2

PREPARING THE DATA

Chapter Objectives

- Analyze basic representations and characteristics of raw and large data sets.
- Apply different normalization techniques on numerical attributes.
- Recognize different techniques for data preparation, including attribute transformation.
- Compare different methods for elimination of missing values.
- Construct a method for uniform representation of time-dependent data.
- Compare different techniques for outlier detection.
- Implement some data preprocessing techniques.

Data Mining: Concepts, Models, Methods, and Algorithms, Third Edition. Mehmed Kantardzic.

Published 2020 by John Wiley & Sons, Inc.

2.1 REPRESENTATION OF RAW DATA

Data samples introduced as rows in Figure 1.5 are basic components in a data-mining process. Every sample is described with several features and there are different types of values for every feature. We will start with the two most common types: *numeric* and *categorical*. Numeric values include real-value variables or integer variables such as age, speed, or length. A feature with numeric values has two important properties: its values have an order relation ($2 < 5$ and $5 < 7$) and a distance relation ($d(2.3, 4.2) = 1.9$).

In contrast, categorical (often called symbolic) variables have neither of these two relations. The two values of a categorical variable can be either equal or not equal: they only support an equality relation (Blue = Blue or Red $\neq$ Black). Examples of variables of this type are eye color, sex, or country of citizenship. A categorical variable with two values can be converted, in principle, to a numeric binary variable with two values: 0 or 1. A categorical variable with N values can be converted into N binary numeric variables, namely, one binary variable for each categorical value. These coded categorical variables are known as "dummy variables" in statistics. For example, if the variable eye color has four values, namely, black, blue, green, and brown, they can be coded with four binary digits.

Feature Value	Code
Black	1000
Blue	0100
Green	0010
Brown	0001

Another way of classifying a variable, based on its values, is to look at it as a continuous variable or a discrete variable.

Continuous variables are also known as *quantitative* or *metric variables*. They are measured using either an interval scale or a ratio scale. Both scales allow the underlying variable to be defined or measured theoretically with infinite precision. The difference between these two scales lies in how the zero point is defined in the scale. The zero point in *the interval scale* is placed arbitrarily, and thus it does not indicate the complete absence of whatever is being measured. The best example of the interval scale is the temperature scale, where 0 °F does not mean a total absence of temperature. Because of the arbitrary placement of the zero point, the ratio relation does not hold true for variables measured using interval scales. For example, 80 °F does not imply twice as much heat as 40 °F. In contrast, *a ratio scale* has an absolute zero point, and, consequently, the ratio relation holds true for variables measured using this scale. Quantities such as height, length, and salary use this type of scale. Continuous variables are represented in large data sets with values that are numbers—real or integers.

Discrete variables are also called qualitative variables. Such variables are measured, or its values defined, using one of two kinds of non-metric scales—*nominal* or

ordinal. A nominal scale is an orderless scale, which uses different symbols, characters, and numbers to represent the different states (values) of the variable being measured. An example of a nominal variable is utility, where customer-type values are residential, commercial, and industrial. These values can be coded alphabetically as *A*, *B*, and *C* or numerically as 1, 2, or 3, but they do not have metric characteristics as the other numeric data have. Another example of a nominal attribute is the zip-code field available in many data sets. In both examples, the numbers used to designate different attribute values have no particular order and no necessary relation to one another.

An *ordinal scale* consists of ordered discrete gradations, e.g., rankings. An ordinal variable is a categorical variable for which an order relation is defined but not a distance relation. Some examples of an ordinal attribute are the rank of a student in a class and the gold, silver, and bronze medal positions in a sports competition. The ordered scale need not be necessarily linear; e.g. the difference between the students ranked 4th and 5th need not be identical to the difference between the students ranked 15th and 16th. All that can be established from an ordered scale for ordinal attributes with greater-than, equal-to, or less-than relations. Typically, ordinal variables encode a numeric variable onto a small set of overlapping intervals corresponding to the values of an ordinal variable. These ordinal variables are closely related to the linguistic or fuzzy variables commonly used in spoken English, e.g., AGE (with values young, middle-aged, and old) and INCOME (with values low, middle-class, upper-middle-class, and rich). More examples are given in Figure 2.1, while the formalization and use of fuzzy values in a data-mining process have been given in Chapter 14.

A special class of discrete variables is periodic variables. A *periodic variable* is a feature for which the distance relation exists but there is no order relation. Examples are days of the week, days of the month, or year. Monday and Tuesday, as the values of a feature, are closer than Monday and Thursday, but Monday can come before or after Friday.

Finally, one additional dimension of classification of data is based on its behavior with respect to time. Some data do not change with time and we consider them *static*

Type	Description	Examples	Operations
Nominal	Just label or different name to distinguish one object form another	Zip code, ID, gender	= or not =
Ordinal	The values provide the ordering of objects	Opinion, grades	< or >
Interval	Unit of measurement, but the origin is arbitrary	Celsius or Fahrenheit, calendar dates	+ or –
Ratio	Unit of measurement and the origin is not arbitrary	Temperature in Kelvin, length, counts, age, income	+, –, *, /

Figure 2.1. Variable types with examples.

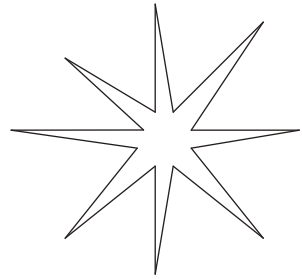


Figure 2.2. High-dimensional data looks conceptually like a porcupine.

data. On the other hand, there are attribute values that change with time and we call this type of data *dynamic* or *temporal data*. The majority of the data-mining methods are more suitable for static data, and special consideration and some preprocessing are often required to mine dynamic data.

Most data-mining problems arise because there are large amounts of samples with different types of features. Besides, these samples are very often high dimensional, which means they have extremely large number of measurable features. This additional dimension of large data sets causes the problem known in data-mining terminology as "the curse of dimensionality." The "curse of dimensionality" is produced because of the geometry of high-dimensional spaces, and these kinds of data spaces are typical for data-mining problems. The properties of high-dimensional spaces often appear counter-intuitive because our experience with the physical world is in a low-dimensional space, such as a space with two or three dimensions. Conceptually, objects in high-dimensional spaces have a larger surface area for a given volume than objects in low-dimensional spaces. For example, a high-dimensional hypercube, if it could be visualized, would look like a porcupine, as in Figure 2.2. As the dimensionality grows larger, the edges grow longer relative to the size of the central part of the hypercube. Four important properties of high-dimensional data are often the guidelines in the interpretation of input data and data-mining results:

1. The size of a data set yielding the same density of data points in an n-dimensional space increases exponentially with dimensions. For example, if a one-dimensional (1D) sample containing n data points has a satisfactory level of density, then to achieve the same density of points in k dimensions, we need n^k data points. If integers 1–100 are values of 1D samples, where the domain of the dimension is [0, 100], then to obtain the same density of samples in a five-dimensional space, we will need $100^5 = 10^{10}$ different samples. This is true even for the largest real-world data sets; because of their large dimensionality, the density of samples is still relatively low and, very often, unsatisfactory for data-mining purposes.
2. A larger radius is needed to enclose a fraction of the data points in a high-dimensional space. For a given fraction of samples, it is possible to determine the edge length e of the hypercube using the formula

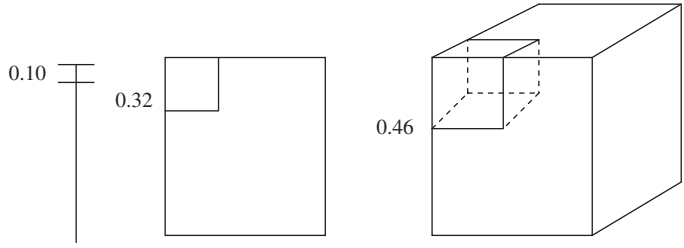


Figure 2.3. Regions enclose 10% of the samples for one-, two-, and three-dimensional space.

$$e(p) = p^{1/d}$$

where p is the prespecified fraction of samples and d is the number of dimensions. For example, if one wishes to enclose 10% of the samples ($p = 0.1$), the corresponding edge for a two-dimensional (2D) space will be $e_2(0.1) = 0.32$, for a three-dimensional space $e_3(0.1) = 0.46$, and for a 10-dimensional space $e_{10}(0.1) = 0.80$. Graphical interpretation of these edges is given in Figure 2.3.

This shows that a very large neighborhood is required to capture even a small portion of the data in a high-dimensional space.

3. Almost every point is closer to an edge than to another sample point in a high-dimensional space. For a sample of size n, the expected distance D between data points in a d-dimensional space is

$$D(d,n) = ½ \left(\frac{1}{n}\right)^{1/d}$$

For example, for a 2D space with 10,000 points, the expected distance is $D(2,10{,}000) = 0.005$ and for a 10-dimensional space with the same number of sample points $D(10,10{,}000) = 0.4$. Keep in mind that the maximum distance from any point to the edge occurs at the center of the distribution and it is 0.5 for normalized values of all dimensions.

4. Almost every point is an outlier. As the dimension of the input space increases, the distance between the prediction point and the center of the classified points increases. For example, when $d = 10$, the expected value of the prediction point is 3.1 standard deviations away from the center of the data belonging to one class. When $d = 20$, the distance is 4.4 standard deviations. From this standpoint, the prediction of every new point looks like an outlier of the initially classified data. This is illustrated conceptually in Figure 2.2, where predicted points are mostly in the edges of the porcupine, far from the central part.

These rules of the "curse of dimensionality" most often have serious consequences when dealing with a finite number of samples in a high-dimensional space. From properties (1) and (2), we see the difficulty in making local estimates for high-dimensional samples; we need more and more samples to establish the required data density for

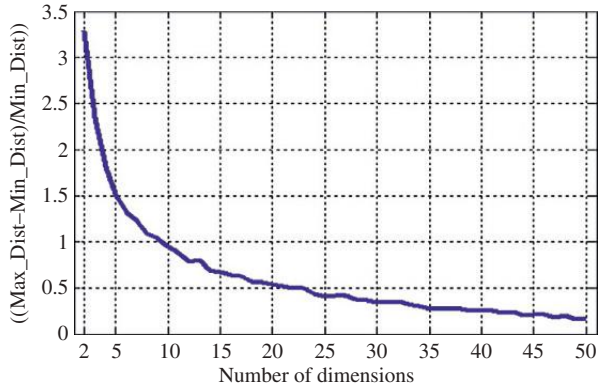


Figure 2.4. With large number of dimensions, the concept of a distance changes its meaning.

performing planned mining activities. Properties (3) and (4) indicate the difficulty of predicting a response at a given point, since any new point will on average be closer to an edge than to the training examples in the central part.

One interesting experiment, performed recently by a group of students, shows the importance of understanding curse of dimensionality concepts for data-mining tasks. They generated randomly 500 points for different n-dimensional spaces. The number of dimensions was between 2 and 50. Then, they measured, in each space, all distances between any pair of points and calculated the parameter P:

$$P_n = \frac{(\text{MAX}-\text{DIST}_n - \text{MIN}-\text{DIST}_n)}{\text{MIN}-\text{DIST}_n}$$

where n is the number of dimensions and MAX-DIST and MIN-DIST are maximum and minimum distances in the given space, respectively. The results are presented in Figure 2.4. What is interesting from the graph is that with high number of dimensions, parameter P_n is reaching the value of 0. That means maximum and minimum distances becoming so close in these spaces; in other words, there are no differences in distances between any two points in these large-dimensional spaces. It is an experimental confirmation that traditional definitions of density and distance between points, which are critical for many data-mining tasks, change its meaning! When dimensionality of a data set increases, data becomes increasingly sparse with mostly outliers in the space that it occupies. Therefore we have to revisit and reevaluate traditional concepts from statistics: distance, similarity, data distribution, mean, standard deviation, etc.

2.2 CHARACTERISTICS OF RAW DATA

All raw data sets initially prepared for data mining are often large; many are related to human beings and have the potential for being messy. A priori, one should expect to find missing values, distortions, misrecording, inadequate sampling, and so on in

these initial data sets. Raw data that do not appear to show any of these problems should immediately arouse suspicion. The only real reason for the high quality of data could be that the presented data have been cleaned up and preprocessed before the analyst sees them, as in data of a correctly designed and prepared data warehouse.

Let us see what the sources and implications of messy data are. First, data may be *missing* for a huge variety of reasons. Sometimes there are mistakes in measurements or recordings, but in many cases, the value is unavailable. To cope with this in a data-mining process, one must not only be able to model with the data that are presented, but even with their values missing. We will see later that some data-mining techniques are more or less sensitive to missing values. If the method is robust enough, then the missing values are not a problem. Otherwise, it is necessary to solve the problem of missing values before the application of a selected data-mining technique. The second cause of messy data is *misrecorded data*, and that is typical in large volumes of data. We have to have mechanisms to discover some of these "unusual" values—in some cases, even to work with them to eliminate their influence on the final results. Further, data may *not* be *from the population* they are supposed to be from. Outliers are typical examples here, and they require careful analysis before the analyst can decide whether they should be dropped from the data-mining process as anomalous or included as unusual examples from the population under study.

It is very important to examine the data thoroughly before undertaking any further steps in formal analysis. Traditionally, data-mining analysts had to familiarize themselves with their data before beginning to model them or use them with some data-mining algorithms. However, with the large size of modern data sets, this is less feasible or even entirely impossible in many cases. Here we must rely on computer programs to check the data for us.

Distorted data, incorrect choice of steps in methodology, misapplication of data-mining tools, too idealized a model, and a model that goes beyond the various sources of uncertainty and ambiguity in the data all represent possibilities for taking the wrong direction in a data-mining process. Therefore, data mining is not just a matter of simply applying a directory of tools to a given problem, but rather a process of critical assessments, exploration, testing, and evaluation. The data should be well defined, consistent, and nonvolatile in nature. The quantity of data should be large enough to support data analysis, querying, reporting, and comparisons of historical data over a long period of time.

Many experts in data mining will agree that one of the most critical steps in a data-mining process is the preparation and transformation of the initial data set. This task often receives little attention in the research literature, mostly because it is considered too application specific. But in most data-mining applications, some parts of a data-preparation process or, sometimes, even the entire process can be described independently of an application and a data-mining method. For some companies with extremely large and often distributed data sets, most of the data-preparation tasks can be performed during the design of the data warehouse, but many specialized transformations may be initialized only when a data-mining analysis is requested.

Raw data are not always (in our opinion very seldom!) the best data set prepared for data mining. Many transformations may be needed to produce features more useful for selected data-mining methods such as prediction or classification. Counting in different ways, using different sampling sizes, taking important ratios, varying data- window sizes for time-dependent data, and including changes in moving averages (MA) may contribute to better data-mining results. Do not expect that the machine will find the best set of transformations without human assistance and do not expect that transformations used in one data-mining application are the best for another.

The preparation of data is sometimes dismissed as a minor topic in the data-mining literature and used just formally as a phase in a data-mining process. In the real world of data-mining applications, the situation is reversed. More effort is expended preparing data than applying data-mining methods. There are two central tasks for the preparation of data:

1. To organize data into a standard form that is ready for processing by data-mining and other computer-based tools (a standard form is a relational table), and
2. To prepare data sets that lead to the best data-mining performances.

2.3 TRANSFORMATION OF RAW DATA

We will review a few general types of transformations of data that are not problem dependent and that may improve data-mining results. Selection of techniques and use in particular applications depend on types of data, amounts of data, and general characteristics of the data-mining task.

2.3.1 Normalizations

Some data-mining methods, typically those that are based on distance computation between points in an n-dimensional space, may need normalized data for best results. The measured values can be scaled to a specific range, e.g., [−1, 1] or [0, 1]. If the values are not normalized, the distance measures will overweight those features that have, on an average, larger values. There are many ways of normalizing data. Here are three simple and effective normalization techniques:

(a) ***Decimal scaling***

Decimal scaling moves the decimal point but still preserves most of the original digit value. The typical scale maintains the values in a range of −1 to 1. The following equation describes decimal scaling, where $v(i)$ is the value of the feature v for case i and $v'(i)$ is the scaled value

$$v'(i) = \frac{v(i)}{10^k}$$

for the smallest k such that $\max(|v'(i)|) < 1$.

First, the maximum $|v'(i)|$ is found in the data set, and then, the decimal point is moved until the new, scaled maximum absolute value is less than 1. The divisor is then applied to all other $v(i)$. For example, if the largest value in the set is 455 and the smallest value is –834, then the maximum absolute value of the feature becomes 0.834, and the divisor for all $v(i)$ is 1000 ($k = 3$).

(b) ***Min–max normalization***

Suppose that the data for a feature v are in a range between 150 and 250. Then, the previous method of normalization will give all normalized data between 0.15 and 0.25; but it will accumulate the values on a small subinterval of the entire range. To obtain better distribution of values on a whole normalized interval, e.g., [0, 1], we can use the min–max formula

$$v'(i) = \frac{(v(i) - \min(v(i)))}{(\max(v(i)) - \min(v(i)))}$$

where the minimum and the maximum values for the feature v are computed on a set automatically or they are estimated by an expert in a given domain. Similar transformation may be used for the normalized interval [−1, 1]. The automatic computation of min and max values requires one additional search through the entire data set, but, computationally, the procedure is very simple. On the other hand, expert estimations of min and max values may cause unintentional accumulation of normalized values.

(c) ***Standard deviation normalization***

Normalization by standard deviation often works well with distance measures, but transforms the data into a form unrecognizable from the original data. For a feature v, the mean value mean(v) and the standard deviation sd(v) are computed for the entire data set. Then, for a case i, the feature value is transformed using the equation

$$v*(i) = \frac{(v(i) - \text{mean}(v))}{\text{sd}(v)}$$

For example, if the initial set of values of the attribute is $v = \{1, 2, 3\}$, then mean (v) = 2, sd(v) = 1, and the new set of normalized values is $v* = \{ -1, 0, 1 \}$.

Why not treat normalization as an implicit part of a data-mining method? The simple answer is that normalizations are useful for several diverse methods of data mining. Also very important is that the normalization is not a one-time or a one-phase event. If a method requires normalized data, available data should be initially transformed and prepared for the selected data-mining technique, but an identical normalization must be applied in all other phases of data mining, and with all new and future data. Therefore, the normalization parameters must be saved along with a solution.

2.3.2 Data Smoothing

A numeric feature, y, may range over many distinct values, sometimes as many as the number of training cases. For many data-mining techniques, minor differences between

these values are not significant and may degrade the performance of the method and the final results. They may be considered as random variations of the same underlying value. Hence, it can be advantageous sometimes to smooth the values of the variable.

Many simple smoothers can be specified that average similar measured values. For example, if the values are real numbers with several decimal places, rounding the values to the given precision could be a simple smoothing algorithm for a large number of samples, where each sample has its own real value. If the set of values for the given feature F is {0.93, 1.01, 1.001, 3.02, 2.99, 5.03, 5.01, 4.98}, then it is obvious that smoothed values will be $F_{\text{smoothed}} = \{1.0, 1.0, 1.0, 3.0, 3.0, 5.0, 5.0, 5.0\}$. This simple transformation is performed without losing any quality in a data set, and, at the same time, it reduces the number of different real values for the feature to only three.

