2

Data

This chapter discusses several data-related issues that are important for successful data mining:

The Type of Data Data sets differ in a number of ways. For example, the attributes used to describe data objects can be of different types—quantitative or qualitative—and data sets often have special characteristics; e.g., some data sets contain time series or objects with explicit relationships to one another. Not surprisingly, the type of data determines which tools and techniques can be used to analyze the data. Indeed, new research in data mining is often driven by the need to accommodate new application areas and their new types of data.

The Quality of the Data Data is often far from perfect. While most data mining techniques can tolerate some level of imperfection in the data, a focus on understanding and improving data quality typically improves the quality of the resulting analysis. Data quality issues that often need to be addressed include the presence of noise and outliers; missing, inconsistent, or duplicate data; and data that is biased or, in some other way, unrepresentative of the phenomenon or population that the data is supposed to describe.

Preprocessing Steps to Make the Data More Suitable for Data Mining Often, the raw data must be processed in order to make it suitable for analysis. While one objective may be to improve data quality, other goals focus on modifying the data so that it better fits a specified data mining technique or tool. For example, a continuous attribute, e.g., length, sometimes needs to be transformed into an attribute with discrete categories, e.g., *short*, *medium*, or *long*, in order to apply a particular technique. As another example, the

number of attributes in a data set is often reduced because many techniques are more effective when the data has a relatively small number of attributes.

Analyzing Data in Terms of Its Relationships One approach to data analysis is to find relationships among the data objects and then perform the remaining analysis using these relationships rather than the data objects themselves. For instance, we can compute the similarity or distance between pairs of objects and then perform the analysis—clustering, classification, or anomaly detection—based on these similarities or distances. There are many such similarity or distance measures, and the proper choice depends on the type of data and the particular application.

Example 2.1 (An Illustration of Data-Related Issues)**.** To further illustrate the importance of these issues, consider the following hypothetical situation. You receive an email from a medical researcher concerning a project that you are eager to work on.

> Hi,
>
> I've attached the data file that I mentioned in my previous email. Each line contains the information for a single patient and consists of five fields. We want to predict the last field using the other fields. I don't have time to provide any more information about the data since I'm going out of town for a couple of days, but hopefully that won't slow you down too much. And if you don't mind, could we meet when I get back to discuss your preliminary results? I might invite a few other members of my team.
>
> Thanks and see you in a couple of days.

Despite some misgivings, you proceed to analyze the data. The first few rows of the file are as follows:

```
012   232   33.5   0    10.7
020   121   16.9   2   210.1
027   165   24.0   0   427.6
⋮
```

A brief look at the data reveals nothing strange. You put your doubts aside and start the analysis. There are only 1000 lines, a smaller data file than you had hoped for, but two days later, you feel that you have made some progress. You arrive for the meeting, and while waiting for others to arrive, you strike

up a conversation with a statistician who is working on the project. When she learns that you have also been analyzing the data from the project, she asks if you would mind giving her a brief overview of your results.

Statistician: So, you got the data for all the patients?

Data Miner: Yes. I haven't had much time for analysis, but I do have a few interesting results.

Statistician: Amazing. There were so many data issues with this set of patients that I couldn't do much.

Data Miner: Oh? I didn't hear about any possible problems.

Statistician: Well, first there is field 5, the variable we want to predict. It's common knowledge among people who analyze this type of data that results are better if you work with the log of the values, but I didn't discover this until later. Was it mentioned to you?

Data Miner: No.

Statistician: But surely you heard about what happened to field 4? It's supposed to be measured on a scale from 1 to 10, with 0 indicating a missing value, but because of a data entry error, all 10's were changed into 0's. Unfortunately, since some of the patients have missing values for this field, it's impossible to say whether a 0 in this field is a real 0 or a 10. Quite a few of the records have that problem.

Data Miner: Interesting. Were there any other problems?

Statistician: Yes, fields 2 and 3 are basically the same, but I assume that you probably noticed that.

Data Miner: Yes, but these fields were only weak predictors of field 5.

Statistician: Anyway, given all those problems, I'm surprised you were able to accomplish anything.

Data Miner: True, but my results are really quite good. Field 1 is a very strong predictor of field 5. I'm surprised that this wasn't noticed before.

Statistician: What? Field 1 is just an identification number.

Data Miner: Nonetheless, my results speak for themselves.

Statistician: Oh, no! I just remembered. We assigned ID numbers after we sorted the records based on field 5. There is a strong connection, but it's meaningless. Sorry.

■

Although this scenario represents an extreme situation, it emphasizes the importance of "knowing your data." To that end, this chapter will address each of the four issues mentioned above, outlining some of the basic challenges and standard approaches.

2.1 Types of Data

A **data set** can often be viewed as a collection of **data objects**. Other names for a data object are *record*, *point*, *vector*, *pattern*, *event*, *case*, *sample*, *instance*, *observation*, or *entity*. In turn, data objects are described by a number of **attributes** that capture the characteristics of an object, such as the mass of a physical object or the time at which an event occurred. Other names for an attribute are *variable*, *characteristic*, *field*, *feature*, or *dimension*.

Example 2.2 (Student Information)**.** Often, a data set is a file, in which the objects are records (or rows) in the file and each field (or column) corresponds to an attribute. For example, Table 2.1 shows a data set that consists of student information. Each row corresponds to a student and each column is an attribute that describes some aspect of a student, such as grade point average (GPA) or identification number (ID).

Table 2.1. A sample data set containing student information.

Student ID	Year	Grade Point Average (GPA)	...
	⋮		
1034262	Senior	3.24	...
1052663	Sophomore	3.51	...
1082246	Freshman	3.62	...
	⋮		

■

Although record-based data sets are common, either in flat files or relational database systems, there are other important types of data sets and systems for storing data. In Section 2.1.2, we will discuss some of the types of data sets that are commonly encountered in data mining. However, we first consider attributes.

2.1.1 Attributes and Measurement

In this section, we consider the types of attributes used to describe data objects. We first define an attribute, then consider what we mean by the type of an attribute, and finally describe the types of attributes that are commonly encountered.

What Is an Attribute?

We start with a more detailed definition of an attribute.

Definition 2.1. An **attribute** is a property or characteristic of an object that can vary, either from one object to another or from one time to another.

For example, eye color varies from person to person, while the temperature of an object varies over time. Note that eye color is a symbolic attribute with a small number of possible values $\{brown, black, blue, green, hazel, etc.\}$, while temperature is a numerical attribute with a potentially unlimited number of values.

At the most basic level, attributes are not about numbers or symbols. However, to discuss and more precisely analyze the characteristics of objects, we assign numbers or symbols to them. To do this in a well-defined way, we need a measurement scale.

Definition 2.2. A **measurement scale** is a rule (function) that associates a numerical or symbolic value with an attribute of an object.

Formally, the process of **measurement** is the application of a measurement scale to associate a value with a particular attribute of a specific object. While this may seem a bit abstract, we engage in the process of measurement all the time. For instance, we step on a bathroom scale to determine our weight, we classify someone as male or female, or we count the number of chairs in a room to see if there will be enough to seat all the people coming to a meeting. In all these cases, the "physical value" of an attribute of an object is mapped to a numerical or symbolic value.

With this background, we can discuss the type of an attribute, a concept that is important in determining if a particular data analysis technique is consistent with a specific type of attribute.

The Type of an Attribute

It is common to refer to the type of an attribute as the **type of a measurement scale**. It should be apparent from the previous discussion that an

attribute can be described using different measurement scales and that the properties of an attribute need not be the same as the properties of the values used to measure it. In other words, the values used to represent an attribute can have properties that are not properties of the attribute itself, and vice versa. This is illustrated with two examples.

Example 2.3 (Employee Age and ID Number)**.** Two attributes that might be associated with an employee are *ID* and *age* (in years). Both of these attributes can be represented as integers. However, while it is reasonable to talk about the average age of an employee, it makes no sense to talk about the average employee ID. Indeed, the only aspect of employees that we want to capture with the ID attribute is that they are distinct. Consequently, the only valid operation for employee IDs is to test whether they are equal. There is no hint of this limitation, however, when integers are used to represent the employee ID attribute. For the age attribute, the properties of the integers used to represent age are very much the properties of the attribute. Even so, the correspondence is not complete because, for example, ages have a maximum, while integers do not. ■

Example 2.4 (Length of Line Segments)**.** Consider Figure 2.1, which shows some objects—line segments—and how the length attribute of these objects can be mapped to numbers in two different ways. Each successive line segment, going from the top to the bottom, is formed by appending the topmost line segment to itself. Thus, the second line segment from the top is formed by appending the topmost line segment to itself twice, the third line segment from the top is formed by appending the topmost line segment to itself three times, and so forth. In a very real (physical) sense, all the line segments are multiples of the first. This fact is captured by the measurements on the right side of the figure, but not by those on the left side. More specifically, the measurement scale on the left side captures only the ordering of the length attribute, while the scale on the right side captures both the ordering and additivity properties. Thus, an attribute can be measured in a way that does not capture all the properties of the attribute. ■

Knowing the type of an attribute is important because it tells us which properties of the measured values are consistent with the underlying properties of the attribute, and therefore, it allows us to avoid foolish actions, such as computing the average employee ID.

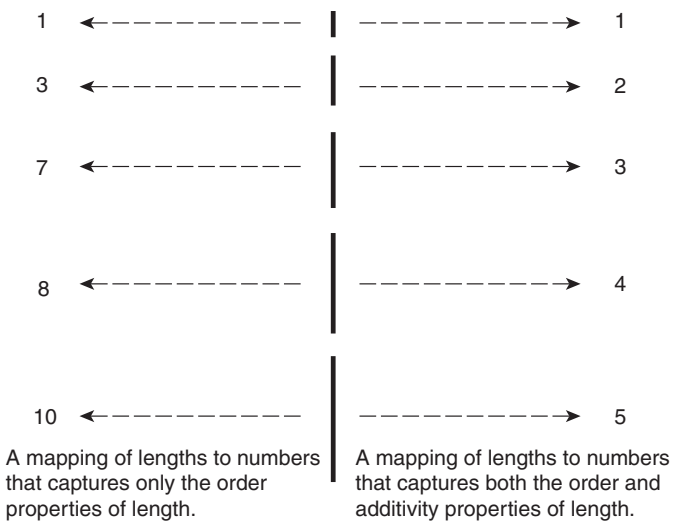


Figure 2.1. The measurement of the length of line segments on two different scales of measurement.

The Different Types of Attributes

A useful (and simple) way to specify the type of an attribute is to identify the properties of numbers that correspond to underlying properties of the attribute. For example, an attribute such as length has many of the properties of numbers. It makes sense to compare and order objects by length, as well as to talk about the differences and ratios of length. The following properties (operations) of numbers are typically used to describe attributes.

