ , which indicates that approximately  $100f_i\%$  of the data are below or equal to the value,  $x_i$ . We say “approximately” because



**Figure 2.4** A histogram for the data set of Table 2.1.

**Table 2.1** A set of unit price data for items sold at a branch of *AllElectronics*.

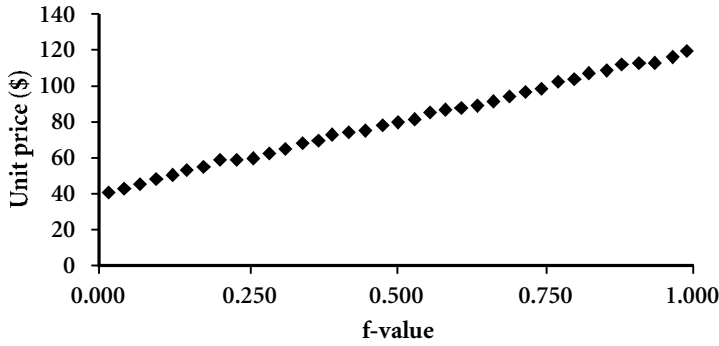
Unit price (\$)	Count of items sold
40	275
43	300
47	250
..	..
74	360
75	515
78	540
..	..
115	320
117	270
120	350

there may not be a value with exactly a fraction,  $f_i$ , of the data below or equal to  $x_i$ . 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

$$f_i = \frac{i - 0.5}{N}. \quad (2.7)$$

These numbers increase in equal steps of  $1/N$ , ranging from  $1/2N$  (which is slightly above zero) to  $1 - 1/2N$  (which is slightly below one). 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



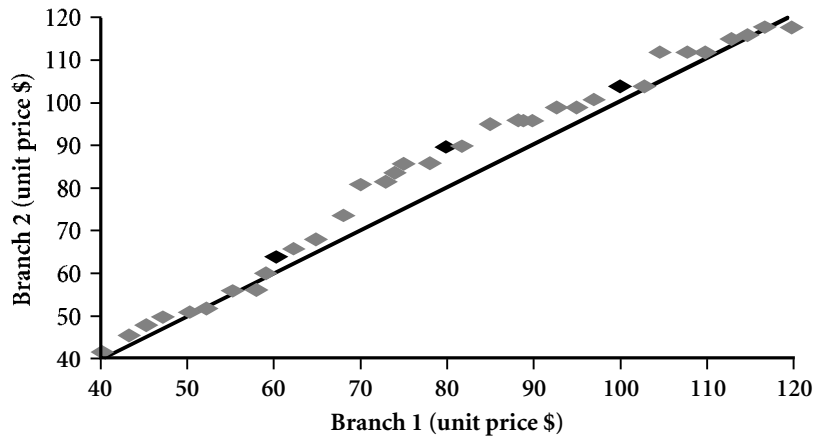
**Figure 2.5** A quantile plot for the unit price data of Table 2.1.

compare their  $Q_1$ , median,  $Q_3$ , and other  $f_i$  values at a glance. Figure 2.5 shows a quantile plot for the *unit price* data of Table 2.1.

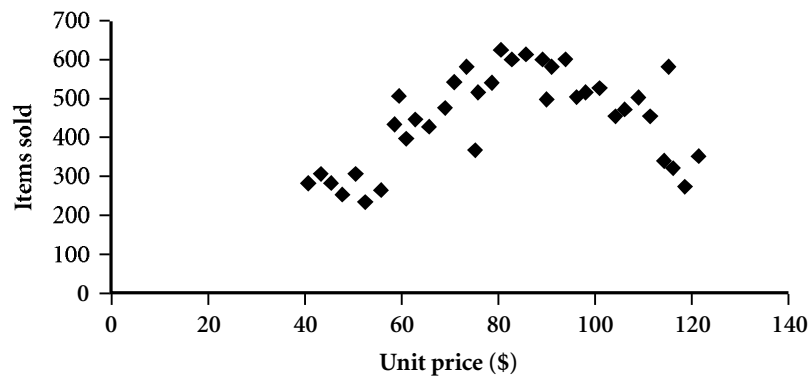
A **quantile-quantile plot**, or **q-q plot**, graphs the quantiles of one univariate distribution against the corresponding quantiles of another. It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another.

Suppose that we have two sets of observations for the variable *unit price*, taken from two different branch locations. Let  $x_1, \dots, x_N$  be the data from the first branch, and  $y_1, \dots, y_M$  be the data from the second, where each data set is sorted in increasing 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.

Figure 2.6 shows a quantile-quantile plot for *unit price* data of items sold at two different branches of *AllElectronics* 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 versus branch 2 for that quantile. For example, here the lowest point in the left corner corresponds to the 0.03 quantile. (To aid in comparison, we also show a straight line that represents the case of when, for each given quantile, the unit price at each branch is the same. In addition, the darker points correspond to the data for  $Q_1$ , the median, and  $Q_3$ , respectively.) We see that at this quantile, the unit price of items sold at branch 1 was slightly less than that at branch 2. In other words, 3% of items sold at branch 1 were less than or equal to \$40, while 3% of items at branch 2 were less than or equal to \$42. At the highest quantile, we see that the unit price of items at branch 2 was slightly less than that at branch 1. 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.



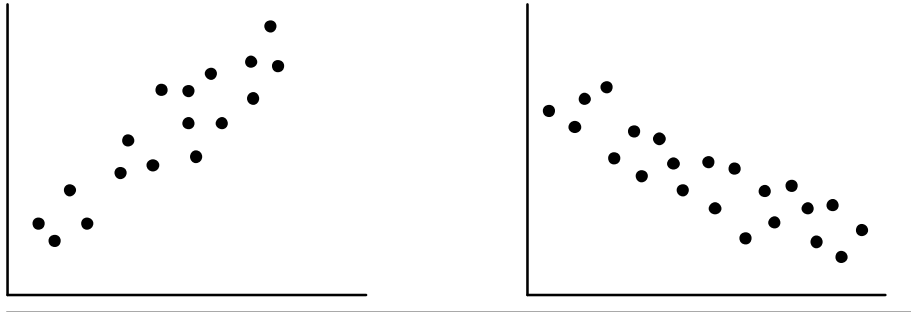
**Figure 2.6** A quantile-quantile plot for unit price data from two different branches.



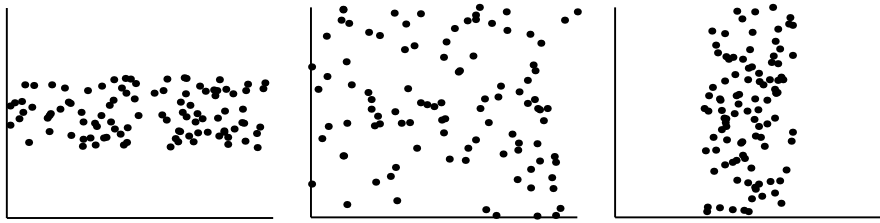
**Figure 2.7** A scatter plot for the data set of Table 2.1.

A **scatter plot** is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numerical 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. Figure 2.7 shows a scatter plot for the set of data in Table 2.1. 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.<sup>3</sup> In Figure 2.8, we see examples of positive and negative correlations between

<sup>3</sup> A statistical test for correlation is given in Section 2.4.1 on data integration (Equation (2.8)).



**Figure 2.8** Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.



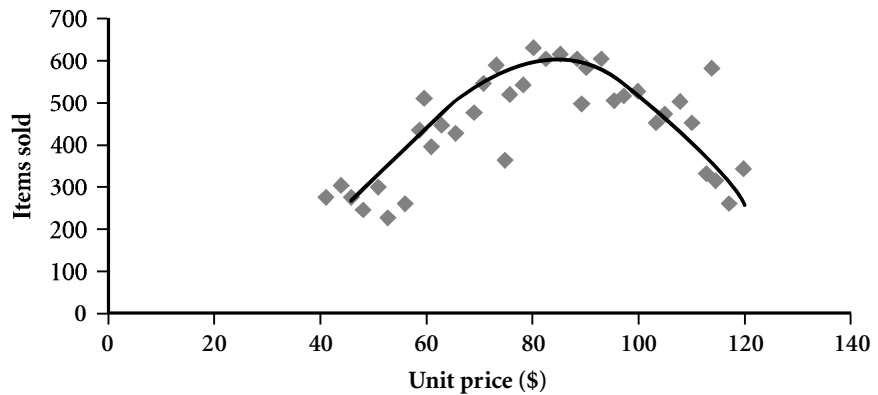
**Figure 2.9** Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.

two attributes in two different data sets. Figure 2.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets.

When dealing with several attributes, the **scatter-plot matrix** is a useful extension to the scatter plot. Given  $n$  attributes, a scatter-plot matrix is an  $n \times n$  grid of scatter plots that provides a visualization of each attribute (or dimension) with every other attribute. The scatter-plot matrix becomes less effective as the number of attributes under study grows. In this case, user interactions such as zooming and panning become necessary to help interpret the individual scatter plots effectively.

A **loess curve** is another important exploratory graphic aid that adds a smooth curve to a scatter plot in order to provide better perception of the pattern of dependence. The word *loess* is short for “local regression.” Figure 2.10 shows a loess curve for the set of data in Table 2.1.

To fit a loess curve, values need to be set for two parameters— $\alpha$ , a smoothing parameter, and  $\lambda$ , the degree of the polynomials that are fitted by the regression. While  $\alpha$  can be any positive number (typical values are between  $1/4$  and  $1$ ),  $\lambda$  can be  $1$  or  $2$ . The goal in choosing  $\alpha$  is to produce a fit that is as smooth as possible without unduly distorting the underlying pattern in the data. The curve becomes smoother as  $\alpha$  increases. There may be some lack of fit, however, indicating possible “missing” data patterns. If  $\alpha$  is very small, the underlying pattern is tracked, yet overfitting of the data may occur where local “wiggles” in the curve may not be supported by the data. If the underlying pattern of the data has a



**Figure 2.10** A loess curve for the data set of Table 2.1.

“gentle” curvature with no local maxima and minima, then local linear fitting is usually sufficient ( $\lambda = 1$ ). However, if there are local maxima or minima, then local quadratic fitting ( $\lambda = 2$ ) typically does a better job of following the pattern of the data and maintaining local smoothness.