Some of these smoothing algorithms are more complex, and they are explained in Section 3.2. Reducing the number of distinct values for a feature means reducing the dimensionality of the data space at the same time. Reduced values are particularly useful for logic-based methods of data mining, as will be explained in Chapter 6. Smoothers in this case can be used to discretize continuous features into a set of features with binary true–false values.

2.3.3 Differences and Ratios

Even small changes to features can produce significant improvement in data-mining performances. The effects of relatively minor transformations of input or output features are particularly important in the specification of the data-mining goals. Two types of simple transformations, *differences* and *ratios*, could make improvements in goal specification, especially if they are applied to the output features.

These transformations sometimes produce better results than the simple initial goal of predicting a number. In one application, e.g., the objective is to move the controls for a manufacturing process to an optimal setting. But instead of optimizing the absolute magnitude specification for the output $s(t + 1)$, it will be more effective to set the goal of a relative move from current value to a final optimal $s(t + 1) - s(t)$. The range of values for the relative moves is generally much smaller than the range of values for the absolute control setting. Therefore, for many data-mining methods, a smaller number of alternatives will improve the efficiency of the algorithm and will very often give better results.

Ratios are the second simple transformation of a target or output features. Using $s(t + 1)/s(t)$ as the output of a data-mining process instead of absolute value $s(t + 1)$ means that the level of increase or decrease in the values of a feature may also improve the performances of the entire mining process.

Differences and ratio transformations are not only useful for output features but also for inputs. They can be used as changes in time for one feature or as a composition of different input features. For example, in many medical data sets, there are two features of a patient, height and weight, that are taken as input parameters for different diagnostic analyses. Many applications show that better diagnostic results are obtained when an initial transformation is performed using a new feature called the body mass index

(BMI), which is the weighted ratio between weight and height. This composite feature is better than the initial parameters to describe some of the characteristics of the patient, such as whether or not the patient is overweight.

Logical transformations can also be used to compose new features. For example, sometimes it is useful to generate a new feature that will determine the logical value of the relation $A > B$ between existing features A and B. But there are no universally best data-transformation methods. The lesson to be learned is that a major role remains for human insight while defining the problem. Attention should be paid to composing features, because relatively simple transformations can sometimes be far more effective for the final performance than switching to some other techniques of data mining.

2.4 MISSING DATA

For many real-world applications of data mining, even when there are huge amounts of data, the subset of cases with complete data may be relatively small. Available samples and also future cases may have values missing. Some of the data-mining methods accept missing values and satisfactorily process data to reach a final conclusion. Other methods require that all values be available. An obvious question is whether these missing values can be filled in during data preparation, prior to the application of the data-mining methods. The simplest solution for this problem is the reduction of the data set and the elimination of all samples with missing values. That is possible when large data sets are available, and missing values occur only in a small percentage of samples. If we do not drop the samples with missing values, then we have to find values for them. What are the practical solutions?

First, a data miner, together with the domain expert, can manually examine samples that have no values and enter a reasonable, probable, or expected value based on a domain experience. The method is straightforward for small numbers of missing values and relatively small data sets. But if there is no obvious or plausible value for each case, the miner is introducing noise into the data set by manually generating a value.

The second approach gives an even simpler solution for elimination of missing values. It is based on a formal, often automatic replacement of missing values with some constants, such as the following:

1. Replace all missing values with a single global constant (a selection of a global constant is highly application dependent).
2. Replace a missing value with its feature mean.
3. Replace a missing value with its feature mean for the given class (this approach is possible only for classification problems where samples are classified in advance).

These simple solutions are tempting. Their main flaw is that the substituted value is not the correct value. By replacing the missing value with a constant or changing the values for a few different features, the data are biased. The replaced value (values) will

homogenize the cases with missing values into a uniform subset directed toward the class with most missing values (an artificial class!). If missing values are replaced with a single global constant for all features, an unknown value may be implicitly made into a positive factor that is not objectively justified.

One possible interpretation of missing values is that they are "don't care" values. In other words, we suppose that these values do not have any influence on the final data-mining results. In that case, a sample with the missing value may be extended to the set of artificial samples, where, for each new sample, the missing value is replaced with one of the possible feature values of a given domain. Although this interpretation may look more natural, the problem with this approach is the combinatorial explosion of artificial samples. For example, if one three-dimensional sample X is given as $X = \{1, ?, 3\}$, where the second feature's value is missing, the process will generate five artificial samples for the feature domain [0, 1, 2, 3, 4]:

$$X_1 = \{1,0,3\},\ X_2 = \{1,1,3\},\ X_3 = \{1,2,3\},\ X_4 = \{1,3,3\},\ \text{and}\ X_5 = \{1,4,3\}$$

Finally, the data miner can generate a predictive model to predict each of the missing values. For example, if three features A, B, and C are given for each sample, then, based on samples that have all three values as a training set, the data miner can generate a model of correlation between features. Different techniques such as regression, Bayesian formalism, clustering, or decision-tree induction may be used depending on data types (all these techniques are explained later in this book in Chapters 5, 6, and 7). Once you have a trained model, you can present a new sample that has a value missing and generate a "predictive" value. For example, if values for features A and B are given, the model generates the value for the feature C. If a missing value is highly correlated with the other known features, this process will generate the best value for that feature. Of course, if you can always predict a missing value with certainty, this means that the feature is redundant in a data set and not necessary in further data-mining analyses. In real-world applications, you should expect an imperfect correlation between the feature with the missing value and other features. Therefore, all automatic methods fill in values that may not be correct. Such automatic methods are among the most popular in the data-mining community. In comparison to the other methods, they use the most information from the present data to predict missing values.

In general, it is speculative and often misleading to replace missing values using a simple, artificial schema of data preparation. It is best to generate multiple solutions of data mining with and without features that have missing values and then analyze and interpret them.

2.5 TIME-DEPENDENT DATA

Practical data-mining applications will range from those having strong time-dependent relationships to those with loose or no time relationships. Real-world problems with time dependencies require special preparation and transformation of data, which are, in many cases, critical for successful data mining. We will start with the

simplest case—a single feature measured over time. This feature has a series of values over fixed time units. For example, a temperature reading could be measured every hour, or the sales of a product could be recorded every day. This is the classical univariate time-series problem, where it is expected that the value of the variable X at a given time be related to previous values. Because the time series is measured at fixed units of time, the series of values can be expressed as

$$X = \{t(1), t(2), t(3), \ldots, t(n)\}$$

where $t(n)$ is the most recent value.

For many time-series problems, the goal is to forecast $t(n+1)$ from previous values of the feature, where these values are directly related to the predicted value. One of the most important steps in preprocessing of raw time-dependent data is the specification of a window or a time lag. This is the number of previous values that influence the prediction. Every window represents one sample of data for further analysis. For example, if the time series consists of the eleven measurements

$$X = \{t(0), t(1), t(2), t(3), t(4), t(5), t(6), t(7), t(8), t(9), t(10)\}$$

and if the window for analysis of the time series is five, then it is possible to reorganize the input data into a tabular form with six samples, which is more convenient (standardized) for the application of data-mining techniques. Transformed data are given in Table 2.1.

The best time lag must be determined by the usual evaluation techniques for a varying complexity measure using independent test data. Instead of preparing the data once and turning them over to the data-mining programs for prediction, additional iterations of data preparation have to be performed. While the typical goal is to predict the next value in time, in some applications, the goal can be modified to predict values in the future, several time units in advance. More formally, given the time-dependent values $t(n-i), \ldots, t(n)$, it is necessary to predict the value $t(n+j)$. In the previous example, taking $j = 3$, the new samples are given in Table 2.2.

TABLE 2.1. Transformation of Time Series to Standard Tabular Form (Window = 5)

Sample	Window					Next Value
	M1	M2	M3	M4	M5	
1	$t(0)$	$t(1)$	$t(2)$	$t(3)$	$t(4)$	$t(5)$
2	$t(1)$	$t(2)$	$t(3)$	$t(4)$	$t(5)$	$t(6)$
3	$t(2)$	$t(3)$	$t(4)$	$t(5)$	$t(6)$	$t(7)$
4	$t(3)$	$t(4)$	$t(5)$	$t(6)$	$t(7)$	$t(8)$
5	$t(4)$	$t(5)$	$t(6)$	$t(7)$	$t(8)$	$t(9)$
6	$t(5)$	$t(6)$	$t(7)$	$t(8)$	$t(9)$	$t(10)$

TABLE 2.2. Time-Series Samples in Standard Tabular Form (Window = 5) with Postponed Predictions (j = 3)

Sample	Window					Next Value
	M1	M2	M3	M4	M5	
1	$t(0)$	$t(1)$	$t(2)$	$t(3)$	$t(4)$	$t(7)$
2	$t(1)$	$t(2)$	$t(3)$	$t(4)$	$t(5)$	$t(8)$
3	$t(2)$	$t(3)$	$t(4)$	$t(5)$	$t(6)$	$t(9)$
4	$t(3)$	$t(4)$	$t(5)$	$t(6)$	$t(7)$	$t(10)$

In general, the further out in the future, the more difficult and less reliable is the forecast. The goal for a time series can easily be changed from predicting the next value in the time series to classification into one of predefined categories. From a data-preparation perspective, there are no significant changes. For example, instead of predicted output value $t(i + 1)$, the new classified output will be binary: T for $t(i + 1) \geq$ threshold value and F for $t(i + 1) <$ threshold value.

The time units can be relatively small, enlarging the number of artificial features in a tabular representation of time series for the same time period. The resulting problem of high dimensionality is the price paid for precision in the standard representation of the time-series data.

In practice, many older values of a feature may be historical relics that are no more relevant and should not be used for analysis. Therefore, for many business and social applications, new trends can make old data less reliable and less useful. This leads to a greater emphasis on recent data, possibly discarding the oldest portions of the time series. Now we are talking not only of a fixed window for the presentation of a time series but also of a fixed size for the data set. Only the n most recent cases are used for analysis, and, even then, they may not be given equal weight. These decisions must be given careful attention and are somewhat dependent on knowledge of the application and past experience. For example, using 20-year-old data about cancer patients will not give the correct picture about the chances of survival today.

Besides standard tabular representation of time series, sometimes it is necessary to additionally preprocess raw data and summarize their characteristics before application of data-mining techniques. Many times it is better to predict the difference $t(n + 1) - t(n)$ instead of the absolute value $t(n + 1)$ as the output. Also, using a ratio, $t(n + 1)/t(n)$, which indicates the percentage of changes, can sometimes give better prediction results. These transformations of the predicted values of the output are particularly useful for logic-based data-mining methods such as decision trees or rules. When differences or ratios are used to specify the goal, features measuring differences or ratios for input features may also be advantageous.

Time-dependent cases are specified in terms of a goal and a time lag or a window of size m. One way of summarizing features in the data set is to average them, producing MA. A single average summarizes the most recent m feature values for each case, and for each increment in time, its value is

$$\mathrm{MA}(i,m) = \frac{1}{m} \cdot \sum_{j=i-m+1}^{i} t(j)$$

Knowledge of the application can aid in specifying reasonable sizes for m. Error estimation should validate these choices. Moving averages weight all time points equally in the average. Typical examples are MA in the stock market such as 200-day MA for DOW or NASDAQ. The objective is to smooth neighboring time points by an MA to reduce the random variation and noise components:

$$\mathrm{MA}(i,m) = t(i) = \mathrm{mean}(i) + \mathrm{error}$$

Another type of average is an *exponential moving average* (EMA) that gives more weight to the most recent time periods. It is described recursively as

$$\mathrm{EMA}(i,m) = p \times t(i) + (1-p) \times \mathrm{EMA}\,(i-1, m-1)$$
$$\mathrm{EMA}\,(i,1) = t(i)$$

where p is a value between 0 and 1. For example, if $p = 0.5$, the most recent value $t(i)$ is equally weighted with the computation for all previous values in the window, where the computation begins with averaging the first two values in the series. The computation starts with the following two equations:

$$\mathrm{EMA}(i,2) = 0.5\,t(i) + 0.5\,t(i-1)$$
$$\mathrm{EMA}(i,3) = 0.5\,t(i) + 0.5\,[0.5\,t(i-1) + 0.5\,t(i-2)]$$

As usual, application knowledge or empirical validation determines the value of p. The EMA has performed very well for many business-related applications, usually producing results superior to the MA.

A moving average summarizes the recent past, but spotting a change in the trend of the data may additionally improve forecasting performances. Characteristics of a trend can be measured by composing features that compare recent measurements to those of the more distant past. Three simple comparative features are:

1. $t(i) - \mathrm{MA}(i, m)$, the difference between the current value and an MA,
2. $\mathrm{MA}(i, m) - \mathrm{MA}(i - k, m)$, the difference between two MA, usually of the same window size, and
3. $t(i)/\mathrm{MA}(i, m)$, the ratio between the current value and an MA, which may be preferable for some applications.

In general, the main components of the summarizing features for a time series are:

1. current values,
2. smoothed values using MA, and
3. derived trends, differences, and ratios.

(a)

Time	a	b
1	5	117
2	8	113
3	4	116
4	9	118
5	10	119
6	12	120

(b)

Sample	$a(n-2)$	$a(n-1)$	$a(n)$	$b(n-2)$	$b(n-1)$	$b(n)$
1	5	8	4	117	113	116
2	8	4	9	113	116	118
3	4	9	8	116	118	119
4	9	10	12	118	119	120

Figure 2.5. Tabulation of time-dependent features a and b. (a) Initial time-dependent data. (b) Samples prepared for data mining with time window = 3.

The immediate extension of a univariate time series is to a multivariate one. Instead of having a single measured value at time i, $t(i)$, multiple measurements $t[a(i), b(j)]$ are taken at the same time. There are no extra steps in data preparation for the multivariate time series. Each series can be transformed into features, and the values of the features at each distinct time $A(i)$ are merged into a sample i. The resultant transformations yield a standard tabular form of data such as the table given in Figure 2.5.

While some data-mining problems are characterized by a single time series, hybrid applications are more frequent in real-world problems, having both time series and features that are not dependent on time. In these cases, standard procedures for time-dependent transformation and summarization of attributes are performed. High dimensions of data generated during these transformations can be reduced through the next phase of a data-mining process: data reduction.

Some data sets do not include a time component explicitly, but the entire analysis is performed in the time domain (typically based on several dates that are attributes of described entities). One very important class of data belonging to this type is *survival data*. Survival data are data concerning how long it takes for a particular event to happen. In many medical applications the event is the death of a patient, and, therefore, we analyze the patient's survival time. In industrial applications, the event is often the failure of a component in a machine. Thus, the output in these sorts of problems is the survival time. The inputs are the patient's records in medical applications and characteristics of the machine component in industrial applications. There are two main characteristics of survival data that make them different from the data in other data-mining problems. The first characteristic is called *censoring*. In many studies, the event has not happened by the end of the study period. So, for some patients in a medical trial, we might know that the patient was still alive after 5 years, but we do not know when the patient died. This sort of observation would be called a censored observation. If the output is censored, we do not know the value of the output, but we do have some information about it. The second characteristic of survival data is that the input values are *time dependent*. Since collecting data entails waiting until the event happens, it is possible for the inputs to change its value during the waiting period. If a patient stops smoking or starts with a new drug during the study, it is

important to know what data to include into the study and how to represent these changes in time. Data-mining analysis for these types of problems concentrates on the survivor function or the hazard function. The survivor function is the probability of the survival time being greater than the time t. The hazard function indicates how likely a failure (of the industrial component) is at time t, given that a failure has not occurred before time t.

2.6 OUTLIER ANALYSIS

Very often, in large data sets, there exist samples that do not comply with the general behavior of the data model. Such samples, which are significantly different or inconsistent with the remaining set of data, are called outliers. Outliers can be caused by measurement error or they may be the result of inherent data variability. If the display of a person's age in the database is −1, the value is obviously not correct, and the error could have been caused by a default setting of the field "unrecorded age" in the computer program. On the other hand, if, in the database, the number of children for one person is 25, this datum is unusual and has to be checked. The value could be a typographical error, or it could be correct and represent real variability for the given attribute.

Many data-mining algorithms try to minimize the influence of outliers on the final model or to eliminate them in the preprocessing phases. Outlier detection methodologies have been used to detect and, where appropriate, remove anomalous observations from data. Outliers arise due to mechanical faults, changes in system behavior, fraudulent behavior, human error, or instrument error or simply through natural deviations in populations. Their detection can identify system faults and fraud before they escalate with potentially catastrophic consequences. The literature describes the field with various names including outlier detection, novelty detection, anomaly detection, noise detection, deviation detection, or exception mining. Efficient detection of such outliers reduces the risk of making poor decisions based on erroneous data and aids in identifying, preventing, and repairing the effects of malicious or faulty behavior. Additionally, many data-mining techniques may not work well in the presence of outliers. Outliers may introduce skewed distributions or complexity into models of the data, which may make it difficult, if not impossible, to fit an accurate model to the data in a computationally feasible manner.

The data-mining analyst has to be very careful in the automatic elimination of outliers because, if the data are correct, that could result in the loss of important hidden information. Some data-mining applications are focused on outlier detection, and it is the essential result of a data analysis. The process consists of two main steps: (1) build a profile of the "normal" behavior and (2) use the "normal" profile to detect outliers. Profile can be patterns or summary statistics for the overall population. The assumption is that there are considerably more "normal" observations than "abnormal"—outliers/anomalies in the data. For example, while detecting fraudulent credit card transactions in a bank, the outliers are typical examples that may indicate

fraudulent activity, and the entire data-mining process is concentrated on their detection. But in many of the other data-mining applications, especially if they are supported with large data sets, outliers are not very useful, and they are more the result of errors in data collection than a characteristic of a data set.

Outlier detection and potential removal from a data set can be described as a process of the selection of k out of n samples that are considerably dissimilar, exceptional, or inconsistent with respect to the remaining data ($k \ll n$). The problem of defining outliers is nontrivial, especially in multidimensional samples. Main types of outlier detection schemes are:

- graphical or visualization techniques,
- statistical-based techniques,
- distance-based techniques, and
- model-based techniques.

Examples of visualization methods include boxplot (1D), scatter plot (2D), and spin plot (3D), and they will be explained in the following chapters. Data-visualization methods that are useful in outlier detection for data with one to three dimensions are much weaker in multidimensional data because of a lack of adequate visualization methodologies for *n*-dimensional spaces. An illustrative example of a visualization of 2D samples and visual detection of outliers is given in Figures 2.6 and 2.7. The main limitations of the approach are time-consuming process and subjective nature of outlier detection.

Statistical-based outlier detection methods can be divided between *univariate methods*, proposed in earlier works in this field, and *multivariate methods* that usually form most of the current body of research. Statistical methods either assume a known underlying distribution of the observations or, at least, are based on statistical estimates of unknown distribution parameters. These methods flag as outliers those observations that deviate from the model assumptions. The approach is often unsuitable for high-dimensional data sets and for arbitrary data sets without prior knowledge of the underlying data distribution.

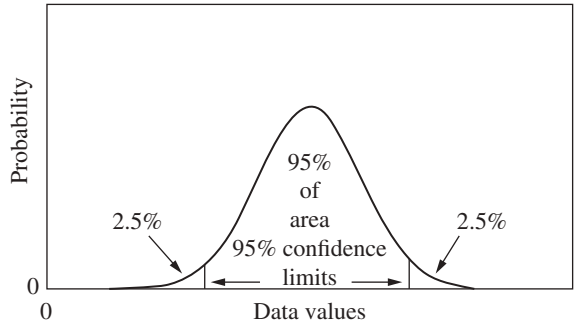


Figure 2.6. Outliers for univariate data based on mean value and standard deviation.