1. **Distinctness** $=$ and $\neq$
2. **Order** $<$, $\leq$, $>$, and $\geq$
3. **Addition** $+$ and $-$
4. **Multiplication** $\times$ and $/$

Given these properties, we can define four types of attributes: **nominal**, **ordinal**, **interval**, and **ratio**. Table 2.2 gives the definitions of these types, along with information about the statistical operations that are valid for each type. Each attribute type possesses all of the properties and operations of the attribute types above it. Consequently, any property or operation that is valid for nominal, ordinal, and interval attributes is also valid for ratio attributes. In other words, the definition of the attribute types is cumulative. However,

Table 2.2. Different attribute types.

Attribute Type		Description	Examples	Operations
Categorical (Qualitative)	Nominal	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another. $(=, \neq)$	zip codes, employee ID numbers, eye color, gender	mode, entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. $(<, >)$	hardness of minerals, $\{good, better, best\}$, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. $(+, -)$	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviation, Pearson's correlation, t and F tests
	Ratio	For ratio variables, both differences and ratios are meaningful. $(\times, /)$	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

this does not mean that the statistical operations appropriate for one attribute type are appropriate for the attribute types above it.

Nominal and ordinal attributes are collectively referred to as **categorical** or **qualitative** attributes. As the name suggests, qualitative attributes, such as employee ID, lack most of the properties of numbers. Even if they are represented by numbers, i.e., integers, they should be treated more like symbols. The remaining two types of attributes, interval and ratio, are collectively referred to as **quantitative** or **numeric** attributes. Quantitative attributes are represented by numbers and have most of the properties of numbers. Note that quantitative attributes can be integer-valued or continuous.

The types of attributes can also be described in terms of transformations that do not change the meaning of an attribute. Indeed, S. Smith Stevens, the

Table 2.3. Transformations that define attribute levels.

Attribute Type		Transformation	Comment
Categorical (Qualitative)	Nominal	Any one-to-one mapping, e.g., a permutation of values	If all employee ID numbers are reassigned, it will not make any difference.
	Ordinal	An order-preserving change of values, i.e., $new\_value = f(old\_value)$, where f is a monotonic function.	An attribute encompassing the notion of good, better, best can be represented equally well by the values $\{1, 2, 3\}$ or by $\{0.5, 1, 10\}$.
Numeric (Quantitative)	Interval	$new\_value = a \times old\_value + b$, a and b constants.	The Fahrenheit and Celsius temperature scales differ in the location of their zero value and the size of a degree (unit).
	Ratio	$new\_value = a \times old\_value$	Length can be measured in meters or feet.

psychologist who originally defined the types of attributes shown in Table 2.2, defined them in terms of these **permissible transformations**. For example, the meaning of a length attribute is unchanged if it is measured in meters instead of feet.

The statistical operations that make sense for a particular type of attribute are those that will yield the same results when the attribute is transformed by using a transformation that preserves the attribute's meaning. To illustrate, the average length of a set of objects is different when measured in meters rather than in feet, but both averages represent the same length. Table 2.3 shows the meaning-preserving transformations for the four attribute types of Table 2.2.

Example 2.5 (Temperature Scales)**.** Temperature provides a good illustration of some of the concepts that have been described. First, temperature can be either an interval or a ratio attribute, depending on its measurement scale. When measured on the Kelvin scale, a temperature of 2° is, in a physically meaningful way, twice that of a temperature of 1°. This is not true when temperature is measured on either the Celsius or Fahrenheit scales, because, physically, a temperature of 1° Fahrenheit (Celsius) is not much different than a temperature of 2° Fahrenheit (Celsius). The problem is that the zero points of the Fahrenheit and Celsius scales are, in a physical sense, arbitrary, and

therefore, the ratio of two Celsius or Fahrenheit temperatures is not physically meaningful. ■

Describing Attributes by the Number of Values

An independent way of distinguishing between attributes is by the number of values they can take.

Discrete A discrete attribute has a finite or countably infinite set of values. Such attributes can be categorical, such as zip codes or ID numbers, or numeric, such as counts. Discrete attributes are often represented using integer variables. **Binary attributes** are a special case of discrete attributes and assume only two values, e.g., true/false, yes/no, male/female, or 0/1. Binary attributes are often represented as Boolean variables, or as integer variables that only take the values 0 or 1.

Continuous A continuous attribute is one whose values are real numbers. Examples include attributes such as temperature, height, or weight. Continuous attributes are typically represented as floating-point variables. Practically, real values can be measured and represented only with limited precision.

In theory, any of the measurement scale types—nominal, ordinal, interval, and ratio—could be combined with any of the types based on the number of attribute values—binary, discrete, and continuous. However, some combinations occur only infrequently or do not make much sense. For instance, it is difficult to think of a realistic data set that contains a continuous binary attribute. Typically, nominal and ordinal attributes are binary or discrete, while interval and ratio attributes are continuous. However, **count attributes**, which are discrete, are also ratio attributes.

Asymmetric Attributes

For asymmetric attributes, only presence—a non-zero attribute value—is regarded as important. Consider a data set in which each object is a student and each attribute records whether a student took a particular course at a university. For a specific student, an attribute has a value of 1 if the student took the course associated with that attribute and a value of 0 otherwise. Because students take only a small fraction of all available courses, most of the values in such a data set would be 0. Therefore, it is more meaningful and more efficient to focus on the non-zero values. To illustrate, if students are compared

on the basis of the courses they don't take, then most students would seem very similar, at least if the number of courses is large. Binary attributes where only non-zero values are important are called **asymmetric binary attributes**. This type of attribute is particularly important for association analysis, which is discussed in Chapter 4. It is also possible to have discrete or continuous asymmetric features. For instance, if the number of credits associated with each course is recorded, then the resulting data set will consist of **asymmetric discrete** or **continuous attributes**.

General Comments on Levels of Measurement

As described in the rest of this chapter, there are many diverse types of data. The previous discussion of measurement scales, while useful, is not complete and has some limitations. We provide the following comments and guidance.

- **Distinctness, order, and meaningful intervals and ratios are only four properties of data—many others are possible**. For instance, some data is inherently cyclical, e.g., position on the surface of the Earth or time. As another example, consider set valued attributes, where each attribute value is a set of elements, e.g., the set of movies seen in the last year. Define one set of elements (movies) to be greater (larger) than a second set if the second set is a subset of the first. However, such a relationship defines only a partial order that does not match any of the attribute types just defined.

- **The numbers or symbols used to capture attribute values may not capture all the properties of the attributes or may suggest properties that are not there**. An illustration of this for integers was presented in Example 2.3, i.e., averages of IDs and out of range ages.

- **Data is often transformed for the purpose of analysis**—see Section 2.3.7. This often changes the distribution of the observed variable to a distribution that is easier to analyze, e.g., a Gaussian (normal) distribution. Often, such transformations only preserve the order of the original values, and other properties are lost. Nonetheless, if the desired outcome is a statistical test of differences or a predictive model, such a transformation is justified.

- **The final evaluation of any data analysis, including operations on attributes, is whether the results make sense from a domain point of view**.

In summary, it can be challenging to determine which operations can be performed on a particular attribute or a collection of attributes without compromising the integrity of the analysis. Fortunately, established practice often serves as a reliable guide. Occasionally, however, standard practices are erroneous or have limitations.

2.1.2 Types of Data Sets

There are many types of data sets, and as the field of data mining develops and matures, a greater variety of data sets become available for analysis. In this section, we describe some of the most common types. For convenience, we have grouped the types of data sets into three groups: record data, graph-based data, and ordered data. These categories do not cover all possibilities and other groupings are certainly possible.

General Characteristics of Data Sets

Before providing details of specific kinds of data sets, we discuss three characteristics that apply to many data sets and have a significant impact on the data mining techniques that are used: dimensionality, distribution, and resolution.

Dimensionality The dimensionality of a data set is the number of attributes that the objects in the data set possess. Analyzing data with a small number of dimensions tends to be qualitatively different from analyzing moderate or high-dimensional data. Indeed, the difficulties associated with the analysis of high-dimensional data are sometimes referred to as the **curse of dimensionality**. Because of this, an important motivation in preprocessing the data is **dimensionality reduction**. These issues are discussed in more depth later in this chapter and in Appendix B.

Distribution The distribution of a data set is the frequency of occurrence of various values or sets of values for the attributes comprising data objects. Equivalently, the distribution of a data set can be considered as a description of the concentration of objects in various regions of the data space. Statisticians have enumerated many types of distributions, e.g., Gaussian (normal), and described their properties. (See Appendix C.) Although statistical approaches for describing distributions can yield powerful analysis techniques, many data sets have distributions that are not well captured by standard statistical distributions.

As a result, many data mining algorithms do not assume a particular statistical distribution for the data they analyze. However, some general aspects of distributions often have a strong impact. For example, suppose a categorical attribute is used as a class variable, where one of the categories occurs 95% of the time, while the other categories together occur only 5% of the time. This **skewness** in the distribution can make classification difficult as discussed in Section 6.11. (Skewness has other impacts on data analysis that are not discussed here.)

A special case of skewed data is **sparsity**. For sparse binary, count or continuous data, most attributes of an object have values of 0. In many cases, fewer than 1% of the values are non-zero. In practical terms, sparsity is an advantage because usually only the non-zero values need to be stored and manipulated. This results in significant savings with respect to computation time and storage. Indeed, some data mining algorithms, such as the association rule mining algorithms described in Chapter 4, work well only for sparse data. Finally, note that often the attributes in sparse data sets are asymmetric attributes.

Resolution It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For instance, the surface of the Earth seems very uneven at a resolution of a few meters, but is relatively smooth at a resolution of tens of kilometers. The patterns in the data also depend on the level of resolution. If the resolution is too fine, a pattern may not be visible or may be buried in noise; if the resolution is too coarse, the pattern can disappear. For example, variations in atmospheric pressure on a scale of hours reflect the movement of storms and other weather systems. On a scale of months, such phenomena are not detectable.

Record Data

Much data mining work assumes that the data set is a collection of records (data objects), each of which consists of a fixed set of data fields (attributes). See Figure 2.2(a). For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes. Record data is usually stored either in **flat** files or in relational databases. Relational databases are certainly more than a collection of records, but data mining often does not use any of the additional information available in a relational database. Rather, the database serves as

a convenient place to find records. Different types of record data are described below and are illustrated in Figure 2.2.

Tid	Refund	Marital Status	Taxable Income	Defaulted Borrower
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

(a) Record data.

TID	*ITEMS*
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diapers, Milk
4	Beer, Bread, Diapers, Milk
5	Soda, Diapers, Milk

(b) Transaction data.