In conclusion, descriptive data summaries 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 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. Section 2.3.1 looks at ways of handling missing values. Section 2.3.2 explains data smoothing techniques. Section 2.3.3 discusses approaches to data cleaning as a process.

### 2.3.1 Missing Values

Imagine that you need to analyze *AllElectronics* sales and customer data. 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.

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 the attribute mean to fill in the missing value:** For example, suppose that the average income of *AllElectronics* customers is \$56,000. Use this value to replace the missing value for *income*.
5. **Use the attribute mean for all samples belonging to the same class as the given tuple:** For example, if classifying customers according to *credit.risk*, replace the missing value with the average *income* value for customers in the same credit risk category as that of the given tuple.
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 Chapter 6.

Methods 3 to 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 the 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, such as “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 design of databases and of data entry procedures should help minimize the number of missing values or errors in the first place.

### 2.3.2 Noisy Data

“*What is noise?*” Noise is a random error or variance in a measured variable. Given a numerical 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:



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

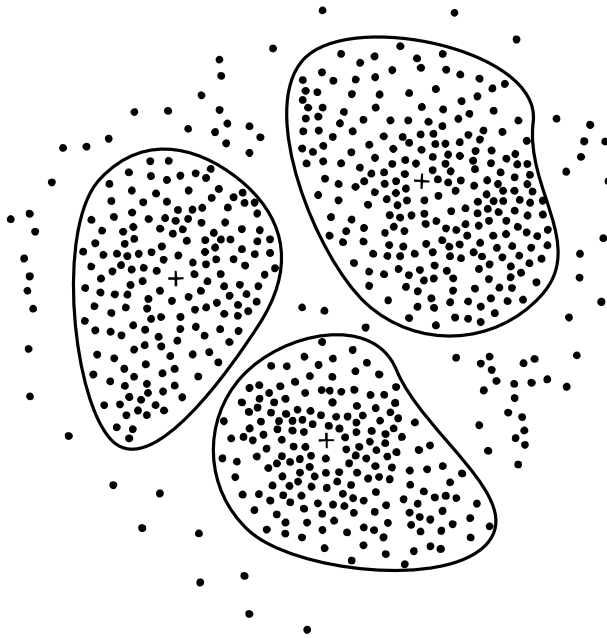
Smoothing by bin boundaries:

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

---

**Figure 2.11** Binning methods for data smoothing.

1. **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. Figure 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 *equal-width*, where the interval range of values in each bin is constant. Binning is also used as a discretization technique and is further discussed in Section 2.6.
2. **Regression:** Data can be smoothed by fitting the data to a function, such as with regression. *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 Section 2.5.4, as well as in Chapter 6.



**Figure 2.12** A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster centroid is marked with a “+”, representing the average point in space for that cluster. Outliers may be detected as values that fall outside of the sets of clusters.

3. **Clustering:** Outliers may be detected by clustering, where similar values are organized into groups, or “clusters.” Intuitively, values that fall outside of the set of clusters may be considered outliers (Figure 2.12). Chapter 7 is dedicated to the topic of clustering and outlier analysis.

Many methods for data smoothing are also methods for data reduction involving discretization. For example, the binning techniques described above 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 make 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 2.6. Some methods of classification, such as neural networks, have built-in data smoothing mechanisms. Classification is the topic of Chapter 6.

### 2.3.3 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 the inconsistent use of codes. Errors in instrumentation devices that record data, and system errors, are another source of discrepancies. 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).<sup>4</sup>

“*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**. For example, what are the domain and data type of each attribute? What are the acceptable values for each attribute? What is the range of the length of values? Do all values fall within the expected range? Are there any known dependencies between attributes? The descriptive data summaries presented in Section 2.2 are useful here for grasping data trends and identifying anomalies. For example, values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. In this step, you may write your own scripts and/or use some of the tools that we discuss further below. 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 (such as “2004/12/25” and “25/12/2004” for *date*). **Field overloading** is another source of errors that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., using an unused bit of an attribute whose value range uses only, say, 31 out of 32 bits).

The data should also be examined regarding unique rules, consecutive rules, and null rules. A **unique rule** says that each value of the given attribute must be different from all other values for that attribute. A **consecutive 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 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 in Section 2.3.1, reasons for missing values may include (1) the person originally asked to provide a value for the attribute refuses and/or finds

---

<sup>4</sup>Data integration and the removal of redundant data that can result from such integration are further described in Section 2.4.1.

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

There are a number of different commercial tools that can aid in the step of discrepancy detection. **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 descriptive data summaries that were described 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*. This is the second step in data cleaning as a process. 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 transforms 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 “20004” 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 created by mistake. 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 (see <http://control.cs.berkeley.edu/abc>) 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 introduced additional errors can be “erased.” The tool performs discrepancy checking automatically 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.

Another approach to increased 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 Data Integration and Transformation

Data mining often requires data integration—the merging of data from multiple data stores. The data may also need to be transformed into forms appropriate for mining. This section describes both data integration and data transformation.

### 2.4.1 Data Integration

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 the data analyst or the computer be sure that *customer\_id* in one database and *cust\_number* in another refer to the same attribute? Examples of metadata for each attribute include the name, meaning, data type, and range of values permitted for the attribute, and null rules for handling blank, zero, or null values (Section 2.3). Such metadata can be used to help avoid errors in schema integration. The metadata may also be used to help transform the data (e.g., where data codes for *pay\_type* in one database may be “H” and “S”, and 1 and 2 in another). Hence, this step also relates to data cleaning, as described earlier.

*Redundancy* is another important issue. 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 numerical 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 inventor, Karl Pearson). This is

$$r_{A,B} = \frac{\sum_{i=1}^N (a_i - \bar{A})(b_i - \bar{B})}{N\sigma_A\sigma_B} = \frac{\sum_{i=1}^N (a_i b_i) - N\bar{A}\bar{B}}{N\sigma_A\sigma_B}, \quad (2.8)$$

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  $B$ ,  $\sigma_A$  and  $\sigma_B$  are the respective standard deviations of  $A$  and  $B$  (as defined in Section 2.2.2), and  $\sum(a_i b_i)$  is the sum of the  $AB$  cross-product (that is, for each tuple, the value for  $A$  is multiplied by the value for  $B$  in that tuple). Note that  $-1 \leq r_{A,B} \leq +1$ . 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).

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*.

For categorical (discrete) data, a correlation relationship between two attributes,  $A$  and  $B$ , can be discovered by a  $\chi^2$  (chi-square) test. Suppose  $A$  has  $c$  distinct values, namely  $a_1, a_2, \dots, a_c$ .  $B$  has  $r$  distinct values, namely  $b_1, b_2, \dots, b_r$ . 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 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  $\chi^2$  value (also known as the *Pearson  $\chi^2$  statistic*) is computed as:

$$\chi^2 = \sum_{i=1}^c \sum_{j=1}^r \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \quad (2.9)$$

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

$$e_{ij} = \frac{\text{count}(A = a_i) \times \text{count}(B = b_j)}{N}, \quad (2.10)$$

where  $N$  is the number of data tuples,  $\text{count}(A = a_i)$  is the number of tuples having value  $a_i$  for  $A$ , and  $\text{count}(B = b_j)$  is the number of tuples having value  $b_j$  for  $B$ . The sum in Equation (2.9) is computed over all of the  $r \times c$  cells. Note that the cells that contribute the most to the  $\chi^2$  value are those whose actual count is very different from that expected.

**Table 2.2** A  $2 \times 2$  contingency table for the data of Example 2.1. Are *gender* and *preferred\_Reading* correlated?

	<i>male</i>	<i>female</i>	Total
<i>fiction</i>	250 (90)	200 (360)	450
<i>non_fiction</i>	50 (210)	1000 (840)	1050
Total	300	1200	1500

The  $\chi^2$  statistic tests the hypothesis that  $A$  and  $B$  are independent. The test is based on a significance level, with  $(r - 1) \times (c - 1)$  degrees of freedom. We will illustrate the use of this statistic in an example below. If the hypothesis can be rejected, then we say that  $A$  and  $B$  are statistically related or associated.

Let's look at a concrete example.

**Example 2.1** Correlation analysis of categorical attributes using  $\chi^2$ . Suppose that a group of 1,500 people was surveyed. The gender of each person was noted. Each person was polled as to whether their 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 (calculated based on the data distribution for both attributes using Equation (2.10)).

Using Equation (2.10), we can verify the expected frequencies for each cell. For example, the expected frequency for the cell (male, fiction) is

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

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 Equation (2.9) for  $\chi^2$  computation, we get

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

For this  $2 \times 2$  table, the degrees of freedom are  $(2 - 1)(2 - 1) = 1$ . For 1 degree of freedom, the  $\chi^2$  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  $\chi^2$  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* are independent and conclude that the two attributes are (strongly) correlated for the given group of people. ■

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 case). 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 of the occurrences of the data. 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.

A third important issue in data integration is 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 (such as free breakfast) and taxes. An attribute in one system may be recorded at a lower level of abstraction than the "same" attribute in another. For example, the *total\_sales* in one database may refer to one branch of *All\_Electronics*, while an attribute of the same name in another database may refer to the total sales for *All\_Electronics* stores in a given region.

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.

The semantic heterogeneity and structure of data pose great challenges in data integration. Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent mining process.

### 2.4.2 Data Transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Data transformation can involve the following:

- **Smoothing**, which works to remove noise from the data. Such techniques include binning, regression, and clustering.
- **Aggregation**, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for analysis of the data at multiple granularities.
- **Generalization** of the data, where low-level or "primitive" (raw) data are replaced by higher-level concepts through the use of concept hierarchies. For example, categorical



attributes, like *street*, can be generalized to higher-level concepts, like *city* or *country*. Similarly, values for numerical attributes, like *age*, may be mapped to higher-level concepts, like *youth*, *middle-aged*, and *senior*.

- **Normalization**, where the attribute data are scaled so as to fall within a small specified range, such as  $-1.0$  to  $1.0$ , or  $0.0$  to  $1.0$ .
- **Attribute construction** (or *feature construction*), where new attributes are constructed and added from the given set of attributes to help the mining process.