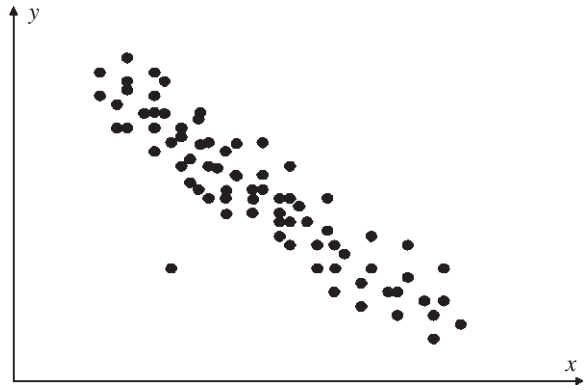


Figure 2.7. Two-dimensional data set with one outlying sample.

Most of the earliest univariate methods for outlier detection rely on the assumption of an underlying known distribution of the data, which is assumed to be identically and independently distributed. Moreover, many discordance tests for detecting univariate outliers further assume that the distribution parameters and the type of expected outliers are also known. Although traditionally the normal distribution has been used as the target distribution, this definition can be easily extended to any unimodal symmetric distribution with positive density function. Traditionally, the sample mean and the sample variance give good estimation for data location and data shape if it is not contaminated by outliers. When the database is contaminated, those parameters may deviate and significantly affect the outlier detection performance. Needless to say, in real-world data-mining applications, these assumptions are often violated.

The simplest approach to outlier detection for 1D samples is based on traditional unimodal statistics. Assuming that the distribution of values is given, it is necessary to find basic statistical parameters such as mean value and variance. Based on these values and the expected (or predicted) number of outliers, it is possible to establish the threshold value as a function of variance. All samples out of the threshold value are candidates for outliers as it is represented in Figure 2.6. The main problem with this simple methodology is an a priori assumption about data distribution. In most real-world examples, the data distribution may not be known.

For example, if the given data set represents the feature *Age* with twenty different values

$$Age = \{3, 56, 23, 39, 156, 52, 41, 22, 9, 28, 139, 31, 55, 20, -67, 37, 11, 55, 45, 37\}$$

then the corresponding statistical parameters are

$$\text{Mean} = 39.9$$

$$\text{Standard deviation} = 45.65$$

If we select the threshold value for normal distribution of data as

$$\text{Threshold} = \text{Mean} \pm 2 \times \text{Standard deviation}$$

then all data that are out of range [−54.1, 131.2] will be potential outliers. Additional knowledge of the characteristics of the feature (Age is always greater than zero) may further reduce the range to [0, 131.2]. In our example there are three values that are outliers based on the given criteria: 156, 139, and −67. With a high probability, we can conclude that all three of them are typo errors (data entered with additional digits or an additional "−" sign).

An additional single-dimensional method is Grubbs' method (extreme Studentized deviate), which calculates a Z value as the difference between the mean value for the attribute and the analyzed value divided by the standard deviation for the attribute. The Z value is compared with a 1 or 5% significance level showing an outlier if the Z parameter is above the threshold value.

In many cases multivariable observations cannot be detected as outliers when each variable is considered independently. Outlier detection is possible only when multivariate analysis is performed and the interactions among different variables are compared within the class of data. Illustrative example is given in Figure 2.7 where analysis of each dimension separately will not give any outlier, while analysis of 2D samples (x, y) gives one outlier detectable even through visual inspection.

Statistical methods for multivariate outlier detection often indicate those samples that are located relatively far from the center of the data distribution. Several distance measures can be implemented for such a task. The Mahalanobis distance measure includes the inter-attribute dependencies so the system can compare attribute combinations. Therefore, it is a well-known approach that depends on estimated parameters of the multivariate distribution. Given n observations x_i from a p-dimensional data set (often $n \gg p$), denote the sample mean vector by x_n and the sample covariance matrix by V_n, where

$$\mathrm{V}_n = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_n)(x_i - \bar{x}_n)^T$$

The *Mahalanobis* distance for each multivariate data point i ($i = 1,\ldots,n$) is denoted by M_i and given by

$$M_i = \left(\sum_{i=1}^{n}(x_i - \bar{x}_n)^T \mathrm{V}_n^{-1}(x_i - \bar{x}_n)\right)^{1/2}$$

Accordingly, those n-dimensional samples with a large *Mahalanobis* distance are indicated as outliers. Many statistical methods require data-specific parameters representing a priori data knowledge. Such information is often not available or is expensive to compute. Also, most real-world data sets simply do not follow one specific distribution model.

Distance-based techniques for outlier detection are simple to implement and make no prior assumptions about the data distribution model. However, they suffer exponential computational growth as they are founded on the calculation of the distances between all samples. The computational complexity is dependent on both the dimensionality of the data set m and the number of samples n and usually is expressed as $O(n^2m)$. Hence, it is not an adequate approach to use with very large data sets. Moreover, this definition can lead to problems when the data set has both dense and sparse regions. For example, as the dimensionality increases, the data points are spread through a larger volume and become less dense. This makes the convex hull harder to discern and is known as the "*curse of dimensionality*."

Distance-based outlier detection method, presented in this section, eliminates some of the limitations imposed by the statistical approach. The most important difference is that this method is applicable to multidimensional samples while most of statistical descriptors analyze only a single dimension, or several dimensions, but separately. The basic computational complexity of this method is the evaluation of distance measures between all samples in an n-dimensional data set. Then, a sample s_i in a data set S is an outlier if at least a fraction p of the samples in S lies at a distance greater than d. In other words, distance-based outliers are those samples that do not have enough neighbors, where neighbors are defined through the multidimensional distance between samples. Obviously, the criterion for outlier detection is based on two parameters, p and d, which may be given in advance using knowledge about the data or which may be changed during the iterations (trial-and-error approach) to select the most representative outliers.

To illustrate the approach we can analyze a set of 2D samples S, where the requirements for outliers are the values of thresholds, $p \geq 4$ and $d \geq$:

$$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\} = \{(2,4), (3,2), (1,1), (4,3), (1,6), (5,3), (4,2)\}$$

The table of Euclidean distances, $d = [(x1 - x2)^2 + (y1 - y2)^2]^{1/2}$, for the set S is given in Table 2.3, and, based on this table, we can calculate a value for the parameter p with the given threshold distance ($d = 3$) for each sample. The results are represented in Table 2.4.

TABLE 2.3. Table of Distances for Data Set S

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1		2.236	3.162	2.236	2.236	3.162	2.828
s_2			2.236	1.414	4.472	2.236	1.000
s_3				3.605	5.000	4.472	3.162
s_4					4.242	1.000	1.000
s_5						5.000	5.000
s_6							1.414

TABLE 2.4. Number of Samples p with the Distance Greater Than d from a Given Sample

Sample	p
s_1	2
s_2	1
s_3	5
s_4	2
s_5	5
s_6	3
s_7	2

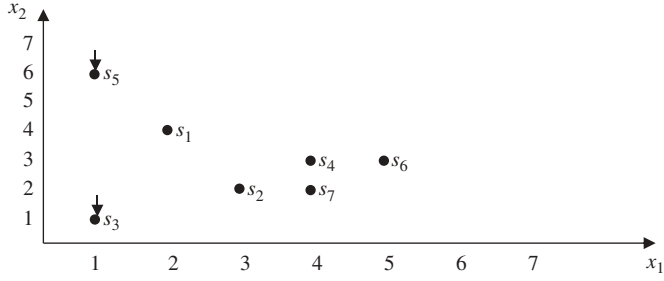


Figure 2.8. Visualization of two-dimensional data set for outlier detection.

Using the results of the applied procedure and given threshold values, it is possible to select samples s_3 and s_5 as outliers (because their values for p is above the threshold value: $p = 4$). The same results could be obtained by visual inspection of a data set, represented in Figure 2.8. Of course, the given data set is very small, and a 2D graphical representation is possible and useful. For n-dimensional, real-world data analyses, the visualization process is much more difficult, and analytical approaches in outlier detection are often more practical and reliable.

There is a possibility for reducing complexity of the algorithm by partitioning the data into n-dimensional cells. If any cell and its directly adjacent neighbors contain more than k points, then the points in the cell are deemed to lie in a dense area of the distribution, so the points contained are unlikely to be outliers. If the number of points is less than k, then all points in the cell are potential outliers. Hence, only a small number of cells need to be processed, and only a relatively small number of distances need to be calculated for outlier detection.

Model-based techniques are the third class of outlier detection methods. These techniques simulate the way in which humans can distinguish unusual samples from a set of other similar samples. These methods define the basic characteristics of the

sample set, and all samples that deviate from these characteristics are outliers. The sequential-exception technique is one possible approach that is based on a dissimilarity function. For a given set of n samples, a possible dissimilarity function is the total variance of the sample set. Now, the task is to define the smallest subset of samples whose removal results in the greatest reduction of the dissimilarity function for the residual set. The general task of finding outliers using this method can be very complex (combinational explosion of different selections of the set of potential outliers—the so-called exception set), and it can be theoretically defined as an NP-hard problem (nondeterministic polynomial time, i.e., intractable). If we settle for a less-than-optimal answer, the algorithm's complexity can be reduced to the linear level using a sequential approach. Using the greedy method, the algorithm reduces the size sequentially, sample by sample (or subset by subset), by selecting at each step the one that causes the greatest decrease in the total variance.

Many data-mining algorithms are robust and as such tolerant to outliers but were specifically optimized for clustering or classification in large data sets. It includes clustering algorithms such as BIRCH and DBSCAN, kNN classification algorithms, and different neural networks. These methodologies are explained with more details later in the book, but the reader has to be aware about applicability of these techniques as powerful tools for outlier detection. For example, in data set represented in Figure 2.9, clustering-based methods consider a cluster of small sizes, including the size of one sample, as clustered outliers. Note that since their main objective is clustering, these methods are not always optimized for outlier detection. In most cases, the outlier detection criteria are implicit and cannot easily be inferred from the clustering procedures.

Most of outlier detection techniques have only focused on continuous real-valued data attributes, and there has been little focus on categorical data. Most approaches require cardinal or at the least ordinal data to allow vector distances to be calculated and have no mechanism for processing categorical data with no implicit ordering.

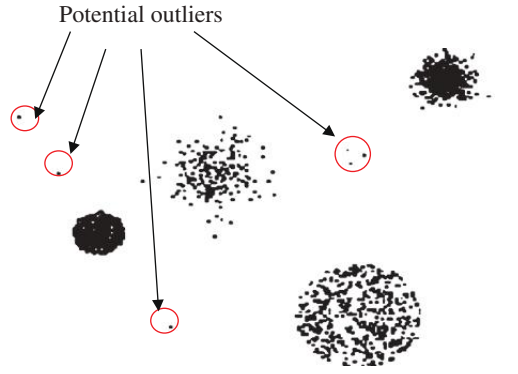


Figure 2.9. Determining outliers through clustering.

computer-based tool to match the presented problem and automatically obtaining a solution. This is a misconception based on an artificial idealization of the world. There are several reasons why this is incorrect. One reason is that data mining is not simply a collection of isolated tools, each completely different from the other and waiting to be matched to the problem. A second reason lies in the notion of matching a problem to a technique. Only very rarely is a research question stated sufficiently precisely that a single and simple application of the method will suffice. In fact, what happens in practice is that data mining becomes an iterative process. One studies the data, examines it using some analytic technique, decides to look at it another way, perhaps modifies it, and then goes back to the beginning and applies another data-analysis tool, reaching either better or different results. This can go round and round many times; each technique is used to probe slightly different aspects of data—to ask a slightly different question of the data. What is essentially being described here is a voyage of discovery that makes modern data mining exciting. Still, data mining is not a random application of statistical, machine learning, and other methods and tools. It is not a random walk through the space of analytic techniques but a carefully planned and considered process of deciding what will be most useful, promising, and revealing.

It is important to realize that the problem of discovering or estimating dependencies from data or discovering totally new data is only one part of the general experimental procedure used by scientists, engineers, and others who apply standard steps to draw conclusions from the data. The general experimental procedure adapted to data-mining problems involves the following steps:

1. **State the problem and formulate the hypothesis**

   Most data-based modeling studies are performed in a particular application domain. Hence, domain-specific knowledge and experience are usually necessary in order to come up with a meaningful problem statement. Unfortunately, many application studies tend to focus on the data-mining technique at the expense of a clear problem statement. In this step, a modeler usually specifies a set of variables for the unknown dependency and, if possible, a general form of this dependency as an initial hypothesis. There may be several hypotheses formulated for a single problem at this stage. The first step requires the combined expertise of an application domain and a data-mining model. In practice, it usually means a close interaction between the data-mining expert and the application expert. In successful data-mining applications, this cooperation does not stop in the initial phase; it continues during the entire data-mining process.

2. **Collect the data**

   This step is concerned with how the data are generated and collected. In general, there are two distinct possibilities. The first is when the data-generation process is under the control of an expert (modeler): this approach is known as a *designed experiment*. The second possibility is when the expert cannot influence the data-generation process: this is known as the *observational approach*. An observational setting, namely, random data generation, is assumed in most data-mining

applications. Typically, the sampling distribution is completely unknown after data are collected, or it is partially and implicitly given in the data-collection procedure. It is very important, however, to understand how data collection affects its theoretical distribution, since such a priori knowledge can be very useful for modeling and, later, for the final interpretation of results. Also, it is important to make sure that the data used for estimating a model and the data used later for testing and applying a model come from the same, unknown, sampling distribution. If this is not the case, the estimated model cannot be successfully used in a final application of the results.

3. **Preprocessing the data**

In the observational setting, data are usually "collected" from the existing databases, data warehouses, and data marts. Data preprocessing usually includes at least two common tasks:

(a) *Outlier detection (and removal)*

Outliers are unusual data values that are not consistent with most observations. Commonly, outliers result from measurement errors and coding and recording errors and, sometimes, are natural, abnormal values. Such nonrepresentative samples can seriously affect the model produced later. There are two strategies for dealing with outliers:

(i) Detect and eventually remove outliers as a part of the preprocessing phase.

(ii) Develop robust modeling methods that are insensitive to outliers.

(b) *Scaling, encoding, and selecting features*

Data preprocessing includes several steps such as variable scaling and different types of encoding. For example, one feature with the range [0, 1] and the other with the range [–100, 1000] will not have the same weight in the applied technique; they will also influence the final data-mining results differently. Therefore, it is recommended to scale them and bring both features to the same weight for further analysis. Also, application-specific encoding methods usually achieve dimensionality reduction by providing a smaller number of informative features for subsequent data modeling.

These two classes of preprocessing tasks are only illustrative examples of a large spectrum of preprocessing activities in a data-mining process.

Data-preprocessing steps should not be considered completely independent from other data-mining phases. In every iteration of the data-mining process, all activities, together, could define new and improved data sets for subsequent iterations. Generally, a good preprocessing method provides an optimal representation for a data-mining technique by incorporating a priori knowledge in the form of application-specific scaling and encoding. More about these techniques and the preprocessing phase in general will be given in Chapters 2 and 3, where we have functionally divided preprocessing and its corresponding techniques into two subphases: data preparation and data-dimensionality reduction.

4. **Estimate the model**

   The selection and implementation of the appropriate data-mining technique is the main task in this phase. This process is not straightforward; usually, in practice, the implementation is based on several models, and selecting the best one is an additional task. The basic principles of learning and discovery from data are given in Chapter 4 of this book. Later, Chapters 5 through 13 explain and analyze specific techniques that are applied to perform a successful learning process from data and to develop an appropriate model.

5. **Interpret the model and draw conclusions**

   In most cases, data-mining models should help in decision-making. Hence, such models need to be interpretable in order to be useful because humans are not likely to base their decisions on complex "black-box" models. Note that the goals of accuracy of the model and accuracy of its interpretation are somewhat contradictory. Usually, simple models are more interpretable, but they are also less accurate. Modern data-mining methods are expected to yield highly accurate results using high-dimensional models. The problem of interpreting these models, also very important, is considered a separate task, with specific techniques to validate the results. A user does not want hundreds of pages of numerical results. He does not understand them; he cannot summarize, interpret, and use them for successful decision-making.

Even though the focus of this book is on steps 3 and 4 in the data-mining process, we have to understand that they are just two steps in a more complex process. All phases, separately, and the entire data-mining process, as a whole, are highly iterative, as has been shown in Figure 1.2. A good understanding of the whole process is important for any successful application. No matter how powerful the data-mining method used in step 4 is, the resulting model will not be valid if the data are not collected and preprocessed correctly or if the problem formulation is not meaningful.

In 1999, several large companies including automaker Daimler-Benz, insurance provider OHRA, hardware and software manufacturer NCR Corp., and statistical software maker SPSS, Inc. formalize and standardize an approach to data-mining process. The result of their work was CRISP-DM, the CRoss-Industry Standard Process for Data Mining presented on Figure 1.3. The process was designed to be independent of any specific tool. The CRISP-DM methodology provides a structured approach in planning a data-mining project. Numerous data-mining applications showed its practicality and flexibility and its usefulness when using analytics to solve complex business issues. This model is an idealized sequence of events. In practice, many of the tasks can be performed in a different order, and it will often be necessary to backtrack to previous activities and repeat certain actions. The model does not try to capture all possible routes through the data-mining process. The reader may recognize the connection and similarities between steps of data mining presented on Figures 1.2 and 1.3.

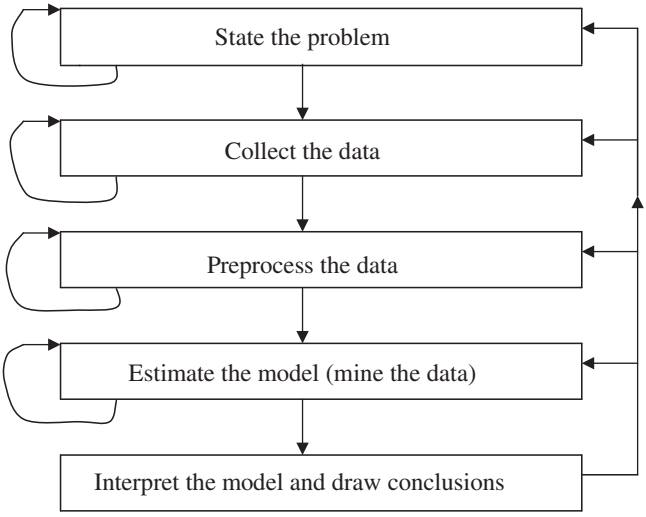


Figure 1.2. The data-mining process.

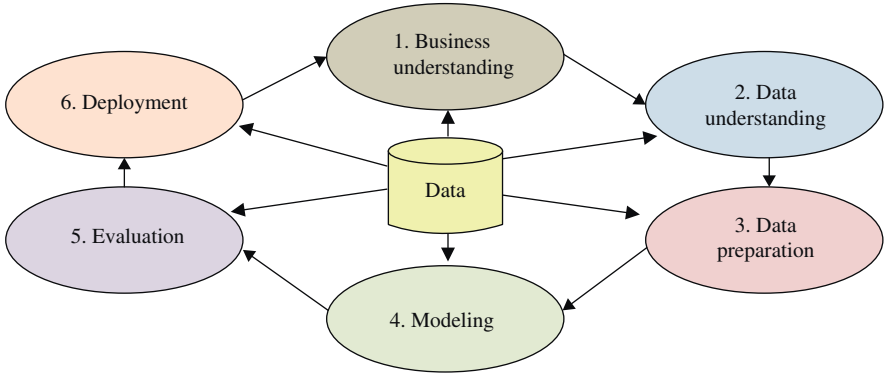


Figure 1.3. CRISP-DM conceptual model.