Projection of x Load	Projection of y Load	Distance	Load	Thickness
10.23	5.27	15.22	27	1.2
12.65	6.25	16.22	22	1.1
13.54	7.23	17.34	23	1.2
14.27	8.43	18.45	25	0.9

(c) Data matrix.

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

(d) Document-term matrix.

Figure 2.2. Different variations of record data.

Transaction or Market Basket Data Transaction data is a special type of record data, where each record (transaction) involves a set of items. Consider a grocery store. The set of products purchased by a customer during one shopping trip constitutes a transaction, while the individual products that were purchased are the items. This type of data is called **market basket data** because the items in each record are the products in a person's "market basket." Transaction data is a collection of sets of items, but it can be viewed as a set of records whose fields are asymmetric attributes. Most often, the attributes are binary, indicating whether an item was purchased, but more

generally, the attributes can be discrete or continuous, such as the number of items purchased or the amount spent on those items. Figure 2.2(b) shows a sample transaction data set. Each row represents the purchases of a particular customer at a particular time.

The Data Matrix If all the data objects in a collection of data have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object. A set of such data objects can be interpreted as an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute. (A representation that has data objects as columns and attributes as rows is also fine.) This matrix is called a **data matrix** or a **pattern matrix**. A data matrix is a variation of record data, but because it consists of numeric attributes, standard matrix operation can be applied to transform and manipulate the data. Therefore, the data matrix is the standard data format for most statistical data. Figure 2.2(c) shows a sample data matrix.

The Sparse Data Matrix A sparse data matrix is a special case of a data matrix where the attributes are of the same type and are asymmetric; i.e., only non-zero values are important. Transaction data is an example of a sparse data matrix that has only 0–1 entries. Another common example is document data. In particular, if the order of the terms (words) in a document is ignored—the "bag of words" approach—then a document can be represented as a term vector, where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation of a collection of documents is often called a **document-term matrix**. Figure 2.2(d) shows a sample document-term matrix. The documents are the rows of this matrix, while the terms are the columns. In practice, only the non-zero entries of sparse data matrices are stored.

Graph-Based Data

A graph can sometimes be a convenient and powerful representation for data. We consider two specific cases: (1) the graph captures relationships among data objects and (2) the data objects themselves are represented as graphs.

Data with Relationships among Objects The relationships among objects frequently convey important information. In such cases, the data is often

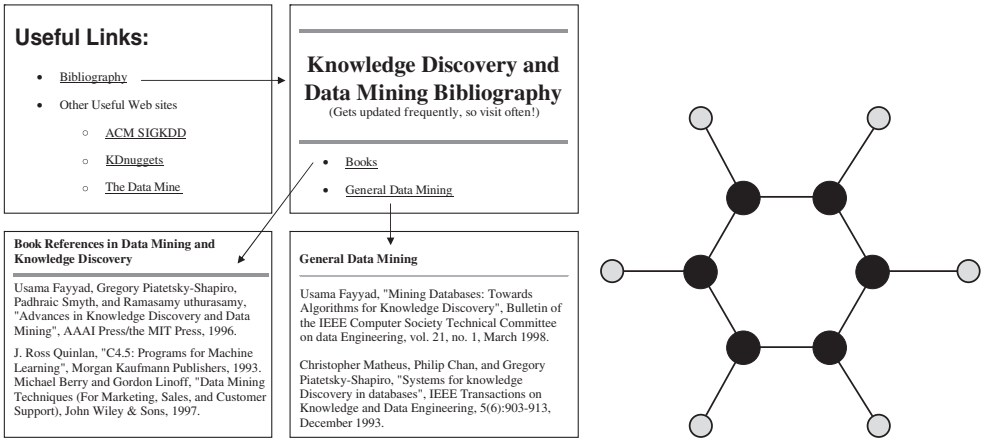
represented as a graph. In particular, the data objects are mapped to nodes of the graph, while the relationships among objects are captured by the links between objects and link properties, such as direction and weight. Consider web pages on the World Wide Web, which contain both text and links to other pages. In order to process search queries, web search engines collect and process web pages to extract their contents. It is well-known, however, that the links to and from each page provide a great deal of information about the relevance of a web page to a query, and thus, must also be taken into consideration. Figure 2.3(a) shows a set of linked web pages. Another important example of such graph data are the social networks, where data objects are people and the relationships among them are their interactions via social media.

Data with Objects That Are Graphs If objects have structure, that is, the objects contain subobjects that have relationships, then such objects are frequently represented as graphs. For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds. Figure 2.3(b) shows a ball-and-stick diagram of the chemical compound benzene, which contains atoms of carbon (black) and hydrogen (gray). A graph representation makes it possible to determine which substructures occur frequently in a set of compounds and to ascertain whether the presence of any of these substructures is associated with the presence or absence of certain chemical properties, such as melting point or heat of formation. Frequent graph mining, which is a branch of data mining that analyzes such data, is considered in Section 7.5.

Ordered Data

For some types of data, the attributes have relationships that involve order in time or space. Different types of ordered data are described next and are shown in Figure 2.4.

Sequential Transaction Data Sequential transaction data can be thought of as an extension of transaction data, where each transaction has a time associated with it. Consider a retail transaction data set that also stores the time at which the transaction took place. This time information makes it possible to find patterns such as "candy sales peak before Halloween." A time can also be associated with each attribute. For example, each record could be the purchase history of a customer, with a listing of items purchased at different times. Using this information, it is possible to find patterns such as



(a) Linked web pages. (b) Benzene molecule.

Figure 2.3. Different variations of graph data.

"people who buy DVD players tend to buy DVDs in the period immediately following the purchase."

Figure 2.4(a) shows an example of sequential transaction data. There are five different times—*t1*, *t2*, *t3*, *t4*, and *t5*; three different customers—C1, C2, and C3; and five different items—A, B, C, D, and E. In the top table, each row corresponds to the items purchased at a particular time by each customer. For instance, at time *t3*, customer C2 purchased items A and D. In the bottom table, the same information is displayed, but each row corresponds to a particular customer. Each row contains information about each transaction involving the customer, where a transaction is considered to be a set of items and the time at which those items were purchased. For example, customer C3 bought items A and C at time *t2*.

Time Series Data Time series data is a special type of ordered data where each record is a **time series**, i.e., a series of measurements taken over time. For example, a financial data set might contain objects that are time series of the daily prices of various stocks. As another example, consider Figure 2.4(c), which shows a time series of the average monthly temperature for Minneapolis during the years 1982 to 1994. When working with temporal data, such as time series, it is important to consider **temporal autocorrelation**; i.e., if

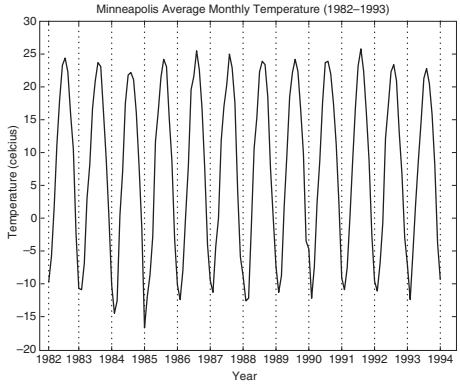
Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t2	C1	C, D
t3	C2	A, D
t4	C2	E
t5	C1	A, E

Customer	Time and Items Purchased
C1	(t1: A,B) (t2:C,D) (t5:A,E)
C2	(t3: A, D) (t4: E)
C3	(t2: A, C)

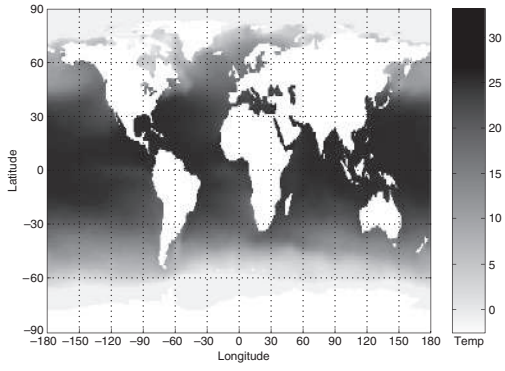
(a) Sequential transaction data.

```
GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG
```

(b) Genomic sequence data.



(c) Temperature time series.



(d) Spatial temperature data.

Figure 2.4. Different variations of ordered data.

two measurements are close in time, then the values of those measurements are often very similar.

Sequence Data Sequence data consists of a data set that is a sequence of individual entities, such as a sequence of words or letters. It is quite similar to sequential data, except that there are no time stamps; instead, there are positions in an ordered sequence. For example, the genetic information of plants and animals can be represented in the form of sequences of nucleotides

that are known as genes. Many of the problems associated with genetic sequence data involve predicting similarities in the structure and function of genes from similarities in nucleotide sequences. Figure 2.4(b) shows a section of the human genetic code expressed using the four nucleotides from which all DNA is constructed: A, T, G, and C.

Spatial and Spatio-Temporal Data Some objects have spatial attributes, such as positions or areas, in addition to other types of attributes. An example of spatial data is weather data (precipitation, temperature, pressure) that is collected for a variety of geographical locations. Often such measurements are collected over time, and thus, the data consists of time series at various locations. In that case, we refer to the data as spatio-temporal data. Although analysis can be conducted separately for each specific time or location, a more complete analysis of spatio-temporal data requires consideration of both the spatial and temporal aspects of the data.

An important aspect of spatial data is **spatial autocorrelation**; i.e., objects that are physically close tend to be similar in other ways as well. Thus, two points on the Earth that are close to each other usually have similar values for temperature and rainfall. Note that spatial autocorrelation is analogous to temporal autocorrelation.

Important examples of spatial and spatio-temporal data are the science and engineering data sets that are the result of measurements or model output taken at regularly or irregularly distributed points on a two- or three-dimensional grid or mesh. For instance, Earth science data sets record the temperature or pressure measured at points (grid cells) on latitude–longitude spherical grids of various resolutions, e.g., 1° by 1°. See Figure 2.4(d). As another example, in the simulation of the flow of a gas, the speed and direction of flow at various instants in time can be recorded for each grid point in the simulation. A different type of spatio-temporal data arises from tracking the trajectories of objects, e.g., vehicles, in time and space.

Handling Non-Record Data

Most data mining algorithms are designed for record data or its variations, such as transaction data and data matrices. Record-oriented techniques can be applied to non-record data by extracting features from data objects and using these features to create a record corresponding to each object. Consider the chemical structure data that was described earlier. Given a set of common substructures, each compound can be represented as a record with binary attributes that indicate whether a compound contains a specific substructure.

Such a representation is actually a transaction data set, where the transactions are the compounds and the items are the substructures.