Smoothing is a form of data cleaning and was addressed in Section 2.3.2. Section 2.3.3 on the data cleaning process also discussed ETL tools, where users specify transformations to correct data inconsistencies. Aggregation and generalization serve as forms of data reduction and are discussed in Sections 2.5 and 2.6, respectively. In this section, we therefore discuss normalization and attribute construction.

An attribute is normalized by scaling its values so that they fall within a small specified range, such as  $0.0$  to  $1.0$ . 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 mining (Chapter 6), 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). There are many methods for data normalization. We study three: *min-max normalization*, *z-score normalization*, and *normalization by decimal scaling*.

**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,  $v$ , of  $A$  to  $v'$  in the range  $[new\_min_A, new\_max_A]$  by computing

$$v' = \frac{v - min_A}{max_A - min_A} (new\_max_A - new\_min_A) + new\_min_A. \quad (2.11)$$

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 2.2 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 and standard deviation of  $A$ . A value,  $v$ , of  $A$  is normalized to  $v'$  by computing

$$v' = \frac{v - \bar{A}}{\sigma_A}, \quad (2.12)$$

where  $\bar{A}$  and  $\sigma_A$  are the mean and standard deviation, respectively, of attribute  $A$ . 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 2.3 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  $\frac{73,600 - 54,000}{16,000} = 1.225$ . ■

**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,  $v$ , of  $A$  is normalized to  $v'$  by computing

$$v' = \frac{v}{10^j}, \quad (2.13)$$

where  $j$  is the smallest integer such that  $\text{Max}(|v'|) < 1$ .

**Example 2.4 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  $1,000$  (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 the latter two methods shown above. It is also necessary to save the normalization parameters (such as the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

In **attribute construction**,<sup>5</sup> new attributes are constructed from the given attributes and added in order to help improve the 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.

## 2.5 Data Reduction

Imagine that you have selected data from the *AllElectronics* data warehouse for analysis. The data set will likely be huge! Complex data analysis and mining on huge amounts of data can take a long time, making such analysis impractical or infeasible.

<sup>5</sup>In the machine learning literature, attribute construction is known as *feature construction*.

**Data reduction** techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results.

Strategies for data reduction include the following:

1. **Data cube aggregation**, where aggregation operations are applied to the data in the construction of a data cube.
2. **Attribute subset selection**, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.
3. **Dimensionality reduction**, where encoding mechanisms are used to reduce the data set size.
4. **Numerosity reduction**, where the data are replaced or estimated by alternative, smaller data representations such as parametric models (which need store only the model parameters instead of the actual data) or nonparametric methods such as clustering, sampling, and the use of histograms.
5. **Discretization and concept hierarchy generation**, where raw data values for attributes are replaced by ranges or higher conceptual levels. Data discretization is a form of numerosity reduction that is very useful for the automatic generation of concept hierarchies. Discretization and concept hierarchy generation are powerful tools for data mining, in that they allow the mining of data at multiple levels of abstraction. We therefore defer the discussion of discretization and concept hierarchy generation to Section 2.6, which is devoted entirely to this topic.

Strategies 1 to 4 above are discussed in the remainder of this section. The computational time spent on data reduction should not outweigh or “erase” the time saved by mining on a reduced data set size.

### 2.5.1 Data Cube Aggregation

Imagine that you have collected the data for your analysis. These data consist of the *AllElectronics* sales per quarter, for the years 2002 to 2004. You are, however, interested in the annual sales (total per year), rather than the total per quarter. Thus the data can be *aggregated* so that the resulting data summarize the total sales per year instead of per quarter. This aggregation is illustrated in Figure 2.13. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

Data cubes are discussed in detail in Chapter 3 on data warehousing. We briefly introduce some concepts here. Data cubes store multidimensional aggregated information. For example, Figure 2.14 shows a data cube for multidimensional analysis of sales data with respect to annual sales per item type for each *AllElectronics* branch. Each cell holds an aggregate data value, corresponding to the data point in multidimensional space. (For readability, only some cell values are shown.) Concept

Year 2004	
Quarter	Sales
Q1	\$224,000
Q2	\$408,000
Q3	\$350,000
Q4	\$586,000

Year 2003	
Quarter	Sales
Q1	\$224,000
Q2	\$408,000
Q3	\$350,000
Q4	\$586,000

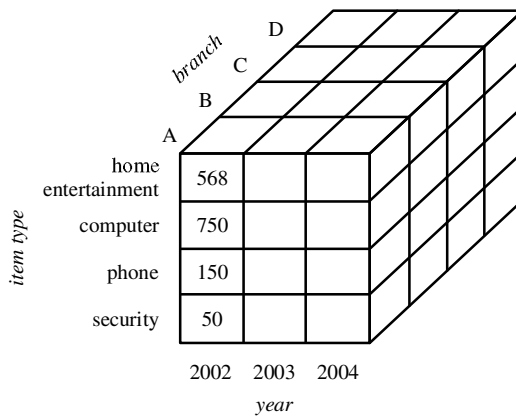
  

Year 2002	
Quarter	Sales
Q1	\$224,000
Q2	\$408,000
Q3	\$350,000
Q4	\$586,000

Year	Sales
2002	\$1,568,000
2003	\$2,356,000
2004	\$3,594,000

**Figure 2.13** Sales data for a given branch of *AllElectronics* for the years 2002 to 2004. On the left, the sales are shown per quarter. On the right, the data are aggregated to provide the annual sales.



**Figure 2.14** A data cube for sales at *AllElectronics*.

hierarchies may exist for each attribute, allowing the analysis of data at multiple levels of abstraction. For example, a hierarchy for *branch* could allow branches to be grouped into regions, based on their address. Data cubes provide fast access to precomputed, summarized data, thereby benefiting on-line analytical processing as well as data mining.

The cube created at the lowest level of abstraction is referred to as the *base cuboid*. The base cuboid should correspond to an individual entity of interest, such as *sales* or *customer*. In other words, the lowest level should be usable, or useful for the analysis. A cube at the highest level of abstraction is the *apex cuboid*. For the sales data of Figure 2.14, the apex cuboid would give one total—the total *sales*

for all three years, for all item types, and for all branches. Data cubes created for varying levels of abstraction are often referred to as *cuboids*, so that a data cube may instead refer to a *lattice of cuboids*. Each higher level of abstraction further reduces the resulting data size. When replying to data mining requests, the *smallest* available cuboid relevant to the given task should be used. This issue is also addressed in Chapter 3.

## 2.5.2 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 as to whether or not they are likely to purchase a popular new CD at *AllElectronics* when notified of a sale, attributes such as the customer's telephone 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 behavior of the data 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.

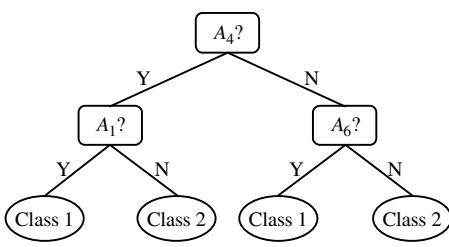
**Attribute subset selection**<sup>6</sup> reduces the data set size by removing irrelevant or redundant attributes (or dimensions). The goal of attribute subset selection is to find a minimum set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes. 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  $n$  attributes, there are  $2^n$  possible subsets. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as  $n$  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 optimal 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

---

<sup>6</sup>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: $\{\}$ $\Rightarrow \{A_1\}$ $\Rightarrow \{A_1, A_4\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_4, A_5, A_6\}$ $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$    $\Rightarrow$ Reduced attribute set: $\{A_1, A_4, A_6\}$

**Figure 2.15** Greedy (heuristic) methods for attribute subset selection.

other attribute evaluation measures can be used, such as the *information gain* measure used in building decision trees for classification.<sup>7</sup>

Basic heuristic methods of attribute subset selection include the following techniques, some of which are illustrated in Figure 2.15.

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, such as ID3, C4.5, and CART, were originally intended for classification. Decision tree induction constructs a flowchart-like structure where each 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

<sup>7</sup>The information gain measure is described in detail in Chapter 6. It is briefly described in Section 2.6.1 with respect to attribute discretization.

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.

### 2.5.3 Dimensionality Reduction

In *dimensionality reduction*, data encoding or 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 loss of information, 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 well-tuned algorithms for string compression. Although they are typically lossless, they allow only limited manipulation of the data. In this section, we instead focus on two popular and effective methods of lossy dimensionality reduction: *wavelet transforms* and *principal components analysis*.

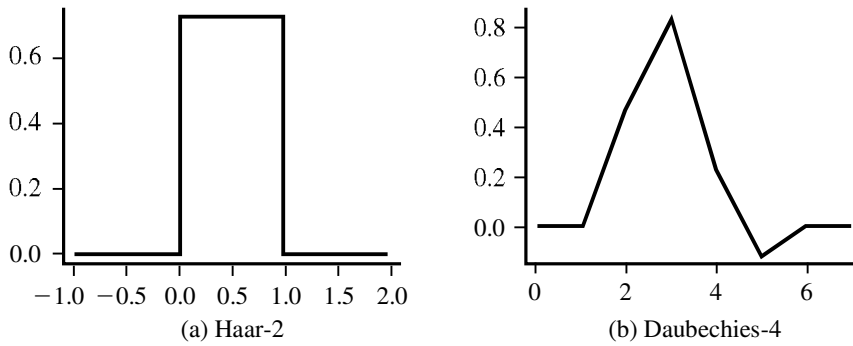
#### Wavelet Transforms

The **discrete wavelet transform** (DWT) is a linear signal processing technique that, when applied to a data vector  $\mathbf{X}$ , transforms it to a numerically different vector,  $\mathbf{X}'$ , 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,  $\mathbf{X} = (x_1, x_2, \dots, x_n)$ , depicting  $n$  measurements made on the tuple from  $n$  database attributes.<sup>8</sup>

“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 of the 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.

---

<sup>8</sup>In our notation, any variable representing a vector is shown in bold italic font; measurements depicting the vector are shown in italic font.