1.4 FROM DATA COLLECTION TO DATA PREPROCESSING

As we enter into the age of digital information, the problem of data overload looms ominously ahead. Our ability to analyze and understand massive *data sets* is far behind our ability to gather and store the data. Recent advances in computing, communications, and digital storage technologies, together with the development of high-throughput data-acquisition technologies, have made it possible to gather and store incredible volumes of data. Large databases of digital information are ubiquitous.

For small or moderate data sets, the preprocessing steps mentioned in the previous chapter in preparation for data mining are usually enough. For really large data sets, there is an increased likelihood that an intermediate, additional step—data reduction—should be performed prior to applying the data-mining techniques. While large data sets have the potential for better mining results, there is no guarantee that they will yield better knowledge than small data sets. Given multidimensional data, a central question is whether it can be determined, prior to searching for all data-mining solutions in all dimensions, that the method has exhausted its potential for mining and discovery in a reduced data set. More commonly, a general solution may be deduced from a subset of available features or cases, and it will remain the same even when the search space is enlarged.

The main theme for simplifying the data in this step is dimension reduction, and the main question is whether some of these prepared and preprocessed data can be discarded without sacrificing the quality of results. There is one additional question about techniques for data reduction: Can the prepared data be reviewed and a subset found in a reasonable amount of time and space? If the complexity of algorithms for data reduction increases exponentially, then there is little to gain in reducing dimensions in big data. In this chapter, we will present basic and relatively efficient techniques for dimension reduction applicable to different data-mining problems.

3.1 DIMENSIONS OF LARGE DATA SETS

The choice of data representation and selection, reduction, or transformation of features is probably the most important issue that determines the quality of a data-mining solution. Besides influencing the nature of a data-mining algorithm, it can determine whether the problem is solvable at all or how powerful the resulting model of data mining is. A large number of features can make available samples of data relatively insufficient for mining. In practice, the number of features can be as many as several hundreds. If we have only a few hundred samples for analysis, dimensionality reduction is required in order for any reliable model to be mined or to be of any practical use. On the other hand, data overload, because of high dimensionality, can make some data-mining algorithms non-applicable, and the only solution is again a reduction of data dimensions. For example, a typical classification task is to separate healthy patients from cancer patients, based on their gene expression "profile." Usually fewer than 100 samples (patients' records) are available altogether for training and testing. But the number of features in the raw data ranges from 6,000 to 60,000. Some initial filtering usually brings the number of features to a few thousand; still it is a huge number and additional reduction is necessary. The three main dimensions of preprocessed data sets, usually represented in the form of flat files, are *columns* (features), *rows* (cases or samples), and *values* of the features.

Therefore, the three basic operations in a data-reduction process are delete a column, delete a row, and reduce the number of values in a column (smooth a feature). These operations attempt to preserve the character of the original data by deleting data

that are nonessential. There are other operations that reduce dimensions, but the new data are unrecognizable when compared with the original data set, and these operations are mentioned here just briefly because they are highly application dependent. One approach is the replacement of a set of initial features with a new composite feature. For example, if samples in a data set have two features, person's height and person's weight, it is possible for some applications in the medical domain to replace these two features with only one body mass index, which is proportional to the quotient of the initial two features. Final reduction of data does not reduce the quality of results; in some applications, the results of data mining are even improved.

Performing standard data-reduction operations (deleting rows, columns, or values) as a preparation for data mining, we need to know what we gain and/or lose with these activities. The overall comparison involves the following parameters for analysis:

1. *Computing time*—Simpler data, a result of the data-reduction process, can hopefully lead to a reduction in the time taken for data mining. In most cases, we cannot afford to spend too much time on the data-preprocessing phases, including a reduction of data dimensions, although the more time we spend in preparation the better the outcome.
2. *Predictive/descriptive accuracy*—This is the dominant measure for most data-mining models since it measures how well the data is summarized and generalized into the model. We generally expect that by using only relevant features, a data-mining algorithm can not only learn faster but also with higher accuracy. Irrelevant data may mislead a learning process and a final model, while redundant data may complicate the task of learning and cause unexpected data-mining results.
3. *Representation of the data-mining model*—The simplicity of representation, obtained usually with data reduction, often implies that a model can be better understood. The simplicity of the induced model and other results depends on its representation. Therefore, if the simplicity of representation improves, a relatively small decrease in accuracy may be tolerable. The need for a balanced view between accuracy and simplicity is necessary, and dimensionality reduction is one of the mechanisms for obtaining this balance.

It would be ideal if we could achieve reduced time, improved accuracy, and simplified representation at the same time, using dimensionality reduction. More often, however, we gain in some and lose in others and balance between them according to the application at hand. It is well known that no single data-reduction method can be best suited for all applications. A decision about method selection is based on available knowledge about an application (relevant data, noise data, metadata, correlated features) and required time constraints for the final solution.

Algorithms that perform all basic operations for data reduction are not simple, especially when they are applied to large data sets. Therefore, it is useful to enumerate the desired properties of these algorithms before giving their detailed descriptions.

Recommended characteristics of data-reduction algorithms that may be guidelines for designers of these techniques are as follows:

1. *Measurable quality*—The quality of approximated results using a reduced data set can be determined precisely.
2. *Recognizable quality*—The quality of approximated results can be easily determined at run time of the data-reduction algorithm, before application of any data-mining procedure.
3. *Monotonicity*—The algorithms are usually iterative, and the quality of results is a nondecreasing function of time and input data quality.
4. *Consistency*—The quality of results is correlated with computation time and input data quality.
5. *Diminishing returns*—The improvement in the solution is large in the early stages (iterations) of the computation, and it diminishes over time.
6. *Interruptability*—The algorithm can be stopped at any time and provide some answers.
7. *Preemptability*—The algorithm can be suspended and resumed with minimal overhead.

3.2 FEATURES REDUCTION

Most of the real-world data-mining applications are characterized by high-dimensional data, where not all of the features are important. For example, high-dimensional data (i.e., data sets with hundreds or even thousands of features) can contain a lot of irrelevant, noisy information that may greatly degrade the performance of data-mining process. Even state-of-art data-mining algorithms cannot overcome the presence of large number of weakly relevant and redundant features. This is usually attributed to the "curse of dimensionality" or to the fact that irrelevant features decrease the signal-to-noise ratio. In addition, many algorithms become computationally intractable when the dimensionality is high.

Data such as images, text, and multimedia are high dimensional in nature, and this dimensionality of data poses a challenge to data-mining tasks. Researchers have found that reducing the dimensionality of data results in a faster computation while maintaining reasonable accuracy. In the presence of many irrelevant features, mining algorithms tend to overfit the model. Therefore, many features can be removed without performance deterioration in the mining process.

When we are talking about data quality and improved performances of reduced data sets, we can see that this issue is not only about noisy or contaminated data (problems mainly solved in the preprocessing phase) but also about irrelevant, correlated, and redundant data. Recall that data with corresponding features are not usually collected solely for data-mining purposes. Therefore, dealing with relevant features alone can be far more effective and efficient. Basically, we want to choose features that are

Recommended characteristics of data-reduction algorithms that may be guidelines for designers of these techniques are as follows:

1. *Measurable quality*—The quality of approximated results using a reduced data set can be determined precisely.
2. *Recognizable quality*—The quality of approximated results can be easily determined at run time of the data-reduction algorithm, before application of any data-mining procedure.
3. *Monotonicity*—The algorithms are usually iterative, and the quality of results is a nondecreasing function of time and input data quality.
4. *Consistency*—The quality of results is correlated with computation time and input data quality.
5. *Diminishing returns*—The improvement in the solution is large in the early stages (iterations) of the computation, and it diminishes over time.
6. *Interruptability*—The algorithm can be stopped at any time and provide some answers.
7. *Preemptability*—The algorithm can be suspended and resumed with minimal overhead.

3.2 FEATURES REDUCTION

Most of the real-world data-mining applications are characterized by high-dimensional data, where not all of the features are important. For example, high-dimensional data (i.e., data sets with hundreds or even thousands of features) can contain a lot of irrelevant, noisy information that may greatly degrade the performance of data-mining process. Even state-of-art data-mining algorithms cannot overcome the presence of large number of weakly relevant and redundant features. This is usually attributed to the "curse of dimensionality" or to the fact that irrelevant features decrease the signal-to-noise ratio. In addition, many algorithms become computationally intractable when the dimensionality is high.

Data such as images, text, and multimedia are high dimensional in nature, and this dimensionality of data poses a challenge to data-mining tasks. Researchers have found that reducing the dimensionality of data results in a faster computation while maintaining reasonable accuracy. In the presence of many irrelevant features, mining algorithms tend to overfit the model. Therefore, many features can be removed without performance deterioration in the mining process.

When we are talking about data quality and improved performances of reduced data sets, we can see that this issue is not only about noisy or contaminated data (problems mainly solved in the preprocessing phase) but also about irrelevant, correlated, and redundant data. Recall that data with corresponding features are not usually collected solely for data-mining purposes. Therefore, dealing with relevant features alone can be far more effective and efficient. Basically, we want to choose features that are

relevant to our data-mining application in order to achieve maximum performance with the minimum measurement and processing effort. A feature-reduction process should result in:

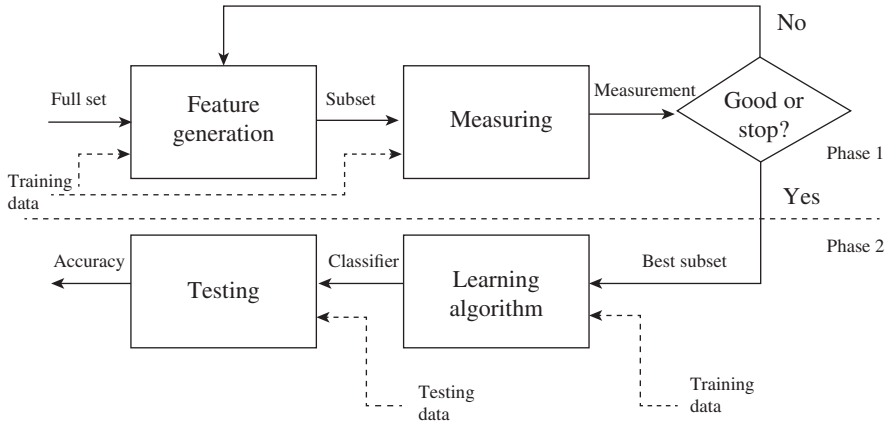
1. less data so that the data-mining algorithm can learn faster,
2. higher accuracy of a data-mining process so that the model can generalize better from data,
3. simple results of a data-mining process so that they are easier to understand and use, and
4. fewer features so that in the next round of data collection, a saving can be made by removing redundant or irrelevant features.

Let us start our detailed analysis of possible column-reduction techniques, where features are eliminated from the data set based on a given criterion. To address the curse of dimensionality, dimensionality reduction techniques are proposed as a data-preprocessing step. This process identifies a suitable low-dimensional representation of original data. Reducing the dimensionality improves the computational efficiency and accuracy of the data analysis. Also, it improves comprehensibility of a data-mining model. Proposed techniques are classified as supervised and unsupervised techniques based on the type of a learning process. Supervised algorithms need a training set with the output class label information to learn the lower-dimensional representation according to some criteria. Unsupervised approaches project the original data to a new lower-dimensional space without utilizing the label (class) information. Dimensionality reduction techniques function either by transforming the existing features to a new reduced set of features or by selecting a subset of the existing features. Therefore, two standard tasks are associated with producing a reduced set of features, and they are classified as follows:

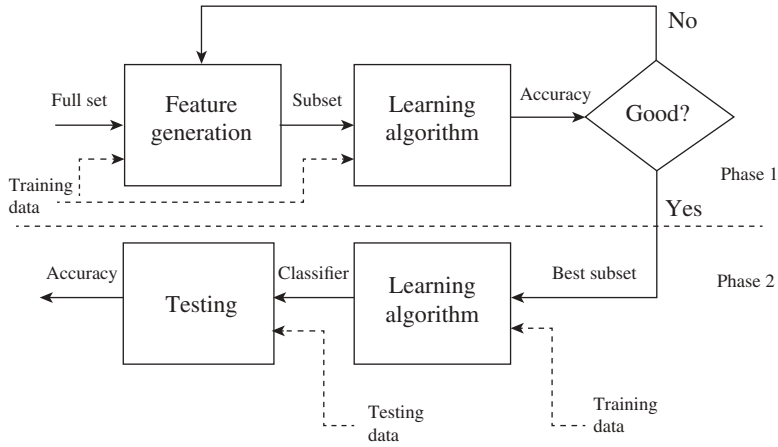
1. *Feature selection*—Based on the knowledge of the application domain and the goals of the mining effort, the human analyst may select a subset of the features found in the initial data set. The process of feature selection can be manual or supported by some automated procedures.

   Roughly speaking, feature selection methods are applied in one of the three conceptual frameworks: the filter model, the wrapper model, and embedded methods. These three basic families differ in how the learning algorithm is incorporated in evaluating and selecting features. In the filter model the selection of features is done as a preprocessing activity, without trying to optimize the performance of any specific data-mining technique directly. This is usually achieved through an (ad hoc) evaluation function using a search method in order to select a subset of features that maximizes this function. Performing an exhaustive search is usually intractable due to the large number of initial features. Therefore, different methods may apply a variety of search heuristics. Wrapper methods select features by "wrapping" the search around the selected

learning algorithm and evaluate feature subsets based on the learning performance of the data-mining technique for each candidate feature subset. The main drawback of this approach is its computational complexity. Main characteristics of both approaches are given in Figure 3.1. Finally, embedded methods incorporate feature search and the learning algorithm into a single optimization problem formulation. When the number of samples and dimensions becomes very large, the filter approach is usually a choice due to its computational efficiency and neutral bias toward any learning methodology.



Filter model



Wrapper model

Figure 3.1. Feature selection approaches.

2. *Feature extraction/transformation*—There are transformations of data that can have a surprisingly strong impact on the results of data-mining methods. In this sense, the composition and/or transformation of features is a greater determining factor in the quality of data-mining results. In most instances, feature composition is dependent on knowledge of the application, and an interdisciplinary approach to feature composition tasks produces significant improvements in the preparation of data. Still, some general-purpose techniques, such as principal component analysis (PCA), are often used and with a high success.

   Feature selection is typically preferred over extraction/transformation when one wishes to keep the original meaning of the features and wishes to determine which of those features are important. Moreover, once features are selected, only these features need to be calculated or collected, whereas, in transformation-based methods all input features are still needed to obtain the reduced dimension.

3.2.1 Feature Selection

In data mining, feature selection, also known as variable selection, feature reduction, attribute selection, or variable subset selection, is a set of techniques, which selects a subset of relevant features for building robust learning models by removing most irrelevant and redundant features from the data. The objective of feature selection is threefold: improving the performance of a data mining model, providing faster and more cost-effective learning process, and providing a better understanding of the underlying process that generates the data. Feature selection algorithms typically fall into two categories: feature ranking and subset selection. Feature ranking ranks all features by a specified metric and eliminates all features that do not achieve an adequate score. Subset selection, on the other hand, searches the set of all features for the optimal subset where features in the selected subset are not ranked. We have to be aware that different feature selection methods may give different reduced features sets.

In the feature ranking algorithm, one can expect a ranked list of features that are ordered according to a specific evaluation measure. A measure can be based on accuracy of available data, consistency, information content, distances between samples, and, finally, statistical dependencies between features. These algorithms do not tell you what the minimum set of features for further analysis is; they indicate only the relevance of a feature compared with others. Minimum subset algorithms, on the other hand, return a minimum feature subset, and no differences are made among features in the subset—all have the same ranking. The features in the subset are relevant for the mining process; the others are irrelevant. In both types of algorithms, it is important to establish a feature-evaluation scheme: the way in which the features are evaluated and then ranked or added to the selected subset.

Feature selection in general can be viewed as a search problem, with each state in the search space specifying a subset of the possible features. If, for example, a data set has three features $\{A_1, A_2, A_3\}$, and in the process of selecting features, the presence of

a feature is coded with 1 and its absence with 0, then there should be a total of 2^3 reduced feature subsets coded with {0, 0, 0}, {1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {1, 1, 0}, {1, 0, 1}, {0, 1, 1}, and {1, 1, 1}. The problem of feature selection is relatively trivial if the search space is small, since we can analyze all subsets in any order and a search will get completed in a short time. However, the search space is usually not small. It is 2^N where the number of dimensions N in typical data-mining applications is large ($N > 20$). This makes the starting point and the search strategy very important. An exhaustive search of all subsets of features very often is replaced with some heuristic search procedures. Using knowledge of the problem, these procedures find near-optimal subsets of features that further improve the quality of the data-mining process.

The objective of feature selection is to find a subset of features with data-mining performances comparable with the full set of features. Given a set of features m, the number of subsets to be evaluated as candidates for column reduction is finite but still very large for iterative analysis through all cases. For practical reasons, an optimal search is not feasible, and simplifications are made to produce reasonable, acceptable, and timely results. If the reduction task is to create a subset, one possibility—the so-called bottom-up approach—starts with an empty set and fills it in by choosing the most relevant features from the initial set of features. This process is based on some heuristic criteria for a feature evaluation. The top-down approach, on the other hand, begins with a full set of original features and then removes one by one those that are shown as irrelevant based on the selected heuristic evaluation measure. Additional approximations to the optimal approach are as follows:

1. To examine only promising subsets of features where promising subsets are usually obtained heuristically. This provides enough room for exploration of competing alternatives.
2. To substitute computationally simple distance measures for the error measures. This approximation will reduce the computation time yet give satisfactory results for comparison of subset alternatives.
3. To select features based only on subsets of large amounts of data. The subsequent steps of data mining will be applied on the whole set.

The application of feature selection and reduction of data dimensionality may be used in all phases of the data-mining process for successful knowledge discovery. It has to be started in the preprocessing phase, but, on many occasions, feature selection and reduction is a part of the data-mining algorithm, even it is applied in postprocessing for better evaluation and consolidation of obtained results.

Let us return to the promising subsets of features. One possible technique for feature selection is based on comparison of *means* and *variances*. To summarize the key characteristics of the distribution of values for a given feature, it is necessary to compute the mean value and the corresponding variance. The main weakness in this approach is that the distribution for the feature is not known. If it is assumed to be a normal curve, the statistics can work out very well, but this may in fact be a poor assumption. Without knowing the shape of the distribution curve, the means and

variances are viewed as heuristics for feature selection, not exact, mathematical modeling tools.