In some cases, it is easy to represent the data in a record format, but this type of representation does not capture all the information in the data. Consider spatio-temporal data consisting of a time series from each point on a spatial grid. This data is often stored in a data matrix, where each row represents a location and each column represents a particular point in time. However, such a representation does not explicitly capture the time relationships that are present among attributes and the spatial relationships that exist among objects. This does not mean that such a representation is inappropriate, but rather that these relationships must be taken into consideration during the analysis. For example, it would not be a good idea to use a data mining technique that ignores the temporal autocorrelation of the attributes or the spatial autocorrelation of the data objects, i.e., the locations on the spatial grid.

2.2 Data Quality

Data mining algorithms are often applied to data that was collected for another purpose, or for future, but unspecified applications. For that reason, data mining cannot usually take advantage of the significant benefits of "addressing quality issues at the source." In contrast, much of statistics deals with the design of experiments or surveys that achieve a prespecified level of data quality. Because preventing data quality problems is typically not an option, data mining focuses on (1) the detection and correction of data quality problems and (2) the use of algorithms that can tolerate poor data quality. The first step, detection and correction, is often called **data cleaning**.

The following sections discuss specific aspects of data quality. The focus is on measurement and data collection issues, although some application-related issues are also discussed.

2.2.1 Measurement and Data Collection Issues

It is unrealistic to expect that data will be perfect. There may be problems due to human error, limitations of measuring devices, or flaws in the data collection process. Values or even entire data objects can be missing. In other cases, there can be spurious or duplicate objects; i.e., multiple data objects that all correspond to a single "real" object. For example, there might be two different records for a person who has recently lived at two different addresses.

Even if all the data is present and "looks fine," there may be inconsistencies—a person has a height of 2 meters, but weighs only 2 kilograms.

In the next few sections, we focus on aspects of data quality that are related to data measurement and collection. We begin with a definition of measurement and data collection errors and then consider a variety of problems that involve measurement error: noise, artifacts, bias, precision, and accuracy. We conclude by discussing data quality issues that involve both measurement and data collection problems: outliers, missing and inconsistent values, and duplicate data.

Measurement and Data Collection Errors

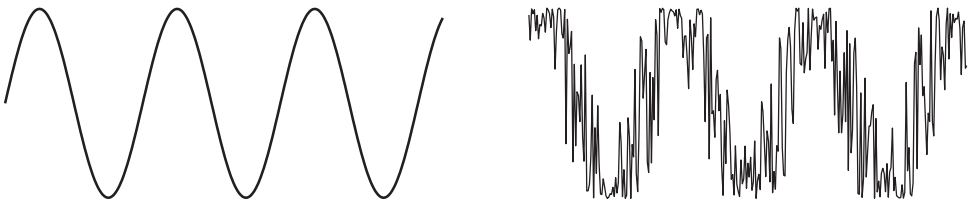
The term **measurement error** refers to any problem resulting from the measurement process. A common problem is that the value recorded differs from the true value to some extent. For continuous attributes, the numerical difference of the measured and true value is called the **error**. The term **data collection error** refers to errors such as omitting data objects or attribute values, or inappropriately including a data object. For example, a study of animals of a certain species might include animals of a related species that are similar in appearance to the species of interest. Both measurement errors and data collection errors can be either systematic or random.

We will only consider general types of errors. Within particular domains, certain types of data errors are commonplace, and well-developed techniques often exist for detecting and/or correcting these errors. For example, keyboard errors are common when data is entered manually, and as a result, many data entry programs have techniques for detecting and, with human intervention, correcting such errors.

Noise and Artifacts

Noise is the random component of a measurement error. It typically involves the distortion of a value or the addition of spurious objects. Figure 2.5 shows a time series before and after it has been disrupted by random noise. If a bit more noise were added to the time series, its shape would be lost. Figure 2.6 shows a set of data points before and after some noise points (indicated by '+'s) have been added. Notice that some of the noise points are intermixed with the non-noise points.

The term noise is often used in connection with data that has a spatial or temporal component. In such cases, techniques from signal or image processing



(a) Time series. (b) Time series with noise.

Figure 2.5. Noise in a time series context.

(a) Three groups of points. (b) With noise points (+) added.

Figure 2.6. Noise in a spatial context.

can frequently be used to reduce noise and thus, help to discover patterns (signals) that might be "lost in the noise." Nonetheless, the elimination of noise is frequently difficult, and much work in data mining focuses on devising **robust algorithms** that produce acceptable results even when noise is present.

Data errors can be the result of a more deterministic phenomenon, such as a streak in the same place on a set of photographs. Such deterministic distortions of the data are often referred to as **artifacts**.

Precision, Bias, and Accuracy

In statistics and experimental science, the quality of the measurement process and the resulting data are measured by precision and bias. We provide the

standard definitions, followed by a brief discussion. For the following definitions, we assume that we make repeated measurements of the same underlying quantity.

Definition 2.3 (Precision)**.** The closeness of repeated measurements (of the same quantity) to one another.

Definition 2.4 (Bias)**.** A systematic variation of measurements from the quantity being measured.

Precision is often measured by the standard deviation of a set of values, while bias is measured by taking the difference between the mean of the set of values and the known value of the quantity being measured. Bias can be determined only for objects whose measured quantity is known by means external to the current situation. Suppose that we have a standard laboratory weight with a mass of 1g and want to assess the precision and bias of our new laboratory scale. We weigh the mass five times, and obtain the following five values: $\{1.015, 0.990, 1.013, 1.001, 0.986\}$. The mean of these values is 1.001, and hence, the bias is 0.001. The precision, as measured by the standard deviation, is 0.013.

It is common to use the more general term, **accuracy**, to refer to the degree of measurement error in data.

Definition 2.5 (Accuracy)**.** The closeness of measurements to the true value of the quantity being measured.

Accuracy depends on precision and bias, but there is no specific formula for accuracy in terms of these two quantities.

One important aspect of accuracy is the use of **significant digits**. The goal is to use only as many digits to represent the result of a measurement or calculation as are justified by the precision of the data. For example, if the length of an object is measured with a meter stick whose smallest markings are millimeters, then we should record the length of data only to the nearest millimeter. The precision of such a measurement would be $\pm$ 0.5mm. We do not review the details of working with significant digits because most readers will have encountered them in previous courses and they are covered in considerable depth in science, engineering, and statistics textbooks.

Issues such as significant digits, precision, bias, and accuracy are sometimes overlooked, but they are important for data mining as well as statistics and science. Many times, data sets do not come with information about the precision of the data, and furthermore, the programs used for analysis return results without any such information. Nonetheless, without some understanding of the

accuracy of the data and the results, an analyst runs the risk of committing serious data analysis blunders.

Outliers

Outliers are either (1) data objects that, in some sense, have characteristics that are different from most of the other data objects in the data set, or (2) values of an attribute that are unusual with respect to the typical values for that attribute. Alternatively, they can be referred to as **anomalous** objects or values. There is considerable leeway in the definition of an outlier, and many different definitions have been proposed by the statistics and data mining communities. Furthermore, it is important to distinguish between the notions of noise and outliers. Unlike noise, outliers can be legitimate data objects or values that we are interested in detecting. For instance, in fraud and network intrusion detection, the goal is to find unusual objects or events from among a large number of normal ones. Chapter 9 discusses anomaly detection in more detail.

Missing Values

It is not unusual for an object to be missing one or more attribute values. In some cases, the information was not collected; e.g., some people decline to give their age or weight. In other cases, some attributes are not applicable to all objects; e.g., often, forms have conditional parts that are filled out only when a person answers a previous question in a certain way, but for simplicity, all fields are stored. Regardless, missing values should be taken into account during the data analysis.

There are several strategies (and variations on these strategies) for dealing with missing data, each of which is appropriate in certain circumstances. These strategies are listed next, along with an indication of their advantages and disadvantages.

Eliminate Data Objects or Attributes A simple and effective strategy is to eliminate objects with missing values. However, even a partially specified data object contains some information, and if many objects have missing values, then a reliable analysis can be difficult or impossible. Nonetheless, if a data set has only a few objects that have missing values, then it may be expedient to omit them. A related strategy is to eliminate attributes that have missing values. This should be done with caution, however, because the eliminated attributes may be the ones that are critical to the analysis.

Estimate Missing Values Sometimes missing data can be reliably estimated. For example, consider a time series that changes in a reasonably smooth fashion, but has a few, widely scattered missing values. In such cases, the missing values can be estimated (interpolated) by using the remaining values. As another example, consider a data set that has many similar data points. In this situation, the attribute values of the points closest to the point with the missing value are often used to estimate the missing value. If the attribute is continuous, then the average attribute value of the nearest neighbors is used; if the attribute is categorical, then the most commonly occurring attribute value can be taken. For a concrete illustration, consider precipitation measurements that are recorded by ground stations. For areas not containing a ground station, the precipitation can be estimated using values observed at nearby ground stations.

Ignore the Missing Value during Analysis Many data mining approaches can be modified to ignore missing values. For example, suppose that objects are being clustered and the similarity between pairs of data objects needs to be calculated. If one or both objects of a pair have missing values for some attributes, then the similarity can be calculated by using only the attributes that do not have missing values. It is true that the similarity will only be approximate, but unless the total number of attributes is small or the number of missing values is high, this degree of inaccuracy may not matter much. Likewise, many classification schemes can be modified to work with missing values.

Inconsistent Values

Data can contain inconsistent values. Consider an address field, where both a zip code and city are listed, but the specified zip code area is not contained in that city. It is possible that the individual entering this information transposed two digits, or perhaps a digit was misread when the information was scanned from a handwritten form. Regardless of the cause of the inconsistent values, it is important to detect and, if possible, correct such problems.

Some types of inconsistences are easy to detect. For instance, a person's height should not be negative. In other cases, it can be necessary to consult an external source of information. For example, when an insurance company processes claims for reimbursement, it checks the names and addresses on the reimbursement forms against a database of its customers.

Once an inconsistency has been detected, it is sometimes possible to correct the data. A product code may have "check" digits, or it may be possible to

double-check a product code against a list of known product codes, and then correct the code if it is incorrect, but close to a known code. The correction of an inconsistency requires additional or redundant information.

Example 2.6 (Inconsistent Sea Surface Temperature)**.** This example illustrates an inconsistency in actual time series data that measures the sea surface temperature (SST) at various points on the ocean. SST data was originally collected using ocean-based measurements from ships or buoys, but more recently, satellites have been used to gather the data. To create a long-term data set, both sources of data must be used. However, because the data comes from different sources, the two parts of the data are subtly different. This discrepancy is visually displayed in Figure 2.7, which shows the correlation of SST values between pairs of years. If a pair of years has a positive correlation, then the location corresponding to the pair of years is colored white; otherwise it is colored black. (Seasonal variations were removed from the data since, otherwise, all the years would be highly correlated.) There is a distinct change in behavior where the data has been put together in 1983. Years within each of the two groups, 1958–1982 and 1983–1999, tend to have a positive correlation with one another, but a negative correlation with years in the other group. This does not mean that this data should not be used, only that the analyst should consider the potential impact of such discrepancies on the data mining analysis. ■

Duplicate Data

A data set can include data objects that are duplicates, or almost duplicates, of one another. Many people receive duplicate mailings because they appear in a database multiple times under slightly different names. To detect and eliminate such duplicates, two main issues must be addressed. First, if there are two objects that actually represent a single object, then one or more values of corresponding attributes are usually different, and these inconsistent values must be resolved. Second, care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates, such as two distinct people with identical names. The term **deduplication** is often used to refer to the process of dealing with these issues.