**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.

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 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. Figure 2.16 shows some wavelet families. Popular wavelet transforms include the Haar-2, Daubechies-4, and Daubechies-6 transforms. 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 a 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  $\mathbf{X}$ , that is, to all pairs of measurements  $(x_{2i}, x_{2i+1})$ . This results in two sets of data 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 sets of data obtained in the previous loop, until the resulting data sets obtained are of length 2.
5. Selected values from the data sets obtained in the above iterations are designated 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. By factoring the matrix used into a product 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.

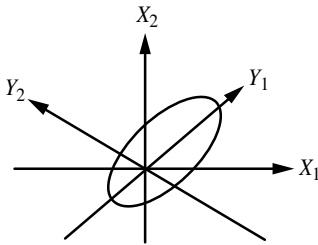
## Principal Components Analysis

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.

Suppose that the data to be reduced consist of tuples or data vectors described by  $n$  attributes or dimensions. **Principal components analysis**, or PCA (also called the Karhunen-Loeve, or K-L, method), searches for  $k$   $n$ -dimensional orthogonal vectors that can best be used to represent the data, where  $k \leq n$ . The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. Unlike attribute subset selection, 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 each point in a direction perpendicular to the others. 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



**Figure 2.17** Principal components analysis.  $Y_1$  and  $Y_2$  are the first two principal components for the given data.

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, Figure 2.17 shows the first two principal components,  $Y_1$  and  $Y_2$ , 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 according to decreasing order of “significance,” the size of the data 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 is computationally inexpensive, 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. In comparison with wavelet transforms, PCA tends to be better at handling sparse data, whereas wavelet transforms are more suitable for data of high dimensionality.

### 2.5.4 Numerosity Reduction

“Can we reduce the data volume by choosing alternative, ‘smaller’ forms of data representation?” Techniques of *numerosity reduction* can indeed be applied for this purpose. These techniques may be parametric or nonparametric. For *parametric methods*, a model is used to estimate the data, so that typically only the data parameters need to be stored, instead of the actual data. (Outliers may also be stored.) Log-linear models, which estimate discrete multidimensional probability distributions, are an example. *Nonparametric methods* for storing reduced representations of the data include histograms, clustering, and sampling.

Let’s look at each of the numerosity reduction techniques mentioned above.

## Regression and Log-Linear Models

Regression and log-linear models can be used to approximate the given data. In (simple) **linear regression**, the data are modeled to fit a straight line. For example, a random variable,  $y$  (called a *response variable*), can be modeled as a linear function of another random variable,  $x$  (called a *predictor variable*), with the equation

$$y = wx + b, \quad (2.14)$$

where the variance of  $y$  is assumed to be constant. In the context of data mining,  $x$  and  $y$  are numerical database attributes. The coefficients,  $w$  and  $b$  (called *regression coefficients*), specify the slope of the line and the  $Y$ -intercept, respectively. These coefficients can be solved for by the *method of least squares*, which minimizes the error between the actual line separating the data and the estimate of the line. **Multiple linear regression** is an extension of (simple) linear regression, which allows a response variable,  $y$ , to be modeled as a linear function of two or more predictor variables.

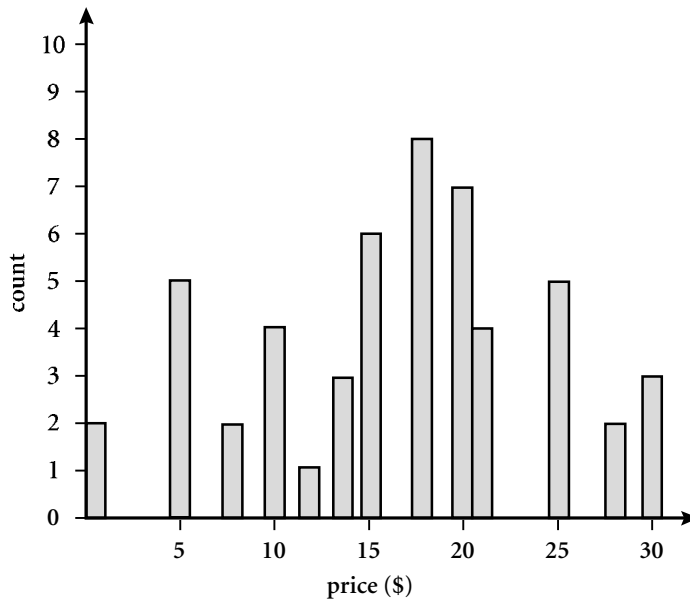
**Log-linear models** approximate discrete multidimensional probability distributions. Given a set of tuples in  $n$  dimensions (e.g., described by  $n$  attributes), we can consider each tuple as a point in an  $n$ -dimensional space. Log-linear models can be used to estimate the probability of each point in a multidimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations. This allows a higher-dimensional data space to be constructed from lower-dimensional spaces. Log-linear models are therefore also useful for dimensionality reduction (since the lower-dimensional points together typically occupy less space than the original data points) and data smoothing (since aggregate estimates in the lower-dimensional space are less subject to sampling variations than the estimates in the higher-dimensional space).

Regression and log-linear models can both be used on sparse data, although their application may be limited. While both methods can handle skewed data, regression does exceptionally well. Regression can be computationally intensive when applied to high-dimensional data, whereas log-linear models show good scalability for up to 10 or so dimensions. Regression and log-linear models are further discussed in Section 6.11.

## Histograms

Histograms use binning to approximate data distributions and are a popular form of data reduction. Histograms were introduced in Section 2.2.3. A **histogram** for an attribute,  $A$ , partitions the data distribution of  $A$  into disjoint subsets, or *buckets*. If each bucket represents only a single attribute-value/frequency pair, the buckets are called *singleton buckets*. Often, buckets instead represent continuous ranges for the given attribute.

**Example 2.5 Histograms.** The following data are a list of prices of commonly sold items at *AllElectronics* (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, 14, 15, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.

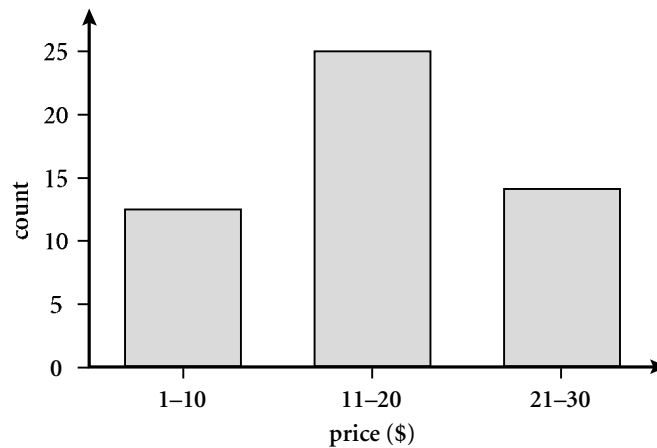


**Figure 2.18** A histogram for *price* using singleton buckets—each bucket represents one price-value/frequency pair.

Figure 2.18 shows a histogram for the data using singleton buckets. To further reduce the data, it is common to have each bucket denote a continuous range of values for the given attribute. In Figure 2.19, 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 (such as the width of \$10 for the buckets in Figure 2.19).
- **Equal-frequency** (or *equidepth*): In an equal-frequency histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (that is, each bucket contains roughly the same number of contiguous data samples).
- **V-Optimal:** If we consider all of the possible histograms for a given number of buckets, the V-Optimal histogram is the one with the least variance. Histogram variance is a weighted sum of the original values that each bucket represents, where bucket weight is equal to the number of values in the bucket.
- **MaxDiff:** In a MaxDiff histogram, we consider the difference between each pair of adjacent values. A bucket boundary is established between each pair for pairs having the  $\beta - 1$  largest differences, where  $\beta$  is the user-specified number of buckets.



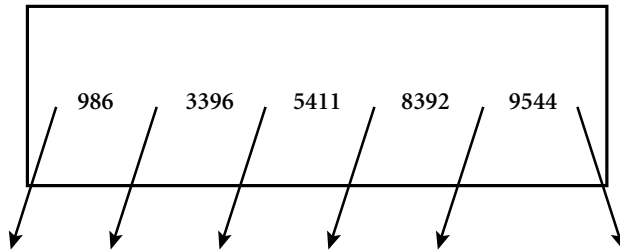
**Figure 2.19** An equal-width histogram for *price*, where values are aggregated so that each bucket has a uniform width of \$10.

V-Optimal and MaxDiff histograms tend to be the most accurate and practical. Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data. The histograms described above for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. Such histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for very high dimensions. Singleton buckets are useful for storing outliers with high frequency.

## Clustering

Clustering techniques consider data tuples as objects. They partition the objects into groups or *clusters*, so that objects within a cluster are “similar” to one another and “dissimilar” to objects in other clusters. Similarity is commonly defined in terms of how “close” the objects are in space, based on a distance function. The “quality” of a cluster may be represented by its *diameter*, the maximum distance between any two objects in the cluster. *Centroid distance* is an alternative measure of cluster quality and is defined as the average distance of each cluster object from the cluster centroid (denoting the “average object,” or average point in space for the cluster). Figure 2.12 of Section 2.3.2 shows a 2-D plot of customer data with respect to customer locations in a city, where the centroid of each cluster is shown with a “+”. Three data clusters are visible.

In data reduction, the cluster representations of the data are used to replace the actual data. The effectiveness of this technique depends on the nature of the data. It is much more effective for data that can be organized into distinct clusters than for smeared data.