In general, if one feature describes different classes of entities, samples of two different classes can be examined. The means of feature values are normalized by its variances and then compared. If the means are far apart, interest in a feature increases; it has potential, in terms of its use in distinguishing between two classes. If the means are indistinguishable, interest wanes in that feature. It is a heuristic, nonoptimal approach to feature selection, but it is consistent with practical experience in many data-mining applications in the triage of features. Next, equations formalize the test, where A and B are sets of feature values measured for two different classes and n_1 and n_2 are the corresponding number of samples:

$$\mathrm{SE}(A-B) = \sqrt{\left(\frac{\mathrm{var}(A)}{n_1} + \frac{\mathrm{var}(B)}{n_2}\right)}$$

$$\mathrm{TEST}: \frac{|\,\mathrm{mean}\,(A) - \mathrm{mean}\,(B)\,|}{\mathrm{SE}(A-B)} > \text{threshold-value}$$

The mean of a feature is compared between two classes without taking into consideration relationship to other features. In this approach to feature selection, we assumed a priori that the given feature is independent of the others. A comparison of means is a natural fit to classification problems. For the purpose of feature selection, a regression problem can be considered as a pseudo-classification problem. For k classes, k pairwise comparisons can be made, comparing each class to its complement. A feature is retained if it is significant for any of the pairwise comparisons.

We can analyze this approach in feature selection through one example. A simple data set is given in Table 3.1 with two input features X and Y and an additional output feature C that classifies samples into two classes A and B. It is necessary to decide whether the features X and Y are candidates for reduction or not. Suppose that the threshold value of the applied test is 0.5.

First, we need to compute a mean value and a variance for both classes and both features X and Y. The analyzed subsets of the feature's values are

TABLE 3.1. Data Set with Three Features

X	Y	C
0.3	0.7	*A*
0.2	0.9	*B*
0.6	0.6	*A*
0.5	0.5	*A*
0.7	0.7	*B*
0.4	0.9	*B*

$X_A = \{0.3, 0.6, 0.5\}$, $X_B = \{0.2, 0.7, 0.4\}$, $Y_A = \{0.7, 0.6, 0.5\}$, and $Y_B = \{0.9, 0.7, 0.9\}$

and the results of applied tests are

$$\mathrm{SE}(X_A - X_B) = \sqrt{\left(\frac{\mathrm{var}(X_A)}{n_1} + \frac{\mathrm{var}(X_B)}{n_2}\right)} = \sqrt{\left(\frac{0.0233}{3} + \frac{0.06333}{3}\right)} = 0.1699$$

$$\mathrm{SE}(Y_A - Y_B) = \sqrt{\left(\frac{\mathrm{var}(Y_A)}{n_1} + \frac{\mathrm{var}(Y_B)}{n_2}\right)} = \sqrt{\left(\frac{0.01}{3} + \frac{0.0133}{3}\right)} = 0.0875$$

$$\frac{|\,\mathrm{mean}(X_A) - \mathrm{mean}(X_B)\,|}{\mathrm{SE}(X_A - X_B)} = \frac{|\,0.4667 - 0.4333\,|}{0.1699} = 0.1961 < 0.5$$

$$\frac{|\,\mathrm{mean}(Y_A) - \mathrm{mean}(Y_B)\,|}{\mathrm{SE}(Y_A - Y_B)} = \frac{|\,0.6 - 0.8333\,|}{0.0875} = 2.6667 > 0.5$$

This analysis shows that X is a candidate for reduction because its mean values are close and, therefore, the final test is below the threshold value. On the other hand, the test for feature Y is significantly above the threshold value; this feature is not a candidate for reduction because it has the potential to be a distinguishing feature between two classes.

Similar idea for features ranking is given in the algorithm, which is based on correlation criteria. Let us consider first the prediction of a continuous outcome y. The Pearson correlation coefficient is defined as

$$\mathcal{R}(i) = \frac{\mathrm{cov}(X_i, Y)}{\sqrt{\mathrm{var}(X_i)\mathrm{var}(Y)}}$$

where cov designates the covariance and var the variance. The estimate of $R(i)$ for the given data set with samples' inputs $x_{k,j}$ and outputs y_k, is defined by

$$R(i) = \frac{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)(y_k - \bar{y})}{\sqrt{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)^2 \sum_{k=1}^{m}(y_k - \bar{y})^2}}$$

where the bar notation stands for an average over the index k (set of all samples). Using $R(i)^2$ as a variable ranking criterion enforces a ranking according to goodness of linear fit of individual variables. Correlation criteria such as $R(i)^2$ can only detect linear dependencies between input features and target or output feature (variable). One common criticism of variable ranking is that it leads to the selection of a redundant subset. The same performance could possibly be achieved with a smaller subset of complementary variables. Still, one may wonder whether deleting presumably redundant variables can result in a performance gain.

Practical experiments show that noise reduction and consequently better model estimation may be obtained by adding features that are presumably redundant. Therefore, we have to be very careful in the preprocessing analysis. Yes, perfectly correlated variables are truly redundant in the sense that no additional information is gained by adding them. But even variables with relatively high correlation (or anticorrelation) do not guarantee absence of variables' complementarity. We can find cases where a feature looks like completely useless by itself, and it is ranked very low, but it can provide significant information to the model and performance improvement when taken with others. These features by itself may have little correlation with the output, target concept, but when it is combined with some other features; they can be strongly correlated with the target feature. Unintentional removal of these features can result in poor mining performance.

The previous simple methods test features separately. Several features may be useful when considered separately, but they may be redundant in their predictive ability. If the features are examined collectively, instead of independently, additional information can be obtained about their characteristics and mutual relations. Assuming normal distributions of values, it is possible to describe an efficient technique for selecting subsets of features. Two descriptors characterize a multivariate normal distribution:

1. M—A vector of the m feature means.
2. C—An $m \times m$ covariance matrix of the means, where $C_{i,i}$ are simply the variance of feature i and $C_{i,j}$ terms are correlations between each pair of features:

$$C_{i,j} = \frac{1}{n} \sum_{k=1}^{n} (((v(k,i) - m(i)) * (v(k,j) - m(j)))$$

where $v(k,i)$ and $v(k,j)$ are the values of features indexed with i and j; $m(i)$ and $m(j)$ are feature means; and n is the number of dimensions

These two descriptors, M and C, provide a basis for detecting redundancies in a set of features. If two classes exist in a data set, then the heuristic measure, DM, for filtering features that separate the two classes is defined as

$$\mathrm{DM} = (M_1 - M_2)\,(C_1 + C_2)^{-1} (M_1 - M_2)^{\mathrm{T}}$$

where M_1 and C_1 are descriptors of samples for the first class and M_2 and C_2 for the second class. Given the target of k best features, all subsets of k from m features must be evaluated to find the subset with the largest DM. With large data sets that have features, this can be a huge search space, and alternative heuristic methods should be considered. One of these methods selects and ranks features based on an entropy measure. Detailed explanations are given in Section 3.4. The other heuristic approach, explained in the following text, is based on a combined correlation and covariance analyses and ranking of all features.

Existing efficient feature selection algorithms usually rank features under assumption of feature independence. Under this framework, features are ranked as relevant mainly based on their individually high correlations with the output feature. Because of the irreducible nature of feature interactions, these algorithms cannot select interacting features. In principle, it can be shown that a feature is relevant due to two reasons: (1) it is strongly correlated with the target feature; or (2) it forms a features' subset and the subset is strongly correlated with the target. A heuristic approach is developed to analyze features of type (2) in the selection process.

In the first part of a selection process, the features are ranked in descending order based on their correlation values with output using previously defined technique. We may assume that a set of features S can be divided into subset $S1$ including relevant features and subset $S2$ containing irrelevant ones. Heuristically, critical for removal are features in $S2$ first, while features in $S1$ are more likely to remain in the final set of selected features.

In the second part, features are evaluated one by one starting from the end of the $S2$ ranked feature list. The monotonic property suggests that the backward elimination search strategy fits best in feature selection. That is, one can start with the full feature set and successively eliminating features one at a time from the bottom of the ranked list if their interaction does not contribute to better correlation with output. The criterion could be, for example, based on covariance matrix. If a feature, together with previously selected features, shows influence on the output with less than threshold value (it could be expressed through some covariance matrix factor!), the feature is removed, and otherwise it is selected. Backward elimination allows every feature to be evaluated with the features it may interact with. The algorithm repeats until all features in the $S2$ list are checked.

3.2.2 Feature Extraction

The art of data mining starts with the design of appropriate data representations. Better performance is often achieved using features derived from the original input. Building a feature representation is an opportunity to incorporate domain knowledge into the data and can be very application specific. Transforming the input set into a new, reduced set of features is called *feature extraction*. If the features extracted are carefully chosen, it is expected that the new features set will extract the relevant information from the input data in order to perform the desired data mining task using this reduced representation.

Feature transformation techniques aim to reduce the dimensionality of data to a small number of dimensions, which are linear or nonlinear combinations of the original dimensions. Therefore, we distinguish two major types of dimension reduction methods: linear and nonlinear. Linear techniques result in k new derived features instead of initial p ($k \ll p$). Components of the new feature are a linear combination of the original features:

$$s_i = w_{i,1}x_1 + \cdots + w_{i,p}x_p \quad \text{for } i = 1, \ldots, k;$$

or in a matrix form

$$s = W x$$

where $W_{k \times p}$ is the linear transformation weight matrix. Such linear techniques are simpler and easier to implement than more recent methods considering nonlinear transforms.

In general, the process reduces feature dimensions by combining features instead of by deleting them. It results in a new set of fewer features with totally new values. One well-known approach is merging features by *principal components*. The features are examined collectively, merged, and transformed into a new set of features that, it is hoped, will retain the original information content in a reduced form. The basic transformation is linear. Given p features, they can be transformed into a single new feature F' by the simple application of weights:

$$F' = \sum_{j=1}^{p} w(j) \cdot f(j)$$

Most likely, a single set of weights $w(j)$ will not be adequate transformation for complex multidimensional data set, and up to p transformations are generated. Each vector of p features combined by weights w is called a principal component, and it defines a new transformed feature. The first vector of m weights is expected to be the strongest, and the remaining vectors are ranked according to their expected usefulness in reconstructing the original data. Eliminating the bottom-ranked transformation will reduce dimensions. The complexity of computation increases significantly with the number of features. The main weakness of the method is that it makes an advance assumption to a linear model that maximizes the variance of features. Formalization of principal components analysis (PCA) and the basic steps of the corresponding algorithm for selecting features are given in Section 3.5.

Examples of additional methodologies in feature extraction include factor analysis (FA), independent component analysis (ICA), and multidimensional scaling (MDS). Probably the last one is the most popular, and it represents the basis for some new, recently developed techniques. Given n samples in a p-dimensional space and an $n \times n$ matrix of distance measures among the samples, multidimensional scaling (MDS) produces a k-dimensional ($k \ll p$) representation of the items such that the distances among the points in the new space reflect the distances in the original data. A variety of distance measures may be used in the technique, and the main characteristic for all these measures is the more similar two samples are, the smaller their distance is. Popular distance measures include the Euclidean distance ($L2$ norm), the Manhattan distance ($L1$, absolute norm), and the maximum norm, and more detail about these measures and their applications is given in Chapter 9. MDS has been typically used to transform the data into two or three dimensions, visualizing the result to uncover hidden structure in the data. A rule of thumb to determine the maximum number of k is to ensure that there are at least twice as many pairs of samples than the number of parameters to be estimated, resulting in $p \geq k + 1$. Results of the MDS technique are indeterminate with respect to translation, rotation, and reflection of data.

PCA and multidimensional scaling (MDS) are both simple methods for linear dimensionality reduction, where an alternative to MDS is FastMap, a computationally efficient algorithm. The other variant, *Isomap*, has also emerged as a powerful technique for nonlinear dimensionality reduction, and it is primary graph-based method.

Isomap is based on computing the low-dimensional representation of a high-dimensional data set that most faithfully preserves the pairwise distances between input samples as measured along geodesic distance (details about geodesic are given in Chapter 12, the section about graph mining). The algorithm can be understood as a variant of MDS in which estimates of geodesic distances are substituted for standard Euclidean distances.

The algorithm has three steps. The first step is to compute the k-nearest neighbors of each input sample and to construct a graph whose vertices represent input samples and whose (undirected) edges connect k-nearest neighbors. The edges are then assigned weights based on the Euclidean distance between nearest neighbors. The second step is to compute the pairwise distances between all nodes (i, j) along shortest paths through the graph. This can be done using well-known Dijkstra's algorithm with complexity $O(n^2 \log n + n^2 k)$. Finally, in the third step, the pairwise distances are fed as input to MDS to determine new reduced set of features.

With the size of data getting bigger and bigger, all feature selection (and reduction) methods also face a problem of oversized data because of a computer's limited resources. But do we really need so much data for selecting features as an initial process in data mining? Or can we settle for less data? We know that some portion of a huge data set can represent it reasonably well. The point is which portion and how large should it be. Instead of looking for the right portion, we can randomly select a part, P, of a data set, use that portion to find the subset of features that satisfy the evaluation criteria, and test this subset on a different part of the data. The results of this test will show whether the task has been successfully accomplished. If an inconsistency is found, we shall have to repeat the process with a slightly enlarged portion of the initial data set. What should be the initial size of the data subset P? Intuitively, we know that its size should not be too small or too large. A simple way to get out of this dilemma is to choose a percentage of data, say, 10%. The right percentage can be determined experimentally.

What are the results of a feature-reduction process, and why do we need this process for every specific application? The purposes vary, depending upon the problem on hand, but, generally, we want:

1. to *improve performances* of the model-generation process and the resulting model itself (typical criteria are speed of learning, predictive accuracy, and simplicity of the model);
2. to *reduce dimensionality* of the model without reduction of its quality through:
   (a) Elimination of irrelevant features,
   (b) Detection and elimination of redundant data and features,

(c) Identification of highly correlated features, and
(d) Extraction of independent features that determine the model; and

3. to help the user *visualize* alternative results, which have fewer dimensions, to improve decision-making.

3.3 RELIEF ALGORITHM

Relief is a feature weight-based algorithm for feature selection inspired by so-called instance-based learning. It relies on relevance evaluation of each feature given in a training data set. The main idea of *Relief* is to compute a ranking score for every feature indicating how well this feature separates neighboring samples. The authors of the Relief algorithm, Kira and Rendell, proved that the ranking score is large for relevant features and small for irrelevant ones.

The core of the *Relief* algorithm is to estimate the quality of features according to how well their values distinguish between samples close to each other. Given training data S, the algorithm randomly selects subset of sample size m, where m is a user-defined parameter. *Relief* analyzes each feature based on a selected subset of samples. For each randomly selected sample X from a training data set, *Relief* searches for its two nearest neighbors: one from the same class, called nearest hit H, and the other one from a different class, called nearest miss M. An example for two-dimensional data is given in Figure 3.2.

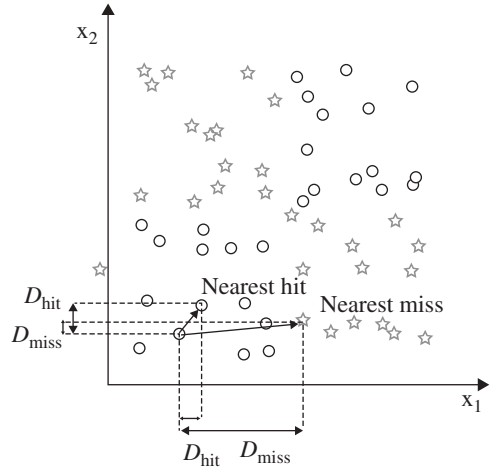


Figure 3.2. Determining nearest hit H and nearest miss M samples.

8. *N-dimensional visualization methods* are usually skipped in the literature as a standard data-mining methodology, although useful information may be discovered using these techniques and tools. Typical data-mining visualization techniques are geometric, icon-based, pixel-oriented, and hierarchical techniques.

This list of data-mining and knowledge-discovery techniques is not exhaustive, and the order does not suggest any priority in the application of these methods. Iterations and interactivity are basic characteristics of these data-mining techniques. Also, with more experience in data-mining applications, the reader will understand the importance of not relying on a single methodology. Parallel application of several techniques that cover the same inductive-learning task is a standard approach in this phase of data mining. In that case, for each iteration in a data-mining process, the results of the different techniques must additionally be evaluated and compared.

4.5 SUPPORT VECTOR MACHINES

The foundations of support vector machines (SVM) have been developed by Vladimir Vapnik and are gaining popularity due to many attractive features and promising empirical performance. The formulation embodies the SRM principle. SVMs were developed to solve the classification problem, but recently they have been extended to the domain of regression problems (for prediction of continuous variables). SVMs can be applied to regression problems by the introduction of an alternative loss function that is modified to include a distance measure. The term SVM is referring to both classification and regression methods, and the terms support vector classification (SVC) and support vector regression (SVR) may be used for more precise specification.

An SVM is a supervised learning algorithm creating learning functions from a set of labeled training data. It has a sound theoretical foundation and requires relatively small number of samples for training, and experiments showed that it is insensitive to the number of sample's dimensions. Initially, the algorithm addresses the general problem of learning to discriminate between members of two classes represented as n-dimensional vectors. The function can be a classification function (the output is binary) or the function can be a general regression function.

SVM's classification function is based on the concept of decision planes that define decision boundaries between classes of samples. A simple example is shown in Figure 4.16a where the samples belong either to class gray or black. The separating line defines a boundary on the right side of which all samples are gray and to the left of which all samples are black. Any new unclassified sample falling to the right will be classified as gray (or classified as black should it fall to the left of the separating line).

The classification problem can be restricted to consideration of the two-class problem without loss of generality. Before considering n-dimensional analysis, let us look at a simple two-dimensional example. Assume we wish to perform a classification, and our data has a categorical target variable with two categories. Also

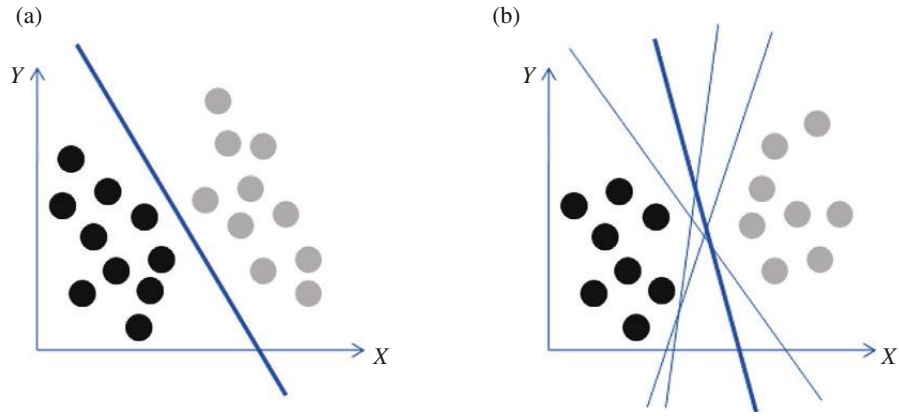


Figure 4.16. Linear separation in 2D space. (a) A decision plane in 2D space is a line. (b) How to select optimal separating line.

assume that there are two input attributes with continuous values. If we plot the data points using the value of one attribute on the X axis and the other on the Y axis, we might end up with an image such as shown in Figure 4.16b. In this problem the goal is to separate the two classes by a function that is induced from available examples. The goal is to produce a classifier that will work well on unseen examples, i.e. it generalizes well. Consider the data in Figure 4.16b. Here there are many possible linear classifiers that can separate the two classes of samples. Are all decision boundaries equally good? How to prove that selected one is the best?