In some cases, two or more objects are identical with respect to the attributes measured by the database, but they still represent different objects. Here, the duplicates are legitimate, but can still cause problems for some algorithms if the possibility of identical objects is not specifically accounted for in their design. An example of this is given in Exercise 17 on page 128.

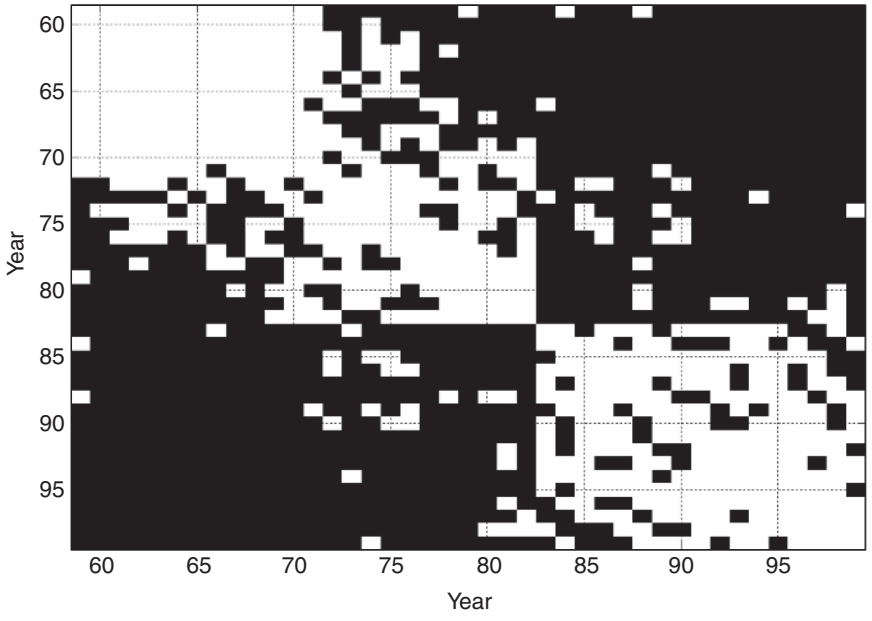


Figure 2.7. Correlation of SST data between pairs of years. White areas indicate positive correlation. Black areas indicate negative correlation.

2.2.2 Issues Related to Applications

Data quality issues can also be considered from an application viewpoint as expressed by the statement "data is of high quality if it is suitable for its intended use." This approach to data quality has proven quite useful, particularly in business and industry. A similar viewpoint is also present in statistics and the experimental sciences, with their emphasis on the careful design of experiments to collect the data relevant to a specific hypothesis. As with quality issues at the measurement and data collection level, many issues are specific to particular applications and fields. Again, we consider only a few of the general issues.

Timeliness Some data starts to age as soon as it has been collected. In particular, if the data provides a snapshot of some ongoing phenomenon or process, such as the purchasing behavior of customers or web browsing patterns, then this snapshot represents reality for only a limited time. If the data is out of date, then so are the models and patterns that are based on it.

Relevance The available data must contain the information necessary for the application. Consider the task of building a model that predicts the accident rate for drivers. If information about the age and gender of the driver is omitted, then it is likely that the model will have limited accuracy unless this information is indirectly available through other attributes.

Making sure that the objects in a data set are relevant is also challenging. A common problem is **sampling bias**, which occurs when a sample does not contain different types of objects in proportion to their actual occurrence in the population. For example, survey data describes only those who respond to the survey. (Other aspects of sampling are discussed further in Section 2.3.2.) Because the results of a data analysis can reflect only the data that is present, sampling bias will typically lead to erroneous results when applied to the broader population.

Knowledge about the Data Ideally, data sets are accompanied by documentation that describes different aspects of the data; the quality of this documentation can either aid or hinder the subsequent analysis. For example, if the documentation identifies several attributes as being strongly related, these attributes are likely to provide highly redundant information, and we usually decide to keep just one. (Consider sales tax and purchase price.) If the documentation is poor, however, and fails to tell us, for example, that the missing values for a particular field are indicated with a -9999, then our analysis of the data may be faulty. Other important characteristics are the precision of the data, the type of features (nominal, ordinal, interval, ratio), the scale of measurement (e.g., meters or feet for length), and the origin of the data.

2.3 Data Preprocessing

In this section, we consider which preprocessing steps should be applied to make the data more suitable for data mining. Data preprocessing is a broad area and consists of a number of different strategies and techniques that are interrelated in complex ways. We will present some of the most important ideas and approaches, and try to point out the interrelationships among them. Specifically, we will discuss the following topics:

- Aggregation
- Sampling
- Dimensionality reduction

- Feature subset selection
- Feature creation
- Discretization and binarization
- Variable transformation

Roughly speaking, these topics fall into two categories: selecting data objects and attributes for the analysis or for creating/changing the attributes. In both cases, the goal is to improve the data mining analysis with respect to time, cost, and quality. Details are provided in the following sections.

A quick note about terminology: In the following, we sometimes use synonyms for attribute, such as feature or variable, in order to follow common usage.

2.3.1 Aggregation

Sometimes "less is more," and this is the case with **aggregation**, the combining of two or more objects into a single object. Consider a data set consisting of transactions (data objects) recording the daily sales of products in various store locations (Minneapolis, Chicago, Paris, ...) for different days over the course of a year. See Table 2.4. One way to aggregate transactions for this data set is to replace all the transactions of a single store with a single storewide transaction. This reduces the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction, and the number of data objects per day is reduced to the number of stores.

An obvious issue is how an aggregate transaction is created; i.e., how the values of each attribute are combined across all the records corresponding to a particular location to create the aggregate transaction that represents the sales of a single store or date. Quantitative attributes, such as *price*, are typically aggregated by taking a sum or an average. A qualitative attribute, such as *item*, can either be omitted or summarized in terms of a higher level category, e.g., televisions versus electronics.

The data in Table 2.4 can also be viewed as a multidimensional array, where each attribute is a dimension. From this viewpoint, aggregation is the process of eliminating attributes, such as the type of item, or reducing the number of values for a particular attribute; e.g., reducing the possible values for *date* from 365 days to 12 months. This type of aggregation is commonly used in Online Analytical Processing (OLAP). References to OLAP are given in the Bibliographic Notes.

There are several motivations for aggregation. First, the smaller data sets resulting from data reduction require less memory and processing time,

Table 2.4. Data set containing information about customer purchases.

Transaction ID	Item	Store Location	Date	Price	...
⋮	⋮	⋮	⋮	⋮	
101123	Watch	Chicago	09/06/04	$25.99	...
101123	Battery	Chicago	09/06/04	$5.99	...
101124	Shoes	Minneapolis	09/06/04	$75.00	...
⋮	⋮	⋮	⋮	⋮	

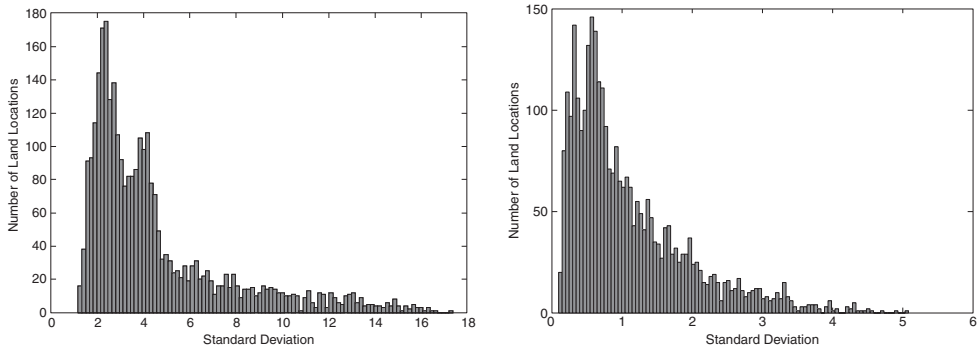
and hence, aggregation often enables the use of more expensive data mining algorithms. Second, aggregation can act as a change of scope or scale by providing a high-level view of the data instead of a low-level view. In the previous example, aggregating over store locations and months gives us a monthly, per store view of the data instead of a daily, per item view. Finally, the behavior of groups of objects or attributes is often more stable than that of individual objects or attributes. This statement reflects the statistical fact that aggregate quantities, such as averages or totals, have less variability than the individual values being aggregated. For totals, the actual amount of variation is larger than that of individual objects (on average), but the percentage of the variation is smaller, while for means, the actual amount of variation is less than that of individual objects (on average). A disadvantage of aggregation is the potential loss of interesting details. In the store example, aggregating over months loses information about which day of the week has the highest sales.

Example 2.7 (Australian Precipitation)**.** This example is based on precipitation in Australia from the period 1982–1993. Figure 2.8(a) shows a histogram for the standard deviation of average monthly precipitation for 3,030 0.5° by 0.5° grid cells in Australia, while Figure 2.8(b) shows a histogram for the standard deviation of the average yearly precipitation for the same locations. The average yearly precipitation has less variability than the average monthly precipitation. All precipitation measurements (and their standard deviations) are in centimeters.

■

2.3.2 Sampling

Sampling is a commonly used approach for selecting a subset of the data objects to be analyzed. In statistics, it has long been used for both the preliminary investigation of the data and the final data analysis. Sampling can also be very useful in data mining. However, the motivations for sampling



(a) Histogram of standard deviation of average monthly precipitation

(b) Histogram of standard deviation of average yearly precipitation

Figure 2.8. Histograms of standard deviation for monthly and yearly precipitation in Australia for the period 1982–1993.

in statistics and data mining are often different. Statisticians use sampling because obtaining the entire set of data of interest is too expensive or time consuming, while data miners usually sample because it is too computationally expensive in terms of the memory or time required to process all the data. In some cases, using a sampling algorithm can reduce the data size to the point where a better, but more computationally expensive algorithm can be used.

The key principle for effective sampling is the following: Using a sample will work almost as well as using the entire data set if the sample is representative. In turn, **a sample is representative** if it has approximately the same property (of interest) as the original set of data. If the mean (average) of the data objects is the property of interest, then a sample is representative if it has a mean that is close to that of the original data. Because sampling is a statistical process, the representativeness of any particular sample will vary, and the best that we can do is choose a sampling scheme that guarantees a high probability of getting a representative sample. As discussed next, this involves choosing the appropriate sample size and sampling technique.