**Figure 2.20** The root of a B+-tree for a given set of data.

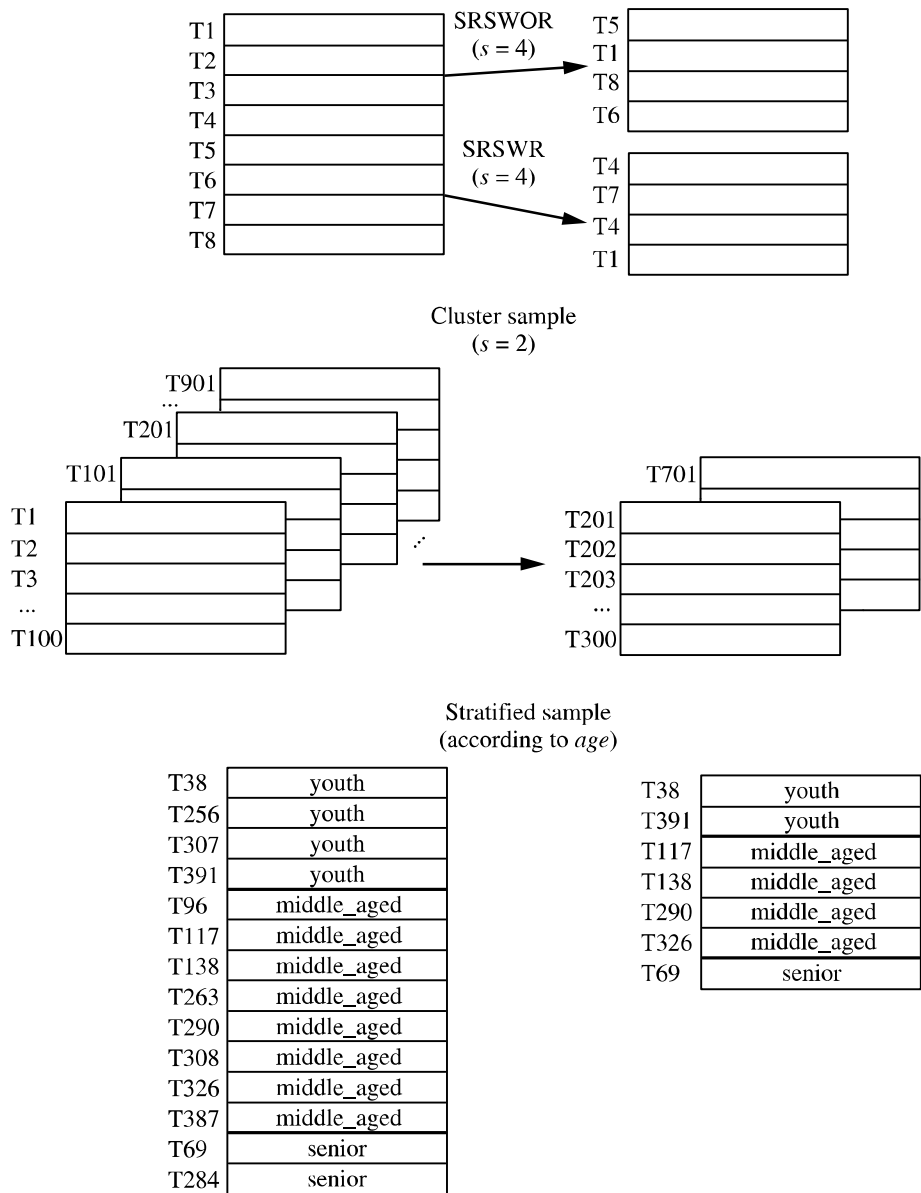
In database systems, **multidimensional index trees** are primarily used for providing fast data access. They can also be used for hierarchical data reduction, providing a multiresolution clustering of the data. This can be used to provide approximate answers to queries. An index tree recursively partitions the multidimensional space for a given set of data objects, with the root node representing the entire space. Such trees are typically balanced, consisting of internal and leaf nodes. Each parent node contains keys and pointers to child nodes that, collectively, represent the space represented by the parent node. Each leaf node contains pointers to the data tuples they represent (or to the actual tuples).

An index tree can therefore store aggregate and detail data at varying levels of resolution or abstraction. It provides a hierarchy of clusterings of the data set, where each cluster has a label that holds for the data contained in the cluster. If we consider each child of a parent node as a bucket, then an index tree can be considered as a *hierarchical histogram*. For example, consider the root of a B+-tree as shown in Figure 2.20, with pointers to the data keys 986, 3396, 5411, 8392, and 9544. Suppose that the tree contains 10,000 tuples with keys ranging from 1 to 9999. The data in the tree can be approximated by an equal-frequency histogram of six buckets for the key ranges 1 to 985, 986 to 3395, 3396 to 5410, 5411 to 8391, 8392 to 9543, and 9544 to 9999. Each bucket contains roughly 10,000/6 items. Similarly, each bucket is subdivided into smaller buckets, allowing for aggregate data at a finer-detailed level. The use of multidimensional index trees as a form of data reduction relies on an ordering of the attribute values in each dimension. Two-dimensional or multidimensional index trees include R-trees, quad-trees, and their variations. They are well suited for handling both sparse and skewed data.

There are many measures for defining clusters and cluster quality. Clustering methods are further described in Chapter 7.

## 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 sample (or subset) of the data. 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, as illustrated in Figure 2.21.



**Figure 2.21** Sampling can be used for data reduction.

- **Simple random sample without replacement (SRSWOR) of size  $s$ :** This is created by drawing  $s$  of the  $N$  tuples from  $D$  ( $s < N$ ), where the probability of drawing any tuple in  $D$  is  $1/N$ , that is, all tuples are equally likely to be sampled.
- **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 an SRS 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 an SRS 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, 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 Data Discretization and Concept Hierarchy Generation

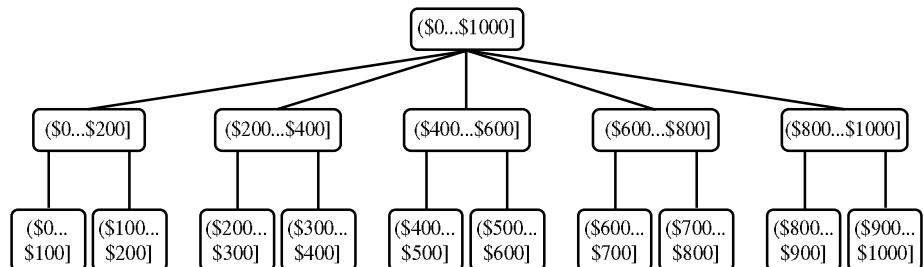
**Data discretization techniques** can be used to reduce the number of values for a given continuous attribute by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values. Replacing numerous values of a continuous attribute by a small number of interval labels thereby reduces and simplifies the original data. This leads to a concise, easy-to-use, knowledge-level representation of mining results.



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 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. Discretization can be performed recursively on an attribute to provide a hierarchical or multiresolution partitioning of the attribute values, known as a concept hierarchy. Concept hierarchies are useful for mining at multiple levels of abstraction.

A concept hierarchy for a given numerical attribute defines a discretization of the attribute. Concept hierarchies can be used to reduce the data by collecting and replacing low-level concepts (such as numerical values for the attribute *age*) with higher-level concepts (such as *youth*, *middle-aged*, or *senior*). Although detail is lost by such data generalization, the generalized data may be more meaningful and easier to interpret. This contributes to a consistent representation of data mining results among multiple mining tasks, which is a common requirement. In addition, mining on a reduced data set requires fewer input/output operations and is more efficient than mining on a larger, ungeneralized data set. Because of these benefits, discretization techniques and concept hierarchies are typically applied before data mining as a preprocessing step, rather than during mining. An example of a concept hierarchy for the attribute *price* is given in Figure 2.22. More than one concept hierarchy can be defined for the same attribute in order to accommodate the needs of various users.

Manual definition of concept hierarchies can be a tedious and time-consuming task for a user or a domain expert. Fortunately, several discretization methods can be used to automatically generate or dynamically refine concept hierarchies for numerical attributes. Furthermore, many hierarchies for categorical attributes are



**Figure 2.22** A concept hierarchy for the attribute *price*, where an interval  $(\$X... \$Y]$  denotes the range from  $\$X$  (exclusive) to  $\$Y$  (inclusive).

implicit within the database schema and can be automatically defined at the schema definition level.

Let's look at the generation of concept hierarchies for numerical and categorical data.

## 2.6.1 Discretization and Concept Hierarchy Generation for Numerical Data

It is difficult and laborious to specify concept hierarchies for numerical attributes because of the wide diversity of possible data ranges and the frequent updates of data values. Such manual specification can also be quite arbitrary.

Concept hierarchies for numerical attributes can be constructed automatically based on data discretization. We examine the following methods: *binning*, *histogram analysis*, *entropy-based discretization*,  $\chi^2$ -*merging*, *cluster analysis*, and *discretization by intuitive partitioning*. In general, each method assumes that the values to be discretized are sorted in ascending order.

### Binning

Binning is a top-down splitting technique based on a specified number of bins. Section 2.3.2 discussed binning methods for data smoothing. These methods are also used as discretization methods for numerosity 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 in order 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.

### Histogram Analysis

Like binning, histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms partition the values for an attribute,  $A$ , into disjoint ranges called *buckets*. Histograms were introduced in Section 2.2.3. Partitioning rules for defining histograms were described in Section 2.5.4. In an *equal-width* histogram, for example, the values are partitioned into equal-sized partitions or ranges (such as in Figure 2.19 for *price*, where each bucket has a width of \$10). With an *equal-frequency* histogram, the values are partitioned so that, ideally, each partition contains the same number of data tuples. 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 control the recursive procedure. This specifies the minimum width of a partition, or the minimum number of values for each partition at each level. Histograms can also be partitioned based on cluster analysis of the data distribution, as described below.

## Entropy-Based Discretization

*Entropy* is one of the most commonly used discretization measures. It was first introduced by Claude Shannon in pioneering work on information theory and the concept of information gain. Entropy-based discretization is a supervised, top-down splitting technique. It explores class distribution information in its calculation and determination of split-points (data values for partitioning an attribute range). To discretize a numerical attribute,  $A$ , the method selects the value of  $A$  that has the minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization. Such discretization forms a concept hierarchy for  $A$ .