The main idea is that the decision boundary should be as far away as possible from the data points of both classes. There is only one that maximizes the margin (maximizes the distance between it and the nearest data point of each class). Intuitively, the margin is defined as the amount of space or separation between the two classes as defined by the hyperplane. Geometrically, the margin corresponds to the shortest distance between the closest data points to a point on the hyperplane. STL suggests that the choice of the maximum margin hyperplane will lead to maximal generalization when predicting the classification of previously unseen examples.

Therefore a linear SVM classifier is termed the optimal separating hyperplane with the maximum margin such as the margin in Figure 4.17b. The goal of SVM modeling in n-dimensional spaces is to find the optimal hyperplane that separates classes of n-dimensional vectors. The split will be chosen again to have the largest distance from the hypersurface to the nearest of the positive and negative samples. Intuitively, this makes the classification correct for testing data that is near, but not identical to the training data.

Why we should maximize the margin? Skinny *margin* is more flexible and thus more complex, and the complexity is not the goal. Fat *margin* is less complex. SRM principle expresses a trade-off between training error and model complexity. It

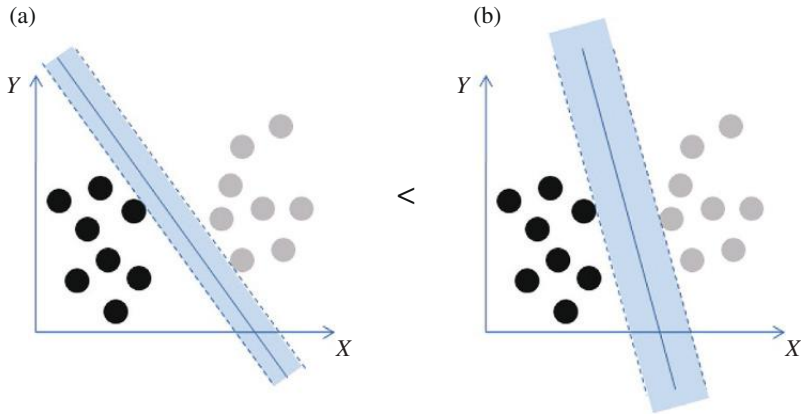


Figure 4.17. Comparison between sizes of margin of different decision boundaries. (a) Margin of decision boundary 1. (b) Margin of decision boundary 2.

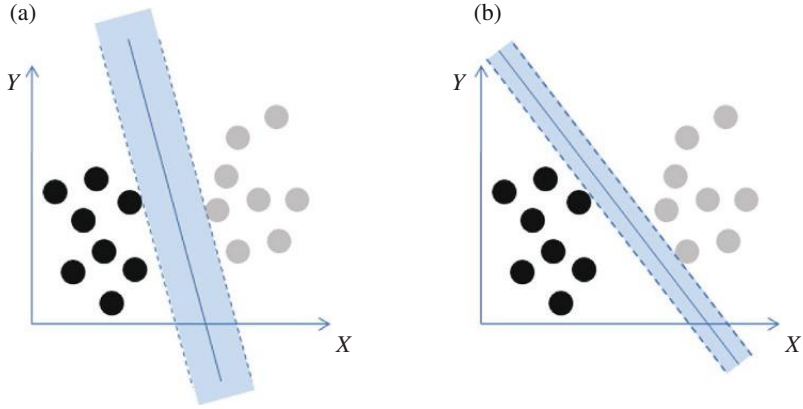


Figure 4.18. SRM principle expresses a trade-off between training error and model complexity. (a) "Fat" margin. (b) "Skinny" margin.

recommends maximum margin, such as the one in Figure 4.18, as an optimal separation criteria ensuring that SVM worst-case generalization errors are minimized.

Based on the vector equation of the line in 2D, we can define function $f(x) = w\,x + b$ as a separation model. For all points above line $f(x) > 0$, and for the points below line $f(x) < 0$. We define the sign of this function $h(x) = \text{sign}(f(x))$ as a classification function because it has the value 1 for all points above the line and the value -1 for all points below line. An example is given in Figure 4.19.

Before we continue, it is important to note that while the above examples show 2D data set, which can be conveniently represented by points in a plane, in fact we will

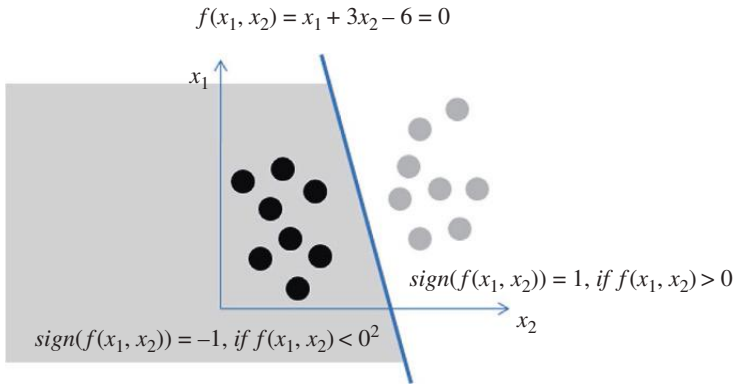


Figure 4.19. Classification function, sign (*f*(*x*)), on a 2D space.

typically be dealing with higher-dimensional data. The question is how to determine an optimal hyperplane in *n*-dimensional spaces based on given set of training samples. Consider the problem of separating the set of training vectors *D* belonging to two classes (coded binary with −1 and 1)

$$D = \left\{ \left(\mathbf{x}^l, \mathbf{y}^l\right), \ldots, \left(\mathbf{x}^l, \mathbf{y}^l\right) \right\}, \quad \mathbf{x} \in \Re^{\mathrm{n}}, \mathbf{y} \in \{-1, 1\},$$

with a hyperplane

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0.$$

The set of vectors is said to be optimally separated by the hyperplane if it is separated without error and the distance between the closest vectors to the hyperplane is maximal. An illustration of the separation with a graphical interpretation of main parameters w and b is given in Figure 4.20a. In this way we have parameterized the function by the weight vector *w* and the scalar *b*. The notation $\langle w, x \rangle$ denotes the inner or scalar product of vectors *w* and *x*, defined by

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{n} w_i x_i$$

In order for our hyperplane to correctly separate the two classes, we need to satisfy the following constraints:

$$\langle \mathbf{w}, \mathbf{x}^i \rangle + b > 0, \quad \text{for all } y^i = 1$$
$$\langle \mathbf{w}, \mathbf{x}^i \rangle + b < 0, \quad \text{for all } y^i = -1$$

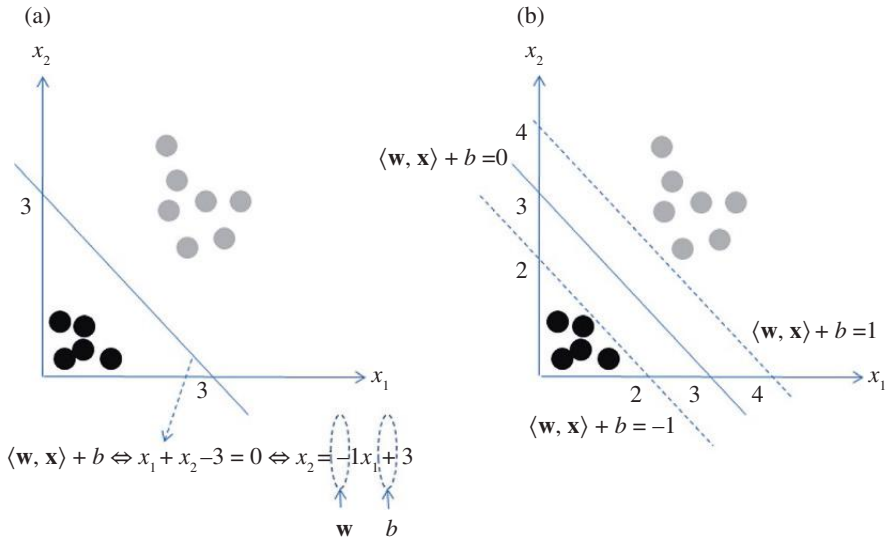


Figure 4.20. A separating hyperplane (w, b) for 2D data. (a) Parameters w and b. (b) Two parallel hyperplanes define margin.

The set of constraints that we have so far is equivalent to saying that these data must lie on the correct side (according to class label) of this decision surface. Next notice that we have also plotted as dotted lines two other hyperplanes represented in Figure 4.20b, which are the hyperplanes where the function $\langle w, x \rangle + b$ is equal to -1 (on the lower left) and $+1$ (on the upper right). In order to find the maximum margin hyperplane, we can see intuitively that we should keep the dotted lines parallel and equidistant to the decision surface, and maximize their distance from one another, while satisfying the constraint that the data lie on the correct side of the dotted lines associated with that class. In mathematical form, the final clause of this sentence (the constraints) can be written as

$$y^i \left(\left\langle \mathbf{w}, \mathbf{x}^i \right\rangle + b \right) \geq 1, \quad i = 1, \ldots, l.$$

The distance between these two margin hyperplanes may be formalized, because it is the parameter we want to maximize. We may obtain the distance between hyperplanes in nD space using equations

$$\left\langle \mathbf{w}, \mathbf{x}^1 \right\rangle + b = 1$$
$$\left\langle \mathbf{w}, \mathbf{x}^2 \right\rangle + b = -1$$

where x^1 and x^2 are any points on corresponding hyperplanes. If we subtract these equations

$$\langle \mathbf{w}, (\mathbf{x}^1 - \mathbf{x}^2) \rangle = 2$$

and representing scalar product of vectors by definition,

$$w(x^1 - x^2)\cos\gamma = 2$$

we obtain

$$\|\mathbf{w}\| \times d = 2,$$

where || || represents Euclidean norm or

$$d = \frac{2}{\|\mathbf{w}\|}$$

Therefore, the problem of "maximum margin" is represented as a maximum of a distance parameter d, which is a function of parameters w. Maximizing d means maximizing $1/|w|$ or minimizing $|w|$. The learning problem may be reformulated as

$$\arg\min_{w} \frac{1}{2}(\mathbf{w}\cdot\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$$

subject to the constraints of linear separability. So, the final problem for optimization is

$$\arg\min_{\mathbf{w},b} \frac{1}{2}(\mathbf{w}\cdot\mathbf{w}) \quad \text{such that} \, y^i\left(\langle \mathbf{w}, \mathbf{x}^i \rangle + b\right) \geq 1 \text{ for all } \; i = 1, 2, \ldots, l$$

The problem of optimization under constraints may be transformed using the Lagrangian $L(w,b,\alpha)$:

$$L(\mathbf{w}, b, \alpha) = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{l} \alpha_i \left\{ \left(\langle \mathbf{w}, \mathbf{x}^i \rangle + b \right) y^i - 1 \right\}$$

where α_i are the Lagrange multipliers, one for each data point. The first term is the same as the original objective function, and the second term captures the inequality constraints. The Lagrangian has to be minimized with respect to w and b:

$$\frac{\partial L}{\partial b} = 0 \;\; \Rightarrow \;\; \sum_{i=0}^{l} \alpha_i y^i = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \;\; \Rightarrow \;\; \mathbf{w}_0 = \sum_{i=0}^{l} y^i \alpha_i \mathbf{x}^i = 0$$

Substituting results of partial derivatives into L leads to the dual formulation of the optimization problem, which has to be maximized with respect to the constraints $\alpha_i \geq 0$:

$$D(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y^i y^j \left(\mathbf{x}^i \cdot \mathbf{x}^j\right)$$

The dual Lagrangian $D(\alpha)$ involves only the Lagrangian multipliers α_i and the training data (there are no more parameters w and b). Finding the solution for real-world problems will usually require application of quadratic programming (QP) optimization techniques. This problem has a global optimum. The optimization approach for SVM provides an accurate implementation of the SRM inductive principle. When α_i parameters are determined, it is necessary to determine the values for w and b, and they determine final classification hyperplane. It is important to note that dual function D is function of only scalar products of sample vectors, not of vectors alone. Once the solution has been found in the form of a vector α^0, the optimal separating hyperplane is given by

$$\mathbf{w}_0 = \sum_{i \in SVs} y^i \alpha_i^0 \mathbf{x}^i$$

$$b_0 = -\frac{1}{2} \mathbf{w}_0 \cdot \left[\mathbf{x}^r + \mathbf{x}^s\right]$$

where x_r and x_s are any support vectors (SVs) from each class. The classifier can then be constructed as

$$f(x) = \text{sign}(\langle \mathbf{w}_0, \mathbf{x} \rangle + b_0) = \text{sign}\left(\sum_{i \in \text{SV}s} y^i \alpha_i^0 \left(\mathbf{x}^i \cdot \mathbf{x}\right) + b_0 \right)$$

Only the points x^i that will have nonzero Lagrangian multipliers α^0 are termed *support vectors*. If the data is linearly separable, all the SVs will lie on the margin, and hence the number of SVs can be very small as it is represented in Figure 4.21. This "sparse" representation can be viewed as data compression in the construction of the classifier. The SVs are the "hard" cases; these are the training samples that are most difficult to classify correctly and that lie closest to the decision boundary.

The SVM learning algorithm is defined so that, in a typical case, the number of SVs is small compared with the total number of training examples. This property allows the SVM to classify new examples efficiently, since the majority of the training examples can be safely ignored. SVMs effectively remove the uninformative patterns from the data set by assigning them α_i weights of zero. So, if internal points that are not SVs are changed, no effect will be made on the decision boundary. The hyperplane is represented sparsely as a linear combination of "SV" points. The SVM automatically identifies a subset of these informative points and uses them to represent the solution.

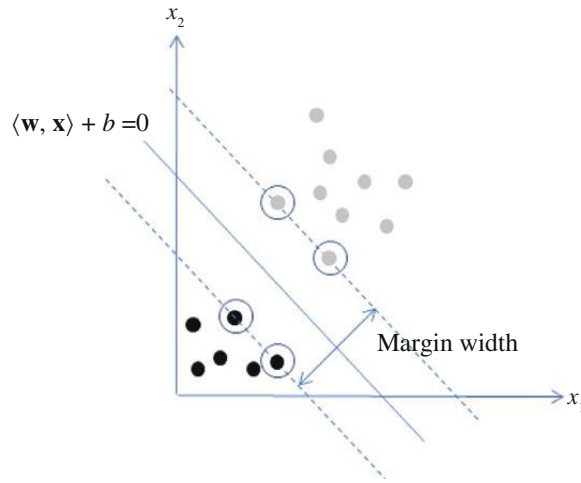


Figure 4.21. A maximal margin hyperplane with its support vectors encircled.

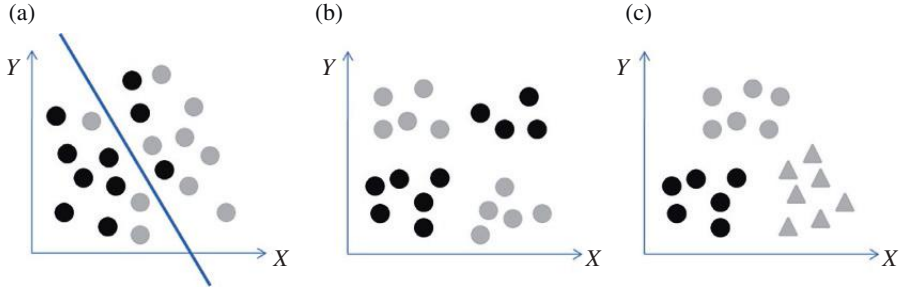


Figure 4.22. Issues for an SVM in real-world applications. (a) Subsets cannot be completely separated. (b) Nonlinear separation. (c) Three categories.

In real-world applications SVMs must deal with (1) handling the cases where subsets cannot be completely separated, (2) separating the points with nonlinear surfaces, and (3) handling classifications with more than two categories. Illustrative examples are given in Figure 4.22. What are solutions in these cases? We will start with the problem of data that are not linearly separable. The points such as shown in Figure 4.23a could be separated only by a nonlinear region. Is it possible to define linear margin where some points may be on opposite sides of hyperplanes?

Obviously, there is no hyperplane that separates all of the samples in one class from all of the samples in the other class. In this case there would be no combination of w and b that could ever satisfy the set of constraints. This situation is depicted in Figure 4.23b, where it becomes apparent that we need to *soften* the constraint that these data lay on the correct side of the +1 and −1 hyperplanes. We need to allow

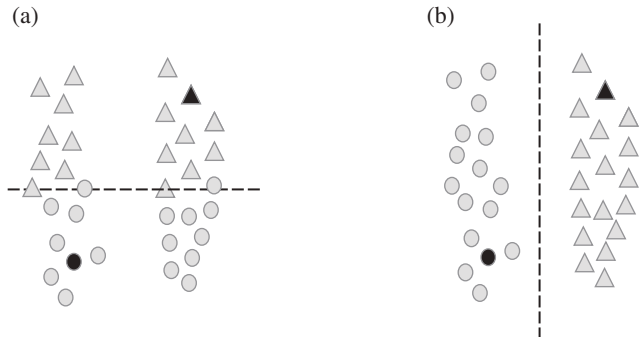


Figure 4.29. Continuity and clustering assumption determine the quality of semi-supervised learning. (a) S3VM is not the best approach. (b) S3VM is the appropriate approach.

2. Build one standard SVM for each labeling case in the previous step (and variety of X_l splitting).
3. Pick the SVM with the largest margin.

Obviously the algorithm has a problem of combinatorial explosion of alternatives in the first step! Variety of optimization methods are applied, and different S3VM implementations shows reasonable performances in practice (for example, min-cut approach in a graph of unlabeled and labeled samples). Obviously, S3VM have serious deficiency in a case of big (unlabeled) data. The methods are using extremely large time in training, and it is currently the biggest challenge in all implementations of S3VM.

In general, there is no uniform SSL solution for all applications with available both unlabeled and labeled data. Depending on available data and knowledge about the problem, we may use different techniques and approaches: from standard supervised learning (such as SVM) when there are enough labeled samples to different SSL techniques including S3VM when two assumptions in data set are satisfied. If SSL is used as a first step, verify and discuss solution by comparing with other approaches including supervised and even unsupervised learning model. Figure 4.29 gives only illustrative examples with applicable and non-applicable S3VM.

4.7 kNN: NEAREST NEIGHBOR CLASSIFIER

Unlike SVM's global classification model, *k*-nearest neighbor (kNN) classifier determines the decision boundary locally. For 1NN we assign each new sample to the class of its closest neighbor as it is represented in Figure 4.30a. Initially we have samples belonging to two classes (+ and −). The new sample "?" should be labeled with the class of its closest neighbor. 1NN classifier is not very robust methodology. The classification decision of each test sample relies on the class of a single training sample,

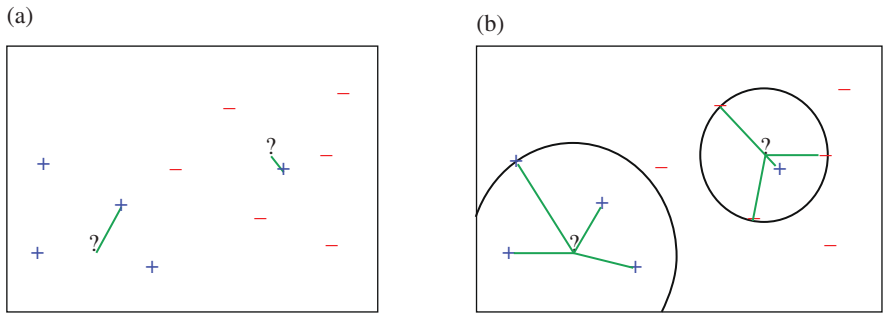


Figure 4.30. *k*-Nearest neighbor classifier. (a) $k = 1$. (b) $k = 4$

which may be incorrectly labeled or atypical. For larger k, kNN will assign new sample to the majority class of its k closest neighbors where k is a parameter of the methodology. An example for $k = 4$ is given in Figure 4.30b. kNN classifier for $k > 1$ is more robust. Larger k values help reduce the effects of noisy points within the training data set.