Sampling Approaches

There are many sampling techniques, but only a few of the most basic ones and their variations will be covered here. The simplest type of sampling is **simple random sampling**. For this type of sampling, there is an equal

probability of selecting any particular object. There are two variations on random sampling (and other sampling techniques as well): (1) **sampling without replacement**—as each object is selected, it is removed from the set of all objects that together constitute the **population**, and (2) **sampling with replacement**—objects are not removed from the population as they are selected for the sample. In sampling with replacement, the same object can be picked more than once. The samples produced by the two methods are not much different when samples are relatively small compared to the data set size, but sampling with replacement is simpler to analyze because the probability of selecting any object remains constant during the sampling process.

When the population consists of different types of objects, with widely different numbers of objects, simple random sampling can fail to adequately represent those types of objects that are less frequent. This can cause problems when the analysis requires proper representation of all object types. For example, when building classification models for rare classes, it is critical that the rare classes be adequately represented in the sample. Hence, a sampling scheme that can accommodate differing frequencies for the object types of interest is needed. **Stratified sampling**, which starts with prespecified groups of objects, is such an approach. In the simplest version, equal numbers of objects are drawn from each group even though the groups are of different sizes. In another variation, the number of objects drawn from each group is proportional to the size of that group.

Example 2.8 (Sampling and Loss of Information)**.** Once a sampling technique has been selected, it is still necessary to choose the sample size. Larger sample sizes increase the probability that a sample will be representative, but they also eliminate much of the advantage of sampling. Conversely, with smaller sample sizes, patterns can be missed or erroneous patterns can be detected. Figure 2.9(a) shows a data set that contains 8000 two-dimensional points, while Figures 2.9(b) and 2.9(c) show samples from this data set of size 2000 and 500, respectively. Although most of the structure of this data set is present in the sample of 2000 points, much of the structure is missing in the sample of 500 points. ■

Example 2.9 (Determining the Proper Sample Size)**.** To illustrate that determining the proper sample size requires a methodical approach, consider the following task.

> Given a set of data consisting of a small number of almost equal-sized groups, find at least one representative point for each of the groups. Assume that the objects in each group are highly similar

(a) 8000 points (b) 2000 points (c) 500 points

Figure 2.9. Example of the loss of structure with sampling.

to each other, but not very similar to objects in different groups. Figure 2.10(a) shows an idealized set of clusters (groups) from which these points might be drawn.

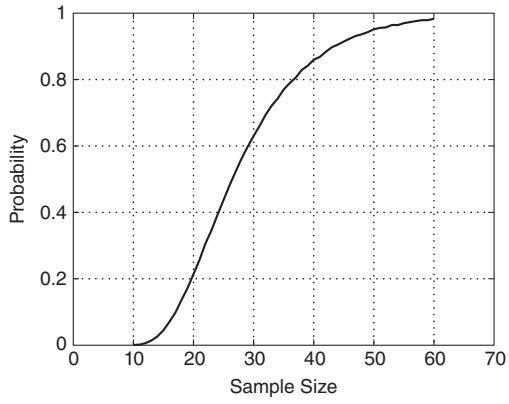
This problem can be efficiently solved using sampling. One approach is to take a small sample of data points, compute the pairwise similarities between points, and then form groups of points that are highly similar. The desired set of representative points is then obtained by taking one point from each of these groups. To follow this approach, however, we need to determine a sample size that would guarantee, with a high probability, the desired outcome; that is, that at least one point will be obtained from each cluster. Figure 2.10(b) shows the probability of getting one object from each of the 10 groups as the sample size runs from 10 to 60. Interestingly, with a sample size of 20, there is little chance (20%) of getting a sample that includes all 10 clusters. Even with a sample size of 30, there is still a moderate chance (almost 40%) of getting a sample that doesn't contain objects from all 10 clusters. This issue is further explored in the context of clustering by Exercise 4 on page 126.

■

Progressive Sampling

The proper sample size can be difficult to determine, so **adaptive** or **progressive sampling** schemes are sometimes used. These approaches start with a small sample, and then increase the sample size until a sample of sufficient size has been obtained. While this technique eliminates the need to determine

(a) Ten groups of points.



(b) Probability a sample contains points from each of 10 groups.

Figure 2.10. Finding representative points from 10 groups.

the correct sample size initially, it requires that there be a way to evaluate the sample to judge if it is large enough.

Suppose, for instance, that progressive sampling is used to learn a predictive model. Although the accuracy of predictive models increases as the sample size increases, at some point the increase in accuracy levels off. We want to stop increasing the sample size at this leveling-off point. By keeping track of the change in accuracy of the model as we take progressively larger samples, and by taking other samples close to the size of the current one, we can get an estimate of how close we are to this leveling-off point, and thus, stop sampling.

2.3.3 Dimensionality Reduction

Data sets can have a large number of features. Consider a set of documents, where each document is represented by a vector whose components are the frequencies with which each word occurs in the document. In such cases, there are typically thousands or tens of thousands of attributes (components), one for each word in the vocabulary. As another example, consider a set of time series consisting of the daily closing price of various stocks over a period of 30 years. In this case, the attributes, which are the prices on specific days, again number in the thousands.

There are a variety of benefits to dimensionality reduction. A key benefit is that many data mining algorithms work better if the dimensionality—the number of attributes in the data—is lower. This is partly because dimensionality reduction can eliminate irrelevant features and reduce noise and partly because of the curse of dimensionality, which is explained below. Another benefit is that a reduction of dimensionality can lead to a more understandable model because the model usually involves fewer attributes. Also, dimensionality reduction may allow the data to be more easily visualized. Even if dimensionality reduction doesn't reduce the data to two or three dimensions, data is often visualized by looking at pairs or triplets of attributes, and the number of such combinations is greatly reduced. Finally, the amount of time and memory required by the data mining algorithm is reduced with a reduction in dimensionality.

The term dimensionality reduction is often reserved for those techniques that reduce the dimensionality of a data set by creating new attributes that are a combination of the old attributes. The reduction of dimensionality by selecting attributes that are a subset of the old is known as feature subset selection or feature selection. It will be discussed in Section 2.3.4.

In the remainder of this section, we briefly introduce two important topics: the curse of dimensionality and dimensionality reduction techniques based on linear algebra approaches such as principal components analysis (PCA). More details on dimensionality reduction can be found in Appendix B.

The Curse of Dimensionality

The curse of dimensionality refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases. Specifically, as dimensionality increases, the data becomes increasingly sparse in the space that it occupies. Thus, the data objects we observe are quite possibly not a representative sample of all possible objects. For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects. For clustering, the differences in density and in the distances between points, which are critical for clustering, become less meaningful. (This is discussed further in Sections 8.1.2, 8.4.6, and 8.4.8.) As a result, many clustering and classification algorithms (and other data analysis algorithms) have trouble with high-dimensional data leading to reduced classification accuracy and poor quality clusters.

Linear Algebra Techniques for Dimensionality Reduction

Some of the most common approaches for dimensionality reduction, particularly for continuous data, use techniques from linear algebra to project the data from a high-dimensional space into a lower-dimensional space. **Principal Components Analysis (PCA)** is a linear algebra technique for continuous attributes that finds new attributes (principal components) that (1) are linear combinations of the original attributes, (2) are **orthogonal** (perpendicular) to each other, and (3) capture the maximum amount of variation in the data. For example, the first two principal components capture as much of the variation in the data as is possible with two orthogonal attributes that are linear combinations of the original attributes. **Singular Value Decomposition (SVD)** is a linear algebra technique that is related to PCA and is also commonly used for dimensionality reduction. For additional details, see Appendices A and B.

2.3.4 Feature Subset Selection

Another way to reduce the dimensionality is to use only a subset of the features. While it might seem that such an approach would lose information, this is not the case if redundant and irrelevant features are present. **Redundant features** duplicate much or all of the information contained in one or more other attributes. For example, the purchase price of a product and the amount of sales tax paid contain much of the same information. **Irrelevant features** contain almost no useful information for the data mining task at hand. For instance, students' ID numbers are irrelevant to the task of predicting students' grade point averages. Redundant and irrelevant features can reduce classification accuracy and the quality of the clusters that are found.

While some irrelevant and redundant attributes can be eliminated immediately by using common sense or domain knowledge, selecting the best subset of features frequently requires a systematic approach. The ideal approach to feature selection is to try all possible subsets of features as input to the data mining algorithm of interest, and then take the subset that produces the best results. This method has the advantage of reflecting the objective and bias of the data mining algorithm that will eventually be used. Unfortunately, since the number of subsets involving n attributes is 2^n, such an approach is impractical in most situations and alternative strategies are needed. There are three standard approaches to feature selection: embedded, filter, and wrapper.

Embedded approaches Feature selection occurs naturally as part of the data mining algorithm. Specifically, during the operation of the data mining algorithm, the algorithm itself decides which attributes to use and which to ignore. Algorithms for building decision tree classifiers, which are discussed in Chapter 3, often operate in this manner.

Filter approaches Features are selected before the data mining algorithm is run, using some approach that is independent of the data mining task. For example, we might select sets of attributes whose pairwise correlation is as low as possible so that the attributes are non-redundant.

Wrapper approaches These methods use the target data mining algorithm as a black box to find the best subset of attributes, in a way similar to that of the ideal algorithm described above, but typically without enumerating all possible subsets.

Because the embedded approaches are algorithm-specific, only the filter and wrapper approaches will be discussed further here.

An Architecture for Feature Subset Selection

It is possible to encompass both the filter and wrapper approaches within a common architecture. The feature selection process is viewed as consisting of four parts: a measure for evaluating a subset, a search strategy that controls the generation of a new subset of features, a stopping criterion, and a validation procedure. Filter methods and wrapper methods differ only in the way in which they evaluate a subset of features. For a wrapper method, subset evaluation uses the target data mining algorithm, while for a filter approach, the evaluation technique is distinct from the target data mining algorithm. The following discussion provides some details of this approach, which is summarized in Figure 2.11.

Conceptually, feature subset selection is a search over all possible subsets of features. Many different types of search strategies can be used, but the search strategy should be computationally inexpensive and should find optimal or near optimal sets of features. It is usually not possible to satisfy both requirements, and thus, trade-offs are necessary.