Let  $D$  consist of data tuples defined by a set of attributes and a class-label attribute. The class-label attribute provides the class information per tuple. The basic method for entropy-based discretization of an attribute  $A$  within the set is as follows:

1. Each value of  $A$  can be considered as a potential interval boundary or split-point (denoted *split\_point*) to partition the range of  $A$ . That is, a split-point for  $A$  can partition the tuples in  $D$  into two subsets satisfying the conditions  $A \leq \text{split\_point}$  and  $A > \text{split\_point}$ , respectively, thereby creating a binary discretization.
2. Entropy-based discretization, as mentioned above, uses information regarding the class label of tuples. To explain the intuition behind entropy-based discretization, we must take a glimpse at classification. Suppose we want to classify the tuples in  $D$  by partitioning on attribute  $A$  and some split-point. Ideally, we would like this partitioning to result in an exact classification of the tuples. For example, if we had two classes, we would hope that all of the tuples of, say, class  $C_1$  will fall into one partition, and all of the tuples of class  $C_2$  will fall into the other partition. However, this is unlikely. For example, the first partition may contain many tuples of  $C_1$ , but also some of  $C_2$ . How much more information would we still need for a perfect classification, after this partitioning? This amount is called the *expected information requirement* for classifying a tuple in  $D$  based on partitioning by  $A$ . It is given by

$$\text{Info}_A(D) = \frac{|D_1|}{|D|} \text{Entropy}(D_1) + \frac{|D_2|}{|D|} \text{Entropy}(D_2), \quad (2.15)$$

where  $D_1$  and  $D_2$  correspond to the tuples in  $D$  satisfying the conditions  $A \leq \text{split\_point}$  and  $A > \text{split\_point}$ , respectively;  $|D|$  is the number of tuples in  $D$ , and so on. The entropy function for a given set is calculated based on the class distribution of the tuples in the set. For example, given  $m$  classes,  $C_1, C_2, \dots, C_m$ , the entropy of  $D_1$  is

$$\text{Entropy}(D_1) = - \sum_{i=1}^m p_i \log_2(p_i), \quad (2.16)$$

where  $p_i$  is the probability of class  $C_i$  in  $D_1$ , determined by dividing the number of tuples of class  $C_i$  in  $D_1$  by  $|D_1|$ , the total number of tuples in  $D_1$ . Therefore, when selecting a split-point for attribute  $A$ , we want to pick the attribute value that gives the minimum expected information requirement (i.e.,  $\min(\text{Info}_A(D))$ ). This would result

in the minimum amount of expected information (still) required to perfectly classify the tuples after partitioning by  $A \leq \text{split\_point}$  and  $A > \text{split\_point}$ . This is equivalent to the attribute-value pair with the maximum information gain (the further details of which are given in Chapter 6 on classification.) Note that the value of  $\text{Entropy}(D_2)$  can be computed similarly as in Equation (2.16).

“But our task is discretization, not classification!” you may exclaim. This is true. We use the split-point to partition the range of  $A$  into two intervals, corresponding to  $A \leq \text{split\_point}$  and  $A > \text{split\_point}$ .

3. The process of determining a split-point is recursively applied to each partition obtained, until some stopping criterion is met, such as when the minimum information requirement on all candidate split-points is less than a small threshold,  $\epsilon$ , or when the number of intervals is greater than a threshold,  $\text{max\_interval}$ .

Entropy-based discretization can reduce data size. Unlike the other methods mentioned here so far, entropy-based discretization uses class information. This makes it more likely that the interval boundaries (split-points) are defined to occur in places that may help improve classification accuracy. The entropy and information gain measures described here are also used for decision tree induction. These measures are revisited in greater detail in Section 6.3.2.

## Interval Merging by $\chi^2$ Analysis

*ChiMerge* is a  $\chi^2$ -based discretization method. The discretization methods that we have studied up to this point have all employed a top-down, splitting strategy. This contrasts with *ChiMerge*, which employs a bottom-up approach by finding the best neighboring intervals and then merging these to form larger intervals, recursively. The method is supervised in that it uses class information. The basic notion is that for accurate discretization, the relative class frequencies should be fairly consistent within an interval. Therefore, if two adjacent intervals have a very similar distribution of classes, then the intervals can be merged. Otherwise, they should remain separate.

*ChiMerge* proceeds as follows. Initially, each distinct value of a numerical attribute  $A$  is considered to be one interval.  $\chi^2$  tests are performed for every pair of adjacent intervals. Adjacent intervals with the least  $\chi^2$  values are merged together, because low  $\chi^2$  values for a pair indicate similar class distributions. This merging process proceeds recursively until a predefined stopping criterion is met.

The  $\chi^2$  statistic was introduced in Section 2.4.1 on data integration, where we explained its use to detect a correlation relationship between two categorical attributes (Equation (2.9)). Because *ChiMerge* treats intervals as discrete categories, Equation (2.9) can be applied. The  $\chi^2$  statistic tests the hypothesis that two adjacent intervals for a given attribute are independent of the class. Following the method in Example 2.1, we can construct a contingency table for our data. The contingency table has two columns (representing the two adjacent intervals) and  $m$  rows, where  $m$  is the number of distinct classes. Applying Equation (2.9) here, the cell value  $o_{ij}$  is the count of tuples in the  $i^{\text{th}}$  interval and  $j^{\text{th}}$  class. Similarly, the expected frequency of  $o_{ij}$  is  $e_{ij} = (\text{number of tuples in interval}$

$i) \times (\text{number of tuples in class } j)/N$ , where  $N$  is the total number of data tuples. Low  $\chi^2$  values for an interval pair indicate that the intervals are independent of the class and can, therefore, be merged.

The stopping criterion is typically determined by three conditions. First, merging stops when  $\chi^2$  values of all pairs of adjacent intervals exceed some threshold, which is determined by a specified significance level. A too (or very) high value of significance level for the  $\chi^2$  test may cause overdiscretization, whereas a too (or very) low value may lead to underdiscretization. Typically, the significance level is set between 0.10 and 0.01. Second, the number of intervals cannot be over a prespecified *max-interval*, such as 10 to 15. Finally, recall that the premise behind ChiMerge is that the relative class frequencies should be fairly consistent within an interval. In practice, some inconsistency is allowed, although this should be no more than a prespecified threshold, such as 3%, which may be estimated from the training data. This last condition can be used to remove irrelevant attributes from the data set.

## Cluster Analysis

Cluster analysis is a popular data discretization method. A clustering algorithm can be applied to discretize a numerical attribute,  $A$ , by partitioning the values of  $A$  into clusters or groups. Clustering takes the distribution of  $A$  into consideration, as well as the closeness of data points, and therefore is able to produce high-quality discretization results. Clustering can be used to generate a concept hierarchy for  $A$  by following either a top-down splitting strategy or a bottom-up merging strategy, where each cluster forms a node of the concept hierarchy. In the former, each initial cluster or partition may be further decomposed into several subclusters, forming a lower level of the hierarchy. In the latter, clusters are formed by repeatedly grouping neighboring clusters in order to form higher-level concepts. Clustering methods for data mining are studied in Chapter 7.

## Discretization by Intuitive Partitioning

Although the above discretization methods are useful in the generation of numerical hierarchies, many users would like to see numerical ranges partitioned into relatively uniform, easy-to-read intervals that appear intuitive or “natural.” For example, annual salaries broken into ranges like (\$50,000, \$60,000] are often more desirable than ranges like (\$51,263.98, \$60,872.34], obtained by, say, some sophisticated clustering analysis.

The 3-4-5 rule can be used to segment numerical data into relatively uniform, natural-seeming intervals. In general, the rule partitions a given range of data into 3, 4, or 5 relatively equal-width intervals, recursively and level by level, based on the value range at the most significant digit. We will illustrate the use of the rule with an example further below. The rule is as follows:

- If an interval covers 3, 6, 7, or 9 distinct values at the most significant digit, then partition the range into 3 intervals (3 equal-width intervals for 3, 6, and 9; and 3 intervals in the grouping of 2-3-2 for 7).

- If it covers 2, 4, or 8 distinct values at the most significant digit, then partition the range into 4 equal-width intervals.
- If it covers 1, 5, or 10 distinct values at the most significant digit, then partition the range into 5 equal-width intervals.

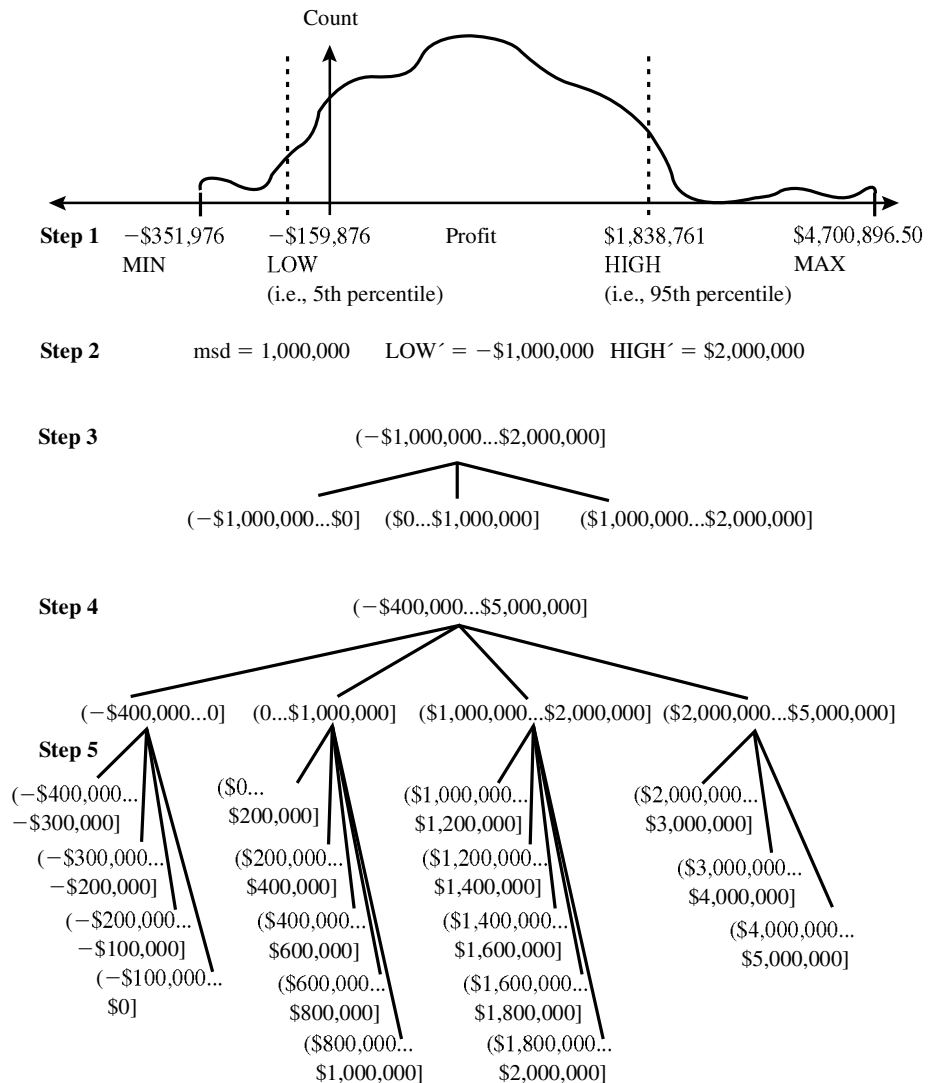
The rule can be recursively applied to each interval, creating a concept hierarchy for the given numerical attribute. Real-world data often contain extremely large positive and/or negative outlier values, which could distort any top-down discretization method based on minimum and maximum data values. For example, the assets of a few people could be several orders of magnitude higher than those of others in the same data set. Discretization based on the maximal asset values may lead to a highly biased hierarchy. Thus the top-level discretization can be performed based on the range of data values representing the majority (e.g., 5th percentile to 95th percentile) of the given data. The extremely high or low values beyond the top-level discretization will form distinct interval(s) that can be handled separately, but in a similar manner.