The rationale of kNN classification is that we expect a test sample X to have the same label as the training sample located in the local region surrounding X. *K*NN classifiers are lazy learners, that is, models are not built explicitly unlike SVM and the other classification models given in the following chapters. Training a kNN classifier simply consists of determining k. In fact, if we preselect a value for k and do not preprocess given samples, then kNN approach requires no training at all. kNN simply memorizes all samples in the training set and then compares the test sample to them. For this reason, kNN is also called *memory-based learning* or *instance-based learning*. It is usually desirable to have as much training data as possible in machine learning. But in kNN, large training sets come with a severe efficiency penalty in classification of testing samples.

Building the kNN model is computationally cheap (just store the training data), but classifying unknown sample is relatively expensive since it requires the computation of the kNN of the testing sample to be labeled. This, in general, requires computing the distance of the unlabeled object to all the objects in the labeled set, which can be expensive particularly for large training sets. Among the various methods of supervised learning, the nearest neighbor classifier achieves consistently high performance, without a priori assumptions about the distributions from which the training examples are drawn. The reader may have noticed the similarity between the problem of finding nearest neighbors for a test sample and ad hoc retrieval methodologies. In standard information retrieval systems such as digital libraries or Web search, we search for the documents (samples) with the highest similarity to the query document represented by a set of keywords. Problems are similar, and often the proposed solutions are applicable in both disciplines.

Decision boundaries in 1NN are concatenated segments of the *Voronoi diagram* as shown in Figure 4.31. The Voronoi diagram decomposes space into Voronoi cells,

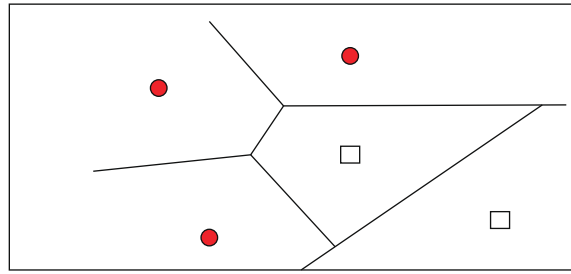


Figure 4.31. Voronoi diagram in 2D space.

where each cell consists of all points that are closer to the sample than to other samples. Assume that we have X training samples in 2D space. The diagram then partitions the 2D plane into $|X|$ convex polygons, each containing its corresponding sample (and no other), where a convex polygon is a convex region in 2D space bounded by lines. For general $k > 1$ case, consider the region in the space for which the set of kNN is the same. This again is a convex polygon and the space is partitioned into convex polygons, within each of which the set of kNN is invariant.

The parameter k in kNN is often chosen based on experience or knowledge about the classification problem at hand. It is desirable for k to be odd to make ties less likely. $k = 3$ and $k = 5$ are common choices, but much larger values up to 100 are also used. An alternative way of setting the parameter is to select k through the iterations of testing process and select k that gives best results on testing set.

Time complexity of the algorithm is linear in the size of the training set as we need to compute the distance of each training sample from the new test sample. Of course, the computing time goes up as k goes up, but the advantage is that higher values of k provide smoothing of the classification surface that reduces vulnerability to noise in the training data. At the same time high value for k may destroy the locality of the estimation since farther samples are taken into account, and large k increases the computational burden. In practical applications, typically, k is in units or tens rather than in hundreds or thousands. The nearest neighbor classifier is quite simple algorithm, but very computationally intensive especially in the testing phase. The choice of the distance measure is another important consideration. It is well known that the Euclidean distance measure become less discriminating as the number of attributes increases, and in some cases it may be better to use cosine or other measures rather than Euclidean distance.

Testing time of the algorithm is independent of the number of classes, and kNN therefore has a potential advantage for classification problems with multiple classes. For the example in Figure 4.32, we have three classes (ω_1, ω_2, ω_3) represented by a set of training samples, and the goal is to find a class label for the testing sample x_u. In this case, we use the Euclidean distance and a value of $k = 5$ neighbors as the threshold. Of the five closest neighbors, four belong to ω_1 class and one belongs to ω_3 class, so x_u is assigned to ω_1 as the predominant class in the neighborhood.

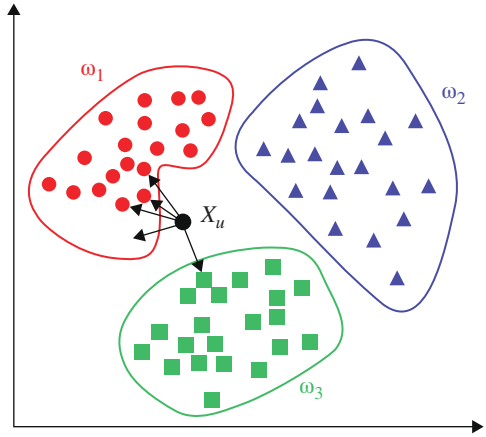


Figure 4.32. Nearest neighbor classifier for $k = 5$.

In summary, kNN classifier only requires a parameter k, a set of labeled training samples, and a metric measure for determining distances in n-dimensional space. kNN classification process is usually based on the following steps:

- Determine parameter k—number of nearest neighbors.
- Calculate the distance between each testing sample and all the training samples.
- Sort the distance and determine nearest neighbors based on the kth threshold.
- Determine the category (class) for each of the nearest neighbors.
- Use simple majority of the category of nearest neighbors as the prediction value of the testing sample classification.

There are many techniques available for improving the performance and speed of a nearest neighbor classification. One solution is to choose a subset of the training data for classification. The idea of the *condensed nearest neighbor (CNN)* is to select the smallest subset Z of training data X such that when Z is used instead of X, error in classification of new testing samples does not increase. 1NN is used as the nonparametric estimator for classification. It approximates the classification function in a piecewise linear manner. Only the samples that define the classifier need to be kept. Other samples, inside regions, need not to be stored because they belong to the same class. An example of CNN classifier in 2D space is given in Figure 4.33. Greedy CNN algorithm is defined with the following steps:

1. Start with empty set Z.
2. Passing samples from X one by one in a random order, and check whether they can be classified correctly by instances in Z.

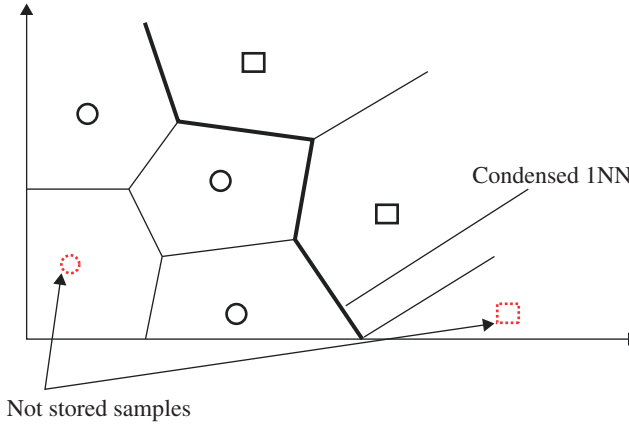


Figure 4.33. CNN classifier in 2D space.

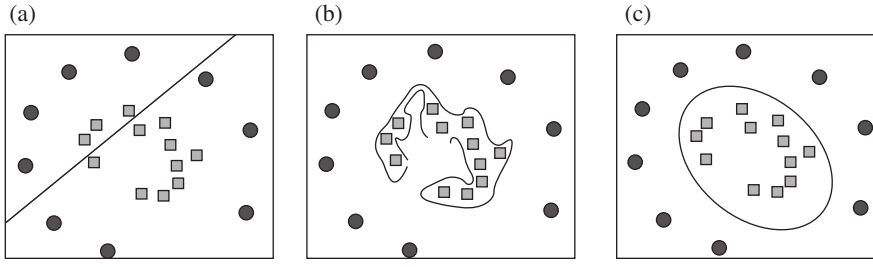


Figure 4.34. Trade-off between model complexity and the amount of data. (a) Too simple model. (b) Too complex model. (c) Appropriate model.

3. If a sample is misclassified, it is added to Z; if it is correctly classified, Z is unchanged.
4. Repeat a few times over training data set until Z is unchanged. Algorithm does not guarantee minimum subset for Z.

kNN methodology is relatively simple and could be applicable in many real-world problems. Still, there are some methodological problems such as scalability, "curse of dimensionality," influence of irrelevant attributes, weight factors in the distance measure, weight factors for votes of k neighbors, etc.

4.8 MODEL SELECTION VS. GENERALIZATION

We assume that the empirical data is given according to an unknown probability distribution. The question arises as to whether a finite set of empirical data includes sufficient information such that the underlying regularities can be learned and

partitioning will give different estimates. A repetition of the process, with different training and testing sets randomly selected, and integration of the error results into one standard parameter, will improve the estimate of the model.

3. *Leave-one-out method*—A model is designed using $(n - 1)$ samples for training and evaluated on the one remaining sample. This is repeated n times with different training sets of size $(n - 1)$. This approach has large computational requirements because n different models have to be designed and compared.
4. *Rotation method (n-fold cross-validation)*—This approach is a compromise between holdout and leave-one-out methods. It divides the available samples into P disjoint subsets, where $1 \le P \le n$. $(P - 1)$ subsets are used for training and the remaining subset for testing. This is the most popular method in practice, especially for problems where the number of samples is relatively small.
5. *Bootstrap method*—This method resamples the available data with replacements to generate a number of "fake" data sets of the same size as the given data set. The number of these new sets is typically several hundreds. These new training sets can be used to define so-called bootstrap estimates of the error rate. Experimental results have shown that the bootstrap estimates can outperform the cross-validation estimates. This method is especially useful in small data set situations.

4.9 MODEL ESTIMATION

A model realized through the data-mining process using different inductive-learning techniques might be estimated using the standard error rate parameter as a measure of its performance. This value expresses an approximation of the true error rate, a parameter defined in STL. The error rate is computed using a testing data set obtained through one of applied resampling techniques. In addition to the accuracy measured by the error rate, data-mining models can be compared with respect to their speed, robustness, scalability, and interpretability, and all these parameters may have an influence on the final verification and validation of the model. In the short overview that follows, we will illustrate primary the characteristics of the error rate parameter for classification tasks; similar approaches and analyses are possible for other common data-mining tasks.

The computation of error rate is based on counting of errors in a testing process. These errors are, for a classification problem, simply defined as misclassification (wrongly classified samples). If all errors are of equal importance, an error rate R is the number of errors E divided by the number of samples S in the testing set:

$$R = \frac{E}{S}$$

The accuracy AC of a model is a part of the testing data set that is classified correctly, and it is computed as one minus the error rate:

$$\text{AC} = 1 - R = \frac{(S - E)}{S}$$

For standard classification problems, there can be as many as $m^2 - m$ types of errors, where m is the number of classes.

Two tools commonly used to assess the performance of different classification models are the *confusion matrix* and the *lift chart*. A confusion matrix, sometimes called a classification matrix, is used to assess the prediction accuracy of a model. It measures whether a model is confused or not, that is, whether the model is making mistakes in its predictions. The format of a confusion matrix for a two-class case with classes yes and no is shown in Table 4.2.

If there are only two classes (positive and negative samples, symbolically represented with T and F or with 1 and 0), we can have only two types of errors:

1. It is expected to be T, but it is classified as F: these are false negative errors (C: False–).
2. It is expected to be F, but it is classified as T: these are false positive errors (B: False+).

If there are more than two classes, the types of errors can be summarized in a confusion matrix, as shown in Table 4.3. For the number of classes $m = 3$, there are six types of errors ($m^2 - m = 3^2 - 3 = 6$), and they are represented in bold type in Table 4.3. Every class contains 30 samples in this example, and the total is 90 testing samples.

TABLE 4.2. Confusion Matrix for Two-Class Classification Model

Predicted Class	Actual Class	
	Class 1	Class 2
Class 1	**A**: True +	**B**: False +
Class 2	**C**: False –	**D**: True –

TABLE 4.3. Confusion Matrix for Three Classes

Classification Model	True Class			Total
	0	1	2	
0	28	**1**	**4**	33
1	**2**	28	**2**	32
2	**0**	**1**	24	25
Total	*30*	*30*	*30*	*90*

TABLE 4.4. Evaluation Metrics for Confusion Matrix 2×2

Evaluation Metrics	Computation Using Confusion Matrix
True positive rate (TP)	TP = $A/(A + C)$
False positive rate (FP)	FP = $B/(B + \mathrm{D})$
Sensitivity (SE)	SE = TP
Specificity (SP)	SP = 1-FP
Accuracy (AC)	AC = $(A + D)/(A + B + C + D)$
Recall (R)	$R = A/(A + C)$
Precision (P)	$P = A/(A + B)$
F-measure (F)	$F = 2PR/(P+R)$

The error rate for this example is

$$R = \frac{E}{S} = \frac{10}{90} = 0.11$$

and the corresponding accuracy is

$$\mathrm{AC} = 1 - R = 1 - 0.11 = 0.89 \,(\text{or as a percentage}: A = 89\%)$$

Accuracy is not always the best measure of the quality of the classification model. It is especially true for the real-world problems where the distribution of classes is unbalanced, for example, if the problem is classification of healthy person from these with the disease. In many cases the medical database for training and testing will contain mostly healthy person (99%) and only small percentage of people with disease (about 1%). In that case, no matter how good the accuracy of a model is estimated to be, there is no guarantee that it reflects the real world. Therefore, we need other measures for model quality. In practice, several measures are developed, and some of best known are presented in Table 4.4. Computation of these measures is based on parameters A, B, C, and D for the confusion matrix in Table 4.2. Selection of the appropriate measure depends on the application domain, and for example, in medical field, the most often used are measures: sensitivity and specificity.

Previous measures are primarily developed for classification problems where the output of the model is expected to be a categorical variable. If the output is numerical, several additional prediction accuracy measures for regression problems are defined. It is assumed that the prediction error for each sample e_i is defined as the difference between its actual output $\mathrm{Y_a}$ value and predicted $\mathrm{Y_p}$ value:

$$\mathrm{e_i} = \mathrm{Y_{ai}} - \mathrm{Y_{pi}}.$$

formalized, "executable" models of classification, and there are two very different ways in which they can be constructed. On the one hand, the model might be obtained by interviewing the relevant expert or experts, and most knowledge-based systems have been built this way despite the well-known difficulties attendant on taking this approach. Alternatively, numerous recorded classifications might be examined, and a model constructed inductively by generalizing from specific examples that are of primary interest for data-mining applications.

The statistical approach to classification explained in Chapter 5 gives one type of model for classification problems: summarizing the statistical characteristics of the set of samples. The other approach is based on logic. Instead of using math operations like addition and multiplication, the logical model is based on expressions that are evaluated as true or false by applying Boolean and comparative operators to the feature values. These methods of modeling give accurate classification results compared with other nonlogical methods, and they have superior explanatory characteristics. Decision trees and decision rules are typical data-mining techniques that belong to a class of methodologies that give the output in the form of logical models.

6.1 DECISION TREES

A particularly efficient method for producing classifiers from data is to generate a decision tree. The decision-tree representation is the most widely used logic method. There is a large number of decision-tree induction algorithms described primarily in the machine-learning and applied-statistics literature. They are supervised learning methods that construct decision trees from a set of input–output samples. It is an efficient nonparametric method for classification and regression. A decision tree is a hierarchical model for supervised learning where the *local region* is identified in a sequence of recursive *splits* through decision nodes with test function. A decision tree is also a nonparametric model in the sense that we *do not assume any parametric form* for the class density.

A typical decision-tree learning system adopts a top-down strategy that searches for a solution in a part of the search space. It guarantees that a simple, but not necessarily the simplest, tree will be found. A decision tree consists of *nodes* where attributes are tested. In a univariate tree, for each internal node, the test uses only one of the attributes for testing. The outgoing *branches* of a node correspond to all the possible outcomes of the test at the node. A simple decision tree for classification of samples with two input attributes X and Y is given in Figure 6.2. All samples with feature values $X > 1$ and $Y = B$ belong to Class2, while the samples with values $X < 1$ belong to Class1, whatever the value for feature Y. The samples, at a nonleaf node in the tree structure, are thus partitioned along the branches, and each child node gets its corresponding subset of samples. Decision trees that use univariate splits have a simple representational form, making it relatively easy for the user to understand the inferred model; at the same time, they represent a restriction on the expressiveness of the model. In general, any restriction on a particular tree representation can significantly

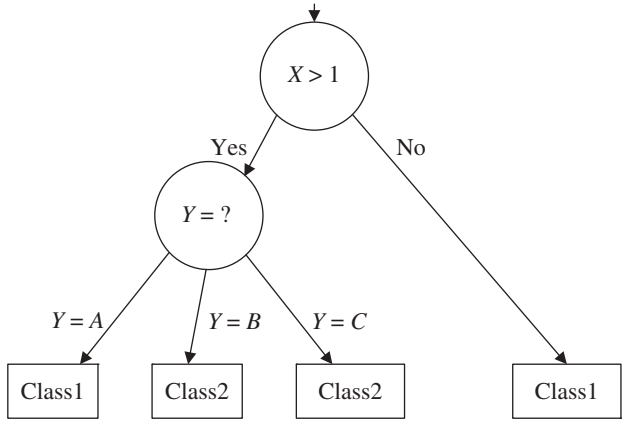


Figure 6.2. A simple decision tree with the tests on attributes *X* and *Y*.

restrict the functional form and thus the approximation power of the model. A well-known tree-growing algorithm for generating decision trees based on univariate splits is Quinlan's *ID3* with an extended version called *C4.5*. Greedy search methods, which involve growing and pruning decision-tree structures, are typically employed in these algorithms to explore the exponential space of possible models.

The ID3 algorithm starts with all the training samples at the root node of the tree. An attribute is selected to partition these samples. For each value of the attribute, a branch is created, and the corresponding subset of samples that have the attribute value specified by the branch is moved to the newly created child node. The algorithm is applied recursively to each child node until all samples at a node are of one class. Every path to the leaf in the decision tree represents a classification rule. Note that the critical decision in such a top-down decision-tree-generation algorithm is the choice of attribute at a node. Attribute selection in ID3 and C4.5 algorithms is based on minimizing an information entropy measure applied to the examples at a node. The approach based on information entropy insists on minimizing the number of tests that will allow a sample to classify in a database. The attribute selection part of ID3 is based on the assumption that the complexity of the decision tree is strongly related to the amount of information conveyed by the value of the given attribute. An information-based heuristic selects the attribute providing the highest information gain, i.e., the attribute that minimizes the information needed in the resulting subtree to classify the sample. An extension of ID3 is the C4.5 algorithm, which extends the domain of classification from categorical attributes to numeric ones. The measure favors attributes that result in partitioning the data into subsets that have low class entropy, i.e., when the majority of examples in it belong to a single class. The algorithm basically chooses the attribute that provides the maximum degree of discrimination between classes locally. More details about basic principles and implementation of these algorithms will be given in the following sections.