An integral part of the search is an evaluation step to judge how the current subset of features compares to others that have been considered. This requires an evaluation measure that attempts to determine the goodness of a subset of attributes with respect to a particular data mining task, such as

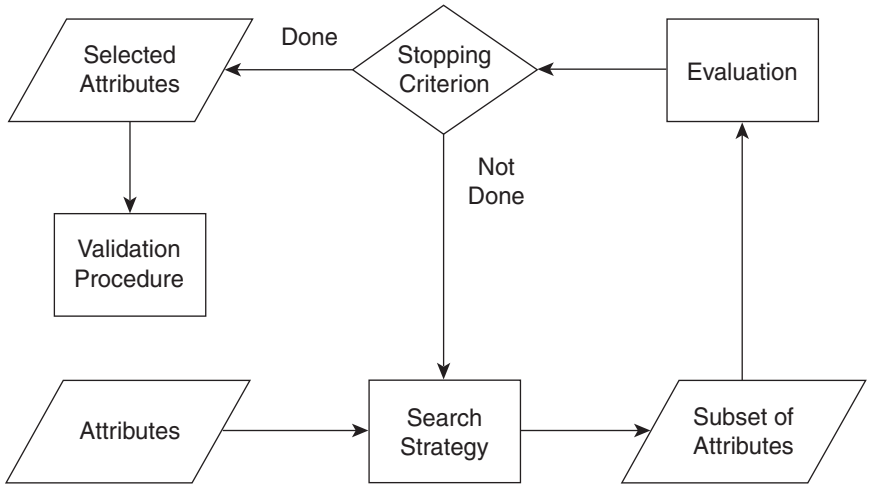


Figure 2.11. Flowchart of a feature subset selection process.

classification or clustering. For the filter approach, such measures attempt to predict how well the actual data mining algorithm will perform on a given set of attributes. For the wrapper approach, where evaluation consists of actually running the target data mining algorithm, the subset evaluation function is simply the criterion normally used to measure the result of the data mining.

Because the number of subsets can be enormous and it is impractical to examine them all, some sort of stopping criterion is necessary. This strategy is usually based on one or more conditions involving the following: the number of iterations, whether the value of the subset evaluation measure is optimal or exceeds a certain threshold, whether a subset of a certain size has been obtained, and whether any improvement can be achieved by the options available to the search strategy.

Finally, once a subset of features has been selected, the results of the target data mining algorithm on the selected subset should be validated. A straightforward validation approach is to run the algorithm with the full set of features and compare the full results to results obtained using the subset of features. Hopefully, the subset of features will produce results that are better than or almost as good as those produced when using all features. Another validation approach is to use a number of different feature selection algorithms to obtain subsets of features and then compare the results of running the data mining algorithm on each subset.

Feature Weighting

Feature weighting is an alternative to keeping or eliminating features. More important features are assigned a higher weight, while less important features are given a lower weight. These weights are sometimes assigned based on domain knowledge about the relative importance of features. Alternatively, they can sometimes be determined automatically. For example, some classification schemes, such as support vector machines (Chapter 6), produce classification models in which each feature is given a weight. Features with larger weights play a more important role in the model. The normalization of objects that takes place when computing the cosine similarity (Section 2.4.5) can also be regarded as a type of feature weighting.

2.3.5 Feature Creation

It is frequently possible to create, from the original attributes, a new set of attributes that captures the important information in a data set much more effectively. Furthermore, the number of new attributes can be smaller than the number of original attributes, allowing us to reap all the previously described benefits of dimensionality reduction. Two related methodologies for creating new attributes are described next: feature extraction and mapping the data to a new space.

Feature Extraction

The creation of a new set of features from the original raw data is known as **feature extraction**. Consider a set of photographs, where each photograph is to be classified according to whether it contains a human face. The raw data is a set of pixels, and as such, is not suitable for many types of classification algorithms. However, if the data is processed to provide higher-level features, such as the presence or absence of certain types of edges and areas that are highly correlated with the presence of human faces, then a much broader set of classification techniques can be applied to this problem.

Unfortunately, in the sense in which it is most commonly used, feature extraction is highly domain-specific. For a particular field, such as image processing, various features and the techniques to extract them have been developed over a period of time, and often these techniques have limited applicability to other fields. Consequently, whenever data mining is applied to a relatively new area, a key task is the development of new features and feature extraction methods.

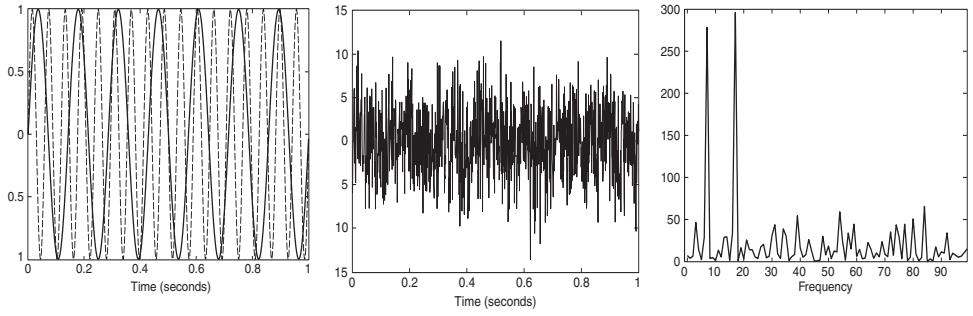
Although feature extraction is often complicated, Example 2.10 illustrates that it can be relatively straightforward.

Example 2.10 (Density)**.** Consider a data set consisting of information about historical artifacts, which, along with other information, contains the volume and mass of each artifact. For simplicity, assume that these artifacts are made of a small number of materials (wood, clay, bronze, gold) and that we want to classify the artifacts with respect to the material of which they are made. In this case, a density feature constructed from the mass and volume features, i.e., $density = mass/volume$, would most directly yield an accurate classification. Although there have been some attempts to automatically perform such simple feature extraction by exploring basic mathematical combinations of existing attributes, the most common approach is to construct features using domain expertise. ■

Mapping the Data to a New Space

A totally different view of the data can reveal important and interesting features. Consider, for example, time series data, which often contains periodic patterns. If there is only a single periodic pattern and not much noise, then the pattern is easily detected. If, on the other hand, there are a number of periodic patterns and a significant amount of noise, then these patterns are hard to detect. Such patterns can, nonetheless, often be detected by applying a **Fourier transform** to the time series in order to change to a representation in which frequency information is explicit. In Example 2.11, it will not be necessary to know the details of the Fourier transform. It is enough to know that, for each time series, the Fourier transform produces a new data object whose attributes are related to frequencies.

Example 2.11 (Fourier Analysis)**.** The time series presented in Figure 2.12(b) is the sum of three other time series, two of which are shown in Figure 2.12(a) and have frequencies of 7 and 17 cycles per second, respectively. The third time series is random noise. Figure 2.12(c) shows the power spectrum that can be computed after applying a Fourier transform to the original time series. (Informally, the power spectrum is proportional to the square of each frequency attribute.) In spite of the noise, there are two peaks that correspond to the periods of the two original, non-noisy time series. Again, the main point is that better features can reveal important aspects of the data. ■



(a) Two time series. (b) Noisy time series. (c) Power spectrum.

Figure 2.12. Application of the Fourier transform to identify the underlying frequencies in time series data.

Many other sorts of transformations are also possible. Besides the Fourier transform, the **wavelet transform** has also proven very useful for time series and other types of data.

2.3.6 Discretization and Binarization

Some data mining algorithms, especially certain classification algorithms, require that the data be in the form of categorical attributes. Algorithms that find association patterns require that the data be in the form of binary attributes. Thus, it is often necessary to transform a continuous attribute into a categorical attribute (**discretization**), and both continuous and discrete attributes may need to be transformed into one or more binary attributes (**binarization**). Additionally, if a categorical attribute has a large number of values (categories), or some values occur infrequently, then it can be beneficial for certain data mining tasks to reduce the number of categories by combining some of the values.

As with feature selection, the best discretization or binarization approach is the one that "produces the best result for the data mining algorithm that will be used to analyze the data." It is typically not practical to apply such a criterion directly. Consequently, discretization or binarization is performed in a way that satisfies a criterion that is thought to have a relationship to good performance for the data mining task being considered. In general, the best discretization depends on the algorithm being used, as well as the other

Table 2.5. Conversion of a categorical attribute to three binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3
awful	0	0	0	0
poor	1	0	0	1
OK	2	0	1	0
good	3	0	1	1
great	4	1	0	0

Table 2.6. Conversion of a categorical attribute to five asymmetric binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
awful	0	1	0	0	0	0
poor	1	0	1	0	0	0
OK	2	0	0	1	0	0
good	3	0	0	0	1	0
great	4	0	0	0	0	1

attributes being considered. Typically, however, the discretization of each attribute is considered in isolation.

Binarization

A simple technique to binarize a categorical attribute is the following: If there are m categorical values, then uniquely assign each original value to an integer in the interval $[0, m-1]$. If the attribute is ordinal, then order must be maintained by the assignment. (Note that even if the attribute is originally represented using integers, this process is necessary if the integers are not in the interval $[0, m-1]$.) Next, convert each of these m integers to a binary number. Since $n = \lceil \log_2(m) \rceil$ binary digits are required to represent these integers, represent these binary numbers using n binary attributes. To illustrate, a categorical variable with 5 values {*awful*, *poor*, *OK*, *good*, *great*} would require three binary variables x_1, x_2, and x_3. The conversion is shown in Table 2.5.

Such a transformation can cause complications, such as creating unintended relationships among the transformed attributes. For example, in Table 2.5, attributes x_2 and x_3 are correlated because information about the *good* value is encoded using both attributes. Furthermore, association analysis requires asymmetric binary attributes, where only the presence of the attribute (value = 1) is important. For association problems, it is therefore necessary to introduce one asymmetric binary attribute for each categorical value, as

shown in Table 2.6. If the number of resulting attributes is too large, then the techniques described in the following sections can be used to reduce the number of categorical values before binarization.

Likewise, for association problems, it can be necessary to replace a single binary attribute with two asymmetric binary attributes. Consider a binary attribute that records a person's gender, male or female. For traditional association rule algorithms, this information needs to be transformed into two asymmetric binary attributes, one that is a 1 only when the person is male and one that is a 1 only when the person is female. (For asymmetric binary attributes, the information representation is somewhat inefficient in that two bits of storage are required to represent each bit of information.)

Discretization of Continuous Attributes

Discretization is typically applied to attributes that are used in classification or association analysis. Transformation of a continuous attribute to a categorical attribute involves two subtasks: deciding how many categories, n, to have and determining how to map the values of the continuous attribute to these categories. In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying $n-1$ **split points**. In the second, rather trivial step, all the values in one interval are mapped to the same categorical value. Therefore, the problem of discretization is one of deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals $\{(x_0, x_1], (x_1, x_2], \ldots, (x_{n-1}, x_n)\}$, where x_0 and x_n can be $+\infty$ or $-\infty$, respectively, or equivalently, as a series of inequalities $x_0 < x \leq x_1, \ldots, x_{n-1} < x < x_n$.