The following example illustrates the use of the 3-4-5 rule for the automatic construction of a numerical hierarchy.

**Example 2.6** **Numeric concept hierarchy generation by intuitive partitioning.** Suppose that profits at different branches of *AllElectronics* for the year 2004 cover a wide range, from  $-\$351,976.00$  to  $\$4,700,896.50$ . A user desires the automatic generation of a concept hierarchy for *profit*. For improved readability, we use the notation  $(l...r]$  to represent the interval  $[l, r]$ . For example,  $(-\$1,000,000...\$0]$  denotes the range from  $-\$1,000,000$  (exclusive) to  $\$0$  (inclusive).

Suppose that the data within the 5th percentile and 95th percentile are between  $-\$159,876$  and  $\$1,838,761$ . The results of applying the 3-4-5 rule are shown in Figure 2.23.

1. Based on the above information, the minimum and maximum values are  $MIN = -\$351,976.00$ , and  $MAX = \$4,700,896.50$ . The low (5th percentile) and high (95th percentile) values to be considered for the top or first level of discretization are  $LOW = -\$159,876$ , and  $HIGH = \$1,838,761$ .
2. Given  $LOW$  and  $HIGH$ , the most significant digit (*msd*) is at the million dollar digit position (i.e.,  $msd = 1,000,000$ ). Rounding  $LOW$  down to the million dollar digit, we get  $LOW' = -\$1,000,000$ ; rounding  $HIGH$  up to the million dollar digit, we get  $HIGH' = +\$2,000,000$ .
3. Since this interval ranges over three distinct values at the most significant digit, that is,  $(2,000,000 - (-1,000,000))/1,000,000 = 3$ , the segment is partitioned into three equal-width subsegments according to the 3-4-5 rule:  $(-\$1,000,000...\$0]$ ,  $(\$0...\$1,000,000]$ , and  $(\$1,000,000...\$2,000,000]$ . This represents the top tier of the hierarchy.



**Figure 2.23** Automatic generation of a concept hierarchy for *profit* based on the 3-4-5 rule.

4. We now examine the MIN and MAX values to see how they “fit” into the first-level partitions. Since the first interval ( $-\$1,000,000 \dots \$0$ ] covers the *MIN* value, that is,  $LOW' < MIN$ , we can adjust the left boundary of this interval to make the interval smaller. The most significant digit of *MIN* is the hundred thousand digit position.

Rounding  $MIN$  down to this position, we get  $MIN' = -\$400,000$ . Therefore, the first interval is redefined as  $(-\$400,000 \dots 0]$ .

Since the last interval,  $(\$1,000,000 \dots \$2,000,000]$ , does not cover the  $MAX$  value, that is,  $MAX > HIGH'$ , we need to create a new interval to cover it. Rounding up  $MAX$  at its most significant digit position, the new interval is  $(\$2,000,000 \dots \$5,000,000]$ . Hence, the topmost level of the hierarchy contains four partitions,  $(-\$400,000 \dots \$0]$ ,  $(\$0 \dots \$1,000,000]$ ,  $(\$1,000,000 \dots \$2,000,000]$ , and  $(\$2,000,000 \dots \$5,000,000]$ .

5. Recursively, each interval can be further partitioned according to the 3-4-5 rule to form the next lower level of the hierarchy:
  - The first interval,  $(-\$400,000 \dots \$0]$ , is partitioned into 4 subintervals:  $(-\$400,000 \dots -\$300,000]$ ,  $(-\$300,000 \dots -\$200,000]$ ,  $(-\$200,000 \dots -\$100,000]$ , and  $(-\$100,000 \dots \$0]$ .
  - The second interval,  $(\$0 \dots \$1,000,000]$ , is partitioned into 5 subintervals:  $(\$0 \dots \$200,000]$ ,  $(\$200,000 \dots \$400,000]$ ,  $(\$400,000 \dots \$600,000]$ ,  $(\$600,000 \dots \$800,000]$ , and  $(\$800,000 \dots \$1,000,000]$ .
  - The third interval,  $(\$1,000,000 \dots \$2,000,000]$ , is partitioned into 5 subintervals:  $(\$1,000,000 \dots \$1,200,000]$ ,  $(\$1,200,000 \dots \$1,400,000]$ ,  $(\$1,400,000 \dots \$1,600,000]$ ,  $(\$1,600,000 \dots \$1,800,000]$ , and  $(\$1,800,000 \dots \$2,000,000]$ .
  - The last interval,  $(\$2,000,000 \dots \$5,000,000]$ , is partitioned into 3 subintervals:  $(\$2,000,000 \dots \$3,000,000]$ ,  $(\$3,000,000 \dots \$4,000,000]$ , and  $(\$4,000,000 \dots \$5,000,000]$ .

Similarly, the 3-4-5 rule can be carried on iteratively at deeper levels, as necessary. ■

### 2.6.2 Concept Hierarchy Generation for Categorical Data

Categorical data are discrete data. Categorical attributes have a finite (but possibly large) number of distinct values, with no ordering among the values. Examples include *geographic location*, *job category*, and *item type*. There are several methods for the generation of concept hierarchies for categorical data.

**Specification of a partial ordering of attributes explicitly at the schema level by users or experts:** Concept hierarchies for categorical attributes or dimensions typically involve a group of attributes. A user or expert can easily define a concept hierarchy by specifying a partial or total ordering of the attributes at the schema level. For example, a relational database or a dimension *location* of a data warehouse may contain the following group of attributes: *street*, *city*, *province\_or\_state*, and *country*. A hierarchy can be defined by specifying the total ordering among these attributes at the schema level, such as  $street < city < province\_or\_state < country$ .

**Specification of a portion of a hierarchy by explicit data grouping:** This is essentially the manual definition of a portion of a concept hierarchy. In a large database, it



is unrealistic to define an entire concept hierarchy by explicit value enumeration. On the contrary, we can easily specify explicit groupings for a small portion of intermediate-level data. For example, after specifying that *province* and *country* form a hierarchy at the schema level, a user could define some intermediate levels manually, such as “ $\{\textit{Alberta}, \textit{Saskatchewan}, \textit{Manitoba}\} \subset \textit{prairies.Canada}$ ” and “ $\{\textit{British Columbia}, \textit{prairies.Canada}\} \subset \textit{Western.Canada}$ ”.

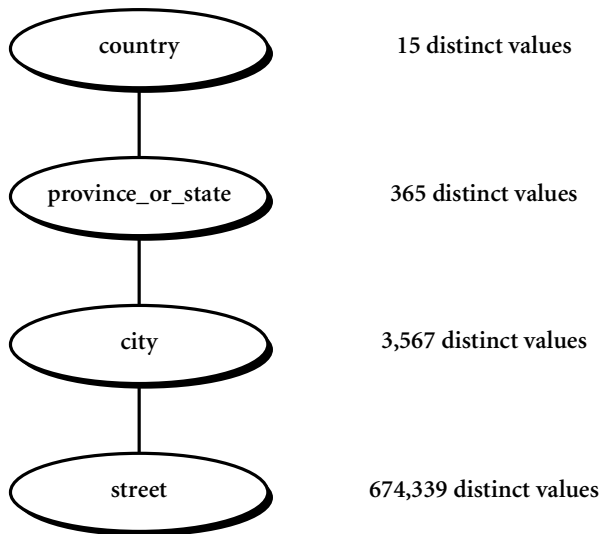
**Specification of a set of attributes, but not of their partial ordering:** A user may specify a set of attributes forming a concept hierarchy, but omit to explicitly state their partial ordering. The system can then try to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy. “*Without knowledge of data semantics, how can a hierarchical ordering for an arbitrary set of categorical attributes be found?*” Consider the following observation that since higher-level concepts generally cover several subordinate lower-level concepts, an attribute defining a high concept level (e.g., *country*) will usually contain a smaller number of distinct values than an attribute defining a lower concept level (e.g., *street*). Based on this observation, a concept hierarchy can be automatically generated based on the number of distinct values per attribute in the given attribute set. The attribute with the most distinct values is placed at the lowest level of the hierarchy. The lower the number of distinct values an attribute has, the higher it is in the generated concept hierarchy. This heuristic rule works well in many cases. Some local-level swapping or adjustments may be applied by users or experts, when necessary, after examination of the generated hierarchy.

Let’s examine an example of this method.

**Example 2.7** Concept hierarchy generation based on the number of distinct values per attribute. Suppose a user selects a set of location-oriented attributes, *street*, *country*, *province\_or\_state*, and *city*, from the *AllElectronics* database, but does not specify the hierarchical ordering among the attributes.