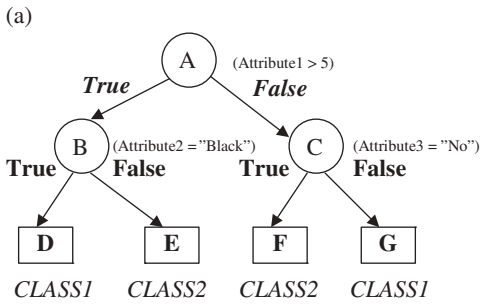
To apply some of the methods, which are based on the inductive-learning approach, several key requirements have to be satisfied:

1. *Attribute-value description*—The data to be analyzed must be in a flat-file form—all information about one object or example must be expressible in terms of a fixed collection of properties or attributes. Each attribute may have either discrete or numeric values, but the attributes used to describe samples must not vary from one case to another. This restriction rules out domains in which samples have an inherently variable structure.
2. *Predefined classes*—The categories to which samples are to be assigned must have been established beforehand. In the terminology of machine learning, this is supervised learning.
3. *Discrete classes*—The classes must be sharply delineated: a case either does or does not belong to a particular class. It is expected that there will be far more samples than classes.
4. *Sufficient data*—Inductive generalization given in the form of decision tree proceeds by identifying patterns in data. The approach is valid if enough number of robust patterns can be distinguished from chance coincidences. As this differentiation usually depends on statistical tests, there must be sufficient number of samples to allow these tests to be effective. The amount of data required is affected by factors such as the number of properties and classes and the complexity of the classification model. As these factors increase, more data will be needed to construct a reliable model.
5. *"Logical" classification models*—These methods construct only such classifiers that can be expressed as decision trees or decision rules. These forms essentially restrict the description of a class to a logical expression whose primitives are statements about the values of particular attributes. Some applications require weighted attributes or their arithmetic combinations for a reliable description of classes. In these situations logical models become very complex, and, in general, they are not effective.

6.2 *C4.5 ALGORITHM*: GENERATING A DECISION TREE

The most important part of the C4.5 algorithm is the process of generating an initial decision tree from the set of training samples. As a result, the algorithm generates a classifier in the form of a decision tree; a structure with two types of nodes: *a leaf*, indicating a class, or *a decision node* that specifies some test to be carried out on a single-attribute value, with one branch and subtree for each possible outcome of the test.

A decision tree can be used to classify a new sample by starting at the root of the tree and moving through it until a leaf is encountered. At each nonleaf decision node, the features' outcome for the test at the node is determined and



(b)

Attribute	Value
Attribute1	***5***
Attribute2	Black
Attribute3	**No**

Figure 6.3. Classification of a new sample based on the decision-tree model. (a) Decision tree. (b) An example for classification.

attention shifts to the root of the selected subtree. For example, if the classification model of the problem is given with the decision tree in Figure 6.3a, and the sample for classification in Figure 6.3b, then the algorithm will create the path through the nodes **A**, **C**, and **F** (leaf node) until it makes the final classification decision: *CLASS2*.

The skeleton of the C4.5 algorithm is based on Hunt's *CLS* method for constructing a decision tree from a set T of training samples. Let the classes be denoted as $\{C_1, C_2,\ldots,C_k\}$. There are three possibilities for the content of the set T:

1. T contains one or more samples, all belonging to a single class C_j. The decision tree for T is a leaf-identifying class C_j.
2. T contains no samples. The decision tree is again a leaf, but the class to be associated with the leaf must be determined from information other than T, such as the overall majority class in T. The C4.5 algorithm uses as a criterion the most frequent class at the parent of the given node.
3. T contains samples that belong to a mixture of classes. In this situation, the idea is to refine T into subsets of samples that are heading toward a single-class collection of samples. Based on single attribute, an appropriate test that has one or more mutually exclusive outcomes $\{O_1, O_2,\ldots,O_n\}$ is chosen. T is partitioned into subsets $T_1, T_2,\ldots,T_n$ where T_i contains all the samples in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test and one branch for each possible outcome (examples of this type of nodes are nodes A, B, and C in the decision tree in Figure 6.3a).

The same tree-building procedure is applied recursively to each subset of training samples, so that the ith branch leads to the decision tree constructed from the subset T_i of training samples. The successive division of the set of training samples proceeds until all the subsets consist of samples belonging to a single class.

The tree-building process is not uniquely defined. For different tests, even for a different order of their application, different trees will be generated. Ideally, we would like to choose a test at each stage of sample-set splitting so that the final tree is small. Since we are looking for a compact decision tree that is consistent with the training set, why not explore all possible trees and select the simplest? Unfortunately, the problem of finding the smallest decision tree consistent with a training data set is NP-complete. Enumeration and analysis of all possible trees will cause a combinatorial explosion for any real-world problem. For example, for a small database with five attributes and only twenty training examples, the possible number of decision trees is greater than 10^6, depending on the number of different values for every attribute. Therefore, most decision-tree construction methods are non-backtracking, greedy algorithms. Once a test has been selected using some heuristics to maximize the measure of progress and the current set of training cases has been partitioned, the consequences of alternative choices are not explored. The measure of progress is a local measure, and the gain criterion for a test selection is based on the information available for a given step of data splitting.

Suppose we have the task of selecting a possible test with n outcomes (n values for a given feature) that partitions the set T of training samples into subsets T_1, T_2,…, T_n. The only information available for guidance is the distribution of classes in T and its subsets T_i. If S is any set of samples, let freq(C_i, S) stand for the number of samples in S that belong to class C_i (out of k possible classes), and let $|S|$ denote the number of samples in the set S.

The original ID3 algorithm used a criterion called *gain* to select the attribute to be tested that is based on the information theory concept: *entropy*. The following relation gives the computation of the entropy of the set T (bits are units):

$$\text{Info}(T) = -\sum_{i=1}^{k} \left(\left(\frac{\text{freq}(C_i, T)}{|T|} \right) \cdot \log_2 \left(\frac{\text{freq}(C_i, T)}{|T|} \right) \right)$$

Now consider a similar measurement after T has been partitioned in accordance with n outcomes of one attribute test X. The expected information requirement can be found as the weighted sum of entropies over the subsets:

$$\text{Info}_x(T) = \sum_{i=1}^{n} \left(\left(\frac{|T_i|}{|T|} \right) \cdot \text{Info}(T_i) \right)$$

The quantity

$$\text{Gain}(X) = \text{Info}(T) - \text{Info}_x(T)$$

measures the information that is gained by partitioning T in accordance with the test X. The gain criterion selects a test X to maximize Gain(X), i.e., this criterion will select an attribute with the highest info-gain.

TABLE 6.1. A Simple Flat Database of Examples for Training

Database T:			
Attribute1	Attribute2	Attribute3	Class
A	70	True	CLASS1
A	90	True	CLASS2
A	85	False	CLASS2
A	95	False	CLASS2
A	70	False	CLASS1
B	90	True	CLASS1
B	78	False	CLASS1
B	65	True	CLASS1
B	75	False	CLASS1
C	80	True	CLASS2
C	70	True	CLASS2
C	80	False	CLASS1
C	80	False	CLASS1
C	96	False	CLASS1

Let us analyze the application of these measures and the creation of a decision tree for one simple example. Suppose that the database T is given in a flat form in which each out of fourteen examples (cases) is described by three input attributes and belongs to one of two given classes: CLASS1 or CLASS2. The database is given in tabular form in Table 6.1.

Nine samples belong to CLASS1 and five samples to CLASS2, so the entropy before splitting is

$$\text{Info}(T) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits}$$

After using Attribute1 to divide the initial set of samples T into three subsets (test x_1 represents the selection one of three values A, B, or C), the resulting information is given by

$$\begin{aligned}\text{Info}_{x1}(T) = {} & \frac{5}{14}\left(-\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right) \\ & + \frac{4}{14}\left(-\frac{4}{4}\log_2\left(\frac{4}{4}\right) - \frac{0}{4}\log_2\left(\frac{0}{4}\right)\right) \\ & + \frac{5}{14}\left(-\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{5}{5}\right)\right) \\ = {} & 0.694 \text{ bits}\end{aligned}$$

The information gained by this test x_1 is

$$\text{Gain}(x_1) = 0.940 - 0.694 = 0.246 \text{ bits}$$

If the test and splitting is based on Attribute3 (test x_2 represents the selection one of two values True or False), a similar computation will give new results:

$$\begin{aligned}\text{Info}_{x2}(T) = &\frac{6}{14}\left(-\frac{3}{6}\log_2\left(\frac{3}{6}\right)-\frac{3}{6}\log_2\left(\frac{3}{6}\right)\right)\\ &+\frac{8}{14}\left(-\frac{6}{8}\log_2\left(\frac{6}{8}\right)-\frac{2}{8}\log_2\left(\frac{2}{8}\right)\right)\\ &= 0.892 \text{ bits}\end{aligned}$$

and corresponding gain is

$$\text{Gain}\,(x_2) = 0.940 - 0.892 = 0.048 \text{ bits}$$

Based on the gain criterion, the decision-tree algorithm will select test x_1 as an initial test for splitting the database T because this gain is higher. To find the optimal test, it will be necessary to analyze a test on Attribute2, which is a numeric feature with continuous values. In general, C4.5 contains mechanisms for proposing three types of tests:

1. The "standard" test on a discrete attribute, with one outcome and one branch for each possible value of that attribute (in our example these are both tests x_1 for Attribute1 and x_2 for Attribute3).
2. If attribute Y has continuous numeric values, a binary test with outcomes $Y \le Z$ and $Y > Z$ could be defined, by comparing its value against a threshold value Z.
3. A more complex test also based also on a discrete attribute, in which the possible values are allocated to a variable number of groups with one outcome and branch for each group.

While we have already explained standard test for categorical attributes, additional explanations are necessary about a procedure for establishing tests on attributes with numeric values. It might seem that tests on continuous attributes would be difficult to formulate, since they contain an arbitrary threshold for splitting all values into two intervals. But there is an algorithm for the computation of optimal threshold value Z. The training samples are first sorted on the values of the attribute Y being considered. There are only a finite number of these values, so let us denote them in sorted order as $\{v_1, v_2,\ldots,v_m\}$. Any threshold value lying between v_i and v_{i+1} will have the same effect as dividing the cases into those whose value of the attribute Y lies in $\{v_1, v_2,\ldots,v_i\}$ and those whose value is in $\{v_{i+1}, v_{i+2},\ldots,v_m\}$. There are thus only $m-1$ possible splits on Y, all of which should be examined systematically to obtain an optimal split. It is usual to choose the midpoint of each interval, $(v_i + v_{i+1})/2$, as the

representative threshold. The algorithm C4.5 differs in choosing as the threshold a smaller value v_i for every interval $\{v_i, v_{i+1}\}$, rather than the midpoint itself. This ensures that the threshold values appearing in either the final decision tree or rules or both actually occur in the database.

To illustrate this threshold-finding process, we could analyze, for our example of database T, the possibilities of Attribute2 splitting. After a sorting process, the set of values for Attribute2 is {65, 70, 75, 78, 80, 85, 90, 95, 96}, and the set of potential threshold values Z is {65, 70, 75, 78, 80, 85, 90, 95}. Out of these eight values, the optimal Z (with the highest information gain) should be selected. For our example, the optimal Z value is $Z = 80$, and the corresponding process of information-gain computation for the test x_3 (Attribute 2 ≤ 80 or Attribute 2 > 80) is the following:

$$\text{Info}_{x3}(T) = \frac{9}{14}\left(-\frac{7}{9}\log_2\left(\frac{7}{9}\right) - \frac{2}{9}\log_2\left(\frac{2}{9}\right)\right) + \frac{5}{14}\left(-\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right)$$

$$= 0.837 \text{ bits}$$

$$\text{Gain}(x_3) = 0.940 - 0.837 = 0.103 \text{ bits}$$

Now, if we compare the information gain for the three attributes in our example, we can see that Attribute1 still gives the highest gain of 0.246 bits, and therefore this attribute will be selected for the first splitting in the construction of a decision tree. The root node will have the test for the values of Attribute1, and three branches will be created, one for each of the attribute values. This initial tree with the corresponding subsets of samples in the children nodes is represented in Figure 6.4.

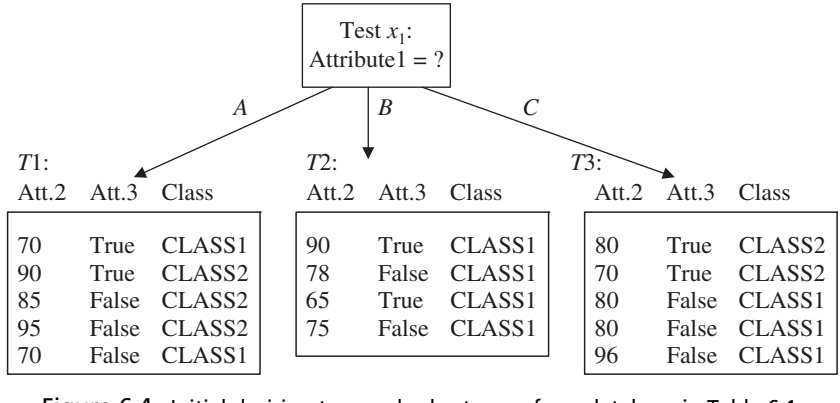


Figure 6.4. Initial decision tree and subset cases for a database in Table 6.1.

After initial splitting, every child node has several samples from the database, and the entire process of test selection and optimization will be repeated for every child node. Because the child node for test x_1: Attribute1 = *B* has four cases and all of them are in CLASS1, this node will be the leaf node, and no additional tests are necessary for this branch of the tree.

For the remaining child node where we have five cases in subset T_1, tests on the remaining attributes can be performed; an optimal test (with maximum information gain) will be test x_4 with two alternatives: Attribute 2 ≤ 70 or Attribute 2 > 70:

$$\text{Info}(T_1) = -\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) = 0.97 \text{ bits}$$

Using Attribute 2 to divide T_1 into two subsets (test x_4 represents the selection of one of two intervals), the resulting information is given by

$$\begin{aligned}\text{Info}_{x4}(T_1) = &\frac{2}{5}\left(-\frac{2}{2}\log_2\left(\frac{2}{2}\right) - \frac{0}{2}\log_2\left(\frac{0}{2}\right)\right) \\ &+ \frac{3}{5}\left(-\frac{0}{3}\log_2\left(\frac{0}{3}\right) - \frac{3}{3}\log_2\left(\frac{3}{3}\right)\right) \\ = &\ 0 \text{ bits}\end{aligned}$$

The information gained by this test is maximal:

$$\text{Gain}\,(x_4) = 0.97 - 0 = 0.97 \text{ bits}$$

and two branches will create the final leaf nodes because the subsets of cases in each of the branches belong to the same class.

A similar computation will be carried out for the third child of the root node. For the subset T_3 of the database *T*, the selected optimal test x_5 is the test on Attribute 3 values. Branches of the tree, Attribute 3 = True and Attribute 3 = False, will create uniform subsets of cases, which belong to the same class. The final decision tree for database *T* is represented in Figure 6.5.

Alternatively, a decision tree can be presented in the form of an executable code (or pseudocode) with *if-then* constructions for branching into a tree structure. The transformation of a decision tree from one representation to the other is very simple and straightforward. The final decision tree for our example is given in pseudocode in Figure 6.6.

While the gain criterion has had some good results in the construction of compact decision trees, it also has one serious deficiency: a strong bias in favor of tests with many outcomes. A solution was found in some kinds of normalization. By analogy with the definition of Info(*S*), an additional parameter was specified:

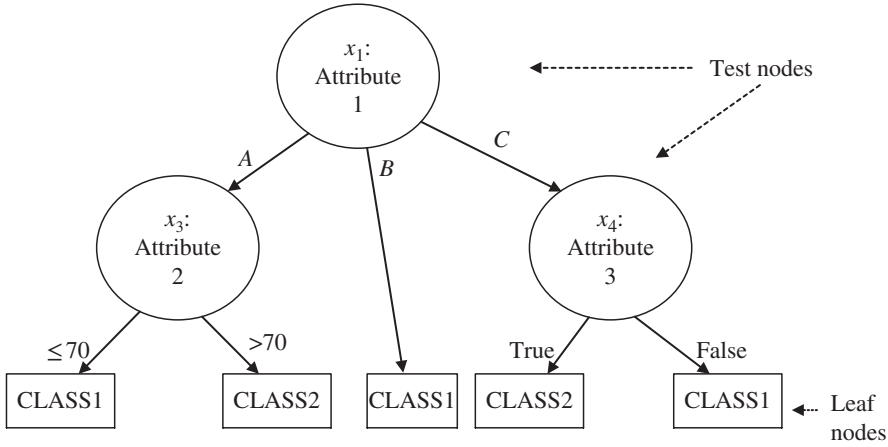


Figure 6.5. A final decision tree for database *T* given in Table 6.1.

```
If      Attribute1 = A
        Then
                If      Attribute2 <= 70
                        Then
                                Classification = CLASS1;
                        Else
                                Classification = CLASS2;
Elseif  Attribute1 = B
        Then
                                Classification = CLASS1;
Elseif  Attribute1 = C
        Then
                If      Attribute3 = True
                        Then
                                Classification = CLASS2;
                        Else
                                Classification = CLASS1.
```

Figure 6.6. A decision tree in the form of pseudocode for the database *T* given in Table 6.1.

$$\text{Split-info}(X) = -\sum_{i=1}^{n}\left(\left(\frac{|T_i|}{|T|}\right)\log_2\left(\frac{|T_i|}{|T|}\right)\right)$$

This represented the potential information generated by dividing set T into n subsets T_i. Now, a new gain measure could be defined:

$$\text{Gain-ratio}(X) = \frac{\text{gain}(X)}{\text{Split-info}(X)}$$

This new gain measure expresses the proportion of information generated by the split that is useful, i.e. that appears helpful in classification. The gain-ratio criterion also selects a test that maximizes the ratio given earlier. This criterion is robust and typically gives a consistently better choice of a test than the previous gain criterion. A computation of the gain-ratio test can be illustrated for our example. To find the gain-ratio measure for the test x_1, an additional parameter Split-info(x_1) is calculated:

$$\text{Split-info}\,(x_1) = -\frac{5}{14}\log_2\left(\frac{5}{14}\right) - \frac{4}{14}\log_2\left(\frac{4}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 1.577\ \text{bits}$$

$$\text{Gain-ratio}(x_1) = \frac{0.246}{1.557} = 0.156$$

A similar procedure should be performed for other tests in the decision tree. Instead of gain measure, the maximal gain ratio will be the criterion for attribute selection, along with a test to split samples into subsets. The final decision tree created using this new criterion for splitting a set of samples will be the most compact.

6.3 UNKNOWN ATTRIBUTE VALUES

The previous version of the C4.5 algorithm is based on the assumption that all values for all attributes are determined. But in a data set, often some attribute values for some samples can be missing—such incompleteness is typical in real-world applications. This might occur because the value is not relevant to a particular sample, or it was not recorded when the data was collected, or an error was made by the person the entering data into a database. To solve the problem of missing values, there are two choices:

1. Discard all samples in a database with missing data.
2. Define a new algorithm or modify an existing algorithm that will work with missing data.