A concept hierarchy for *location* can be generated automatically, as illustrated in Figure 2.24. First, sort the attributes in ascending order based on the number of distinct values in each attribute. This results in the following (where the number of distinct values per attribute is shown in parentheses): *country* (15), *province\_or\_state* (365), *city* (3567), and *street* (674,339). Second, generate the hierarchy from the top down according to the sorted order, with the first attribute at the top level and the last attribute at the bottom level. Finally, the user can examine the generated hierarchy, and when necessary, modify it to reflect desired semantic relationships among the attributes. In this example, it is obvious that there is no need to modify the generated hierarchy. ■

Note that this heuristic rule is not foolproof. For example, a time dimension in a database may contain 20 distinct years, 12 distinct months, and 7 distinct days of the week. However, this does not suggest that the time hierarchy should be “*year* < *month* < *days\_of\_the\_week*”, with *days\_of\_the\_week* at the top of the hierarchy.



**Figure 2.24** Automatic generation of a schema concept hierarchy based on the number of distinct attribute values.

**Specification of only a partial set of attributes:** Sometimes a user can be sloppy when defining a hierarchy, or have only a vague idea about what should be included in a hierarchy. Consequently, the user may have included only a small subset of the relevant attributes in the hierarchy specification. For example, instead of including all of the hierarchically relevant attributes for *location*, the user may have specified only *street* and *city*. To handle such partially specified hierarchies, it is important to embed data semantics in the database schema so that attributes with tight semantic connections can be pinned together. In this way, the specification of one attribute may trigger a whole group of semantically tightly linked attributes to be “dragged in” to form a complete hierarchy. Users, however, should have the option to override this feature, as necessary.

**Example 2.8** **Concept hierarchy generation using prespecified semantic connections.** Suppose that a data mining expert (serving as an administrator) has pinned together the five attributes *number*, *street*, *city*, *province\_or\_state*, and *country*, because they are closely linked semantically regarding the notion of *location*. If a user were to specify only the attribute *city* for a hierarchy defining *location*, the system can automatically drag in all of the above five semantically related attributes to form a hierarchy. The user may choose to drop any of these attributes, such as *number* and *street*, from the hierarchy, keeping *city* as the lowest conceptual level in the hierarchy. ■

## 2.7 Summary

- **Data preprocessing** is an important issue for both data warehousing and data mining, as real-world data tend to be incomplete, noisy, and inconsistent. Data preprocessing includes data cleaning, data integration, data transformation, and data reduction.
- **Descriptive data summarization** provides 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*, *quartiles*, *interquartile range*, *variance*, and *standard deviation* for measuring the dispersion of data. Graphical representations, such as *histograms*, *boxplots*, *quantile plots*, *quantile-quantile plots*, *scatter plots*, and *scatter-plot matrices*, facilitate visual inspection of the data and are thus useful for data preprocessing and mining.
- **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. Metadata, correlation analysis, data conflict detection, and the resolution of semantic heterogeneity contribute toward smooth data integration.
- **Data transformation** routines convert the data into appropriate forms for mining. For example, attribute data may be **normalized** so as to fall between a small range, such as 0.0 to 1.0.
- **Data reduction** techniques such as data cube aggregation, attribute subset selection, dimensionality reduction, numerosity reduction, and discretization can be used to obtain a reduced representation of the data while minimizing the loss of information content.
- **Data discretization and automatic generation of concept hierarchies** for numerical data can involve techniques such as binning, histogram analysis, entropy-based discretization,  $\chi^2$  analysis, cluster analysis, and discretization by intuitive partitioning. For categorical data, concept hierarchies may be generated based on the number of distinct values of the attributes defining the hierarchy.
- Although numerous methods of data preprocessing have been developed, data preprocessing remains an active area of research, due to the huge amount of inconsistent or dirty data and the complexity of the problem.

### Exercises

- 2.1 *Data quality* can be assessed in terms of accuracy, completeness, and consistency. Propose two other dimensions of data quality.

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

<i>age</i>	<i>frequency</i>
1–5	200
5–15	450
15–20	300
20–50	1500
50–80	700
80–110	44

Compute an *approximate median* value for the data.

- 2.3 Give three additional commonly used statistical measures (i.e., not illustrated in this chapter) for the characterization of *data dispersion*, and discuss how they can be computed efficiently in large databases.
- 2.4 Suppose that the data for analysis includes the attribute *age*. The *age* values for the data tuples are (in increasing 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.
- What is the *mean* of the data? What is the *median*?
  - What is the *mode* of the data? Comment on the data's modality (i.e., bimodal, trimodal, etc.).
  - What is the *midrange* of the data?
  - Can you find (roughly) the first quartile ( $Q_1$ ) and the third quartile ( $Q_3$ ) of the data?
  - Give the *five-number summary* of the data.
  - Show a *boxplot* of the data.
  - How is a *quantile-quantile plot* different from a *quantile plot*?
- 2.5 In many applications, new data sets are incrementally added to the existing large data sets. Thus an important consideration for computing descriptive data summary is whether a measure can be computed efficiently in incremental manner. Use *count*, *standard deviation*, and *median* as examples to show that a distributive or algebraic measure facilitates efficient incremental computation, whereas a holistic measure does not.
- 2.6 In real-world data, tuples with *missing values* for some attributes are a common occurrence. Describe various methods for handling this problem.
- 2.7 Using the data for *age* given in Exercise 2.4, answer the following.
- Use *smoothing by bin means* to smooth the data, using a bin depth of 3. Illustrate your steps. Comment on the effect of this technique for the given data.
  - How might you determine *outliers* in the data?
  - What other methods are there for *data smoothing*?

2.8 Discuss issues to consider during *data integration*.

2.9 Suppose a hospital tested the age and body fat data for 18 randomly selected adults with the following result:

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

- Calculate the mean, median, and standard deviation of *age* and *%fat*.
- Draw the boxplots for *age* and *%fat*.
- Draw a *scatter plot* and a *q-q plot* based on these two variables.
- Normalize the two variables based on *z-score normalization*.
- Calculate the *correlation coefficient* (Pearson's product moment coefficient). Are these two variables positively or negatively correlated?

2.10 What are the value ranges of the following *normalization methods*?

- min-max normalization
- z-score normalization
- normalization by decimal scaling

2.11 Use the two methods below to *normalize* the following group of data:

200, 300, 400, 600, 1000

- min-max normalization by setting *min* = 0 and *max* = 1
- z-score normalization

2.12 Using the data for *age* given in Exercise 2.4, answer the following:

- Use min-max normalization to transform the value 35 for *age* onto the range [0.0, 1.0].
- Use z-score normalization to transform the value 35 for *age*, where the standard deviation of *age* is 12.94 years.
- Use normalization by decimal scaling to transform the value 35 for *age*.
- Comment on which method you would prefer to use for the given data, giving reasons as to why.

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

- stepwise forward selection
- stepwise backward elimination
- a combination of forward selection and backward elimination

- 2.14 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:
- equal-frequency (equidepth) partitioning
  - equal-width partitioning
  - clustering
- 2.15 Using the data for *age* given in Exercise 2.4,
- Plot an equal-width histogram of width 10.
  - Sketch examples of each of the following sampling techniques: SRSWOR, SRSWR, cluster sampling, stratified sampling. Use samples of size 5 and the strata “youth,” “middle-aged,” and “senior.”
- 2.16 [Contributed by Chen Chen] The *median* is one of the most important holistic 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.
- 2.17 [Contributed by Deng Cai] It is important to define or select similarity measures in data analysis. However, there is no commonly accepted subjective similarity measure. Using different similarity measures may deduce different results. Nonetheless, some apparently different similarity measures may be equivalent after some transformation.
- Suppose we have the following two-dimensional 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

- Consider the data as two-dimensional data points. Given a new data point,  $x = (1.4, 1.6)$  as a query, rank the database points based on similarity with the query using (1) Euclidean distance (Equation 7.5), and (2) cosine similarity (Equation 7.16).
  - 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.
- 2.18 ChiMerge [Ker92] is a supervised, bottom-up (i.e., merge-based) *data discretization* method. It relies on  $\chi^2$  analysis: adjacent intervals with the least  $\chi^2$  values are merged together until the stopping criterion is satisfied.

- (a) Briefly describe how ChiMerge works.
  - (b) Take the IRIS data set, obtained from <http://www.ics.uci.edu/~mlearn/MLRepository.html> (UC-Irvine Machine Learning Data Repository), as a data set to be discretized. Perform data discretization for each of the four numerical attributes using the ChiMerge method. (Let the stopping criteria be: *max-interval* = 6.) You need to write a small program to do this to avoid clumsy numerical computation. Submit your simple analysis and your test results: split points, final intervals, and your documented source program.
- 2.19 Propose an algorithm, in pseudo-code or in your favorite programming language, for the following:
- (a) The automatic generation of a concept hierarchy for categorical data based on the number of distinct values of attributes in the given schema
  - (b) The automatic generation of a concept hierarchy for numerical data based on the *equal-width* partitioning rule
  - (c) The automatic generation of a concept hierarchy for numerical data based on the *equal-frequency* partitioning rule
- 2.20 Robust data loading poses a challenge in database systems because the input data are often dirty. In many cases, an input record may have several missing values and some records could be *contaminated* (i.e., 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.

## Bibliographic Notes

Data preprocessing is discussed in a number of textbooks, including English [Eng99], Pyle [Pyl99], Loshin [Los01], Redman [Red01], and Dasu and Johnson [DJ03]. More specific references to individual preprocessing techniques are given below.

Methods for descriptive data summarization have been studied in the statistics literature long before the onset of computers. Good summaries of statistical descriptive data mining methods include Freedman, Pisani, and Purves [FPP97], and Devore [Dev95]. For statistics-based visualization of data using boxplots, quantile plots, quantile-quantile plots, scatter plots, and loess curves, see Cleveland [Cle93].

For discussion regarding data quality, see Redman [Red92], Wang, Storey, and Firth [WSF95], Wand and Wang [WW96], Ballou and Tayi [BT99], and Olson [Ols03]. Potter's Wheel (<http://control.cs.berkeley.edu/abc>), the interactive data cleaning tool described in Section 2.3.3, is presented in Raman and Hellerstein [RH01]. An example of the development of declarative languages for the specification of data transformation operators is given in Galhardas, Florescu, Shasha, et al. [GFS<sup>+</sup>01]. The handling of missing attribute values is discussed in Friedman [Fri77], Breiman, Friedman, Olshen,