Unsupervised Discretization A basic distinction between discretization methods for classification is whether class information is used (supervised) or not (unsupervised). If class information is not used, then relatively simple approaches are common. For instance, the **equal width** approach divides the range of the attribute into a user-specified number of intervals each having the same width. Such an approach can be badly affected by outliers, and for that reason, an **equal frequency (equal depth)** approach, which tries to put the same number of objects into each interval, is often preferred. As another example of unsupervised discretization, a clustering method, such as K-means (see Chapter 5), can also be used. Finally, visually inspecting the data can sometimes be an effective approach.

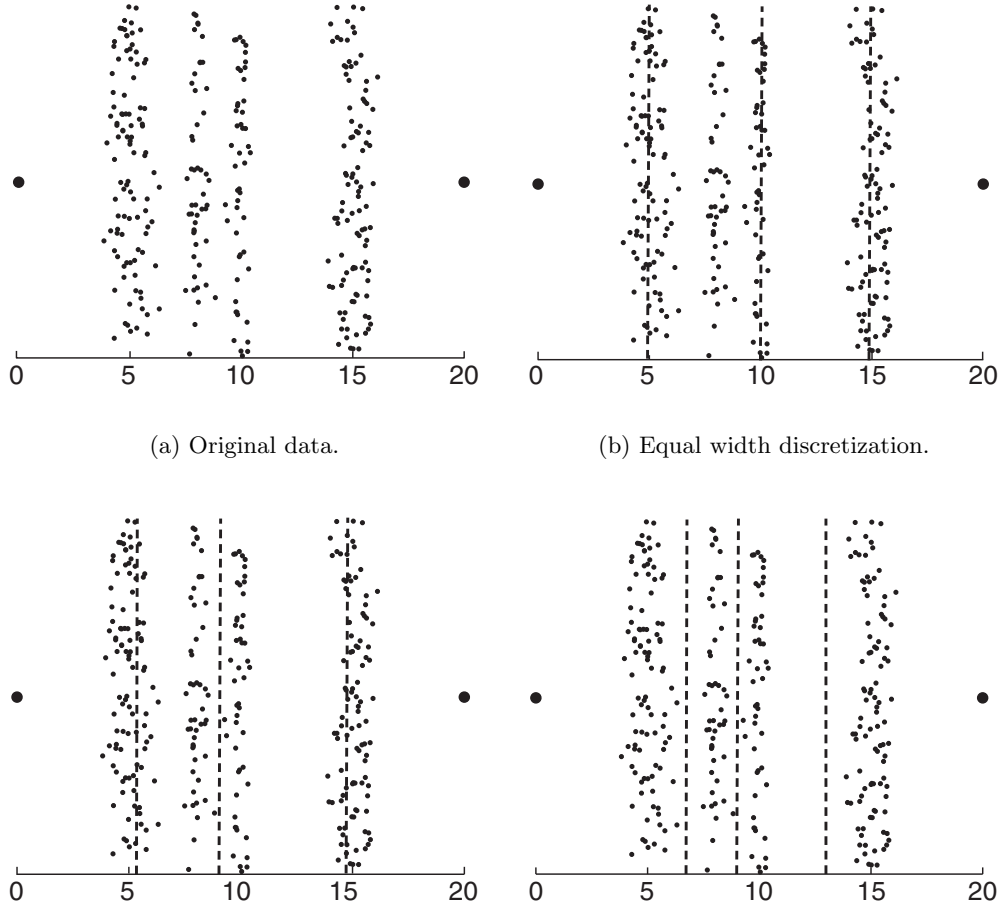
Example 2.12 (Discretization Techniques)**.** This example demonstrates how these approaches work on an actual data set. Figure 2.13(a) shows data points belonging to four different groups, along with two outliers—the large dots on either end. The techniques of the previous paragraph were applied to discretize the x values of these data points into four categorical values. (Points in the data set have a random y component to make it easy to see how many points are in each group.) Visually inspecting the data works quite well, but is not automatic, and thus, we focus on the other three approaches. The split points produced by the techniques equal width, equal frequency, and K-means are shown in Figures 2.13(b), 2.13(c), and 2.13(d), respectively. The split points are represented as dashed lines.

In this particular example, if we measure the performance of a discretization technique by the extent to which different objects that clump together have the same categorical value, then K-means performs best, followed by equal frequency, and finally, equal width. More generally, the best discretization will depend on the application and often involves domain-specific discretization. For example, the discretization of people into low income, middle income, and high income is based on economic factors.

■

Supervised Discretization If classification is our application and class labels are known for some data objects, then discretization approaches that use class labels often produce better classification. This should not be surprising, since an interval constructed with no knowledge of class labels often contains a mixture of class labels. A conceptually simple approach is to place the splits in a way that maximizes the purity of the intervals, i.e., the extent to which an interval contains a single class label. In practice, however, such an approach requires potentially arbitrary decisions about the purity of an interval and the minimum size of an interval.

To overcome such concerns, some statistically based approaches start with each attribute value in a separate interval and create larger intervals by merging adjacent intervals that are similar according to a statistical test. An alternative to this bottom-up approach is a top-down approach that starts by bisecting the initial values so that the resulting two intervals give minimum entropy. This technique only needs to consider each value as a possible split point, because it is assumed that intervals contain ordered sets of values. The splitting process is then repeated with another interval, typically choosing the interval with the worst (highest) entropy, until a user-specified number of intervals is reached, or a stopping criterion is satisfied.



(a) Original data.

(b) Equal width discretization.

(c) Equal frequency discretization.

(d) K-means discretization.

Figure 2.13. Different discretization techniques.

Entropy-based approaches are one of the most promising approaches to discretization, whether bottom-up or top-down. First, it is necessary to define **entropy**. Let k be the number of different class labels, m_i be the number of values in the i^{th} interval of a partition, and m_{ij} be the number of values of class j in interval i. Then the entropy e_i of the i^{th} interval is given by the equation

$$e_i = -\sum_{j=1}^{k} p_{ij} \log_2 p_{ij},$$

where $p_{ij} = m_{ij}/m_i$ is the probability (fraction of values) of class j in the i^{th} interval. The total entropy, e, of the partition is the weighted average of the individual interval entropies, i.e.,

$$e = \sum_{i=1}^{n} w_i e_i,$$

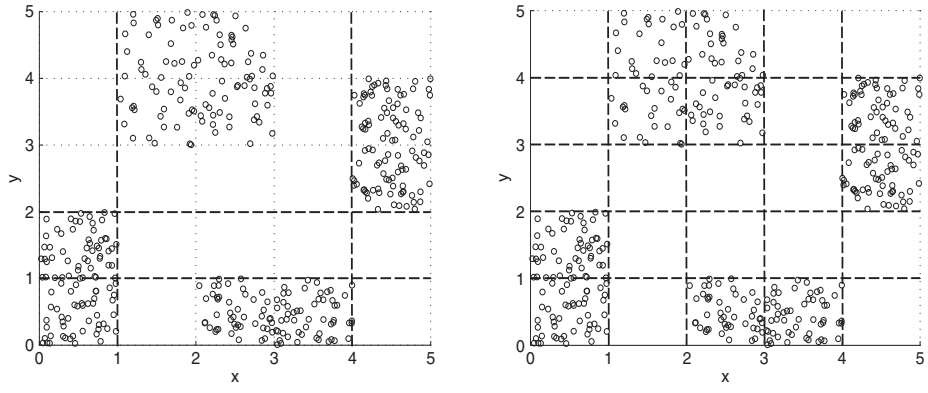
where m is the number of values, $w_i = m_i/m$ is the fraction of values in the i^{th} interval, and n is the number of intervals. Intuitively, the entropy of an interval is a measure of the purity of an interval. If an interval contains only values of one class (is perfectly pure), then the entropy is 0 and it contributes nothing to the overall entropy. If the classes of values in an interval occur equally often (the interval is as impure as possible), then the entropy is a maximum.

Example 2.13 (Discretization of Two Attributes)**.** The top-down method based on entropy was used to independently discretize both the x and y attributes of the two-dimensional data shown in Figure 2.14. In the first discretization, shown in Figure 2.14(a), the x and y attributes were both split into three intervals. (The dashed lines indicate the split points.) In the second discretization, shown in Figure 2.14(b), the x and y attributes were both split into five intervals. ■

This simple example illustrates two aspects of discretization. First, in two dimensions, the classes of points are well separated, but in one dimension, this is not so. In general, discretizing each attribute separately often guarantees suboptimal results. Second, five intervals work better than three, but six intervals do not improve the discretization much, at least in terms of entropy. (Entropy values and results for six intervals are not shown.) Consequently, it is desirable to have a stopping criterion that automatically finds the right number of partitions.

Categorical Attributes with Too Many Values

Categorical attributes can sometimes have too many values. If the categorical attribute is an ordinal attribute, then techniques similar to those for continuous attributes can be used to reduce the number of categories. If the categorical attribute is nominal, however, then other approaches are needed. Consider a university that has a large number of departments. Consequently,



(a) Three intervals (b) Five intervals

Figure 2.14. Discretizing x and y attributes for four groups (classes) of points.

a *department name* attribute might have dozens of different values. In this situation, we could use our knowledge of the relationships among different departments to combine departments into larger groups, such as *engineering*, *social sciences*, or *biological sciences*. If domain knowledge does not serve as a useful guide or such an approach results in poor classification performance, then it is necessary to use a more empirical approach, such as grouping values together only if such a grouping results in improved classification accuracy or achieves some other data mining objective.

2.3.7 Variable Transformation

A **variable transformation** refers to a transformation that is applied to all the values of a variable. (We use the term variable instead of attribute to adhere to common usage, although we will also refer to attribute transformation on occasion.) In other words, for each object, the transformation is applied to the value of the variable for that object. For example, if only the magnitude of a variable is important, then the values of the variable can be transformed by taking the absolute value. In the following section, we discuss two important types of variable transformations: simple functional transformations and normalization.

Simple Functions

For this type of variable transformation, a simple mathematical function is applied to each value individually. If x is a variable, then examples of such transformations include x^k, $\log x$, e^x, $\sqrt{x}$, $1/x$, $\sin x$, or $|x|$. In statistics, variable transformations, especially sqrt, log, and $1/x$, are often used to transform data that does not have a Gaussian (normal) distribution into data that does. While this can be important, other reasons often take precedence in data mining. Suppose the variable of interest is the number of data bytes in a session, and the number of bytes ranges from 1 to 1 billion. This is a huge range, and it can be advantageous to compress it by using a $\log_{10}$ transformation. In this case, sessions that transferred 10^8 and 10^9 bytes would be more similar to each other than sessions that transferred 10 and 1000 bytes ($9 - 8 = 1$ versus $3 - 1 = 2$). For some applications, such as network intrusion detection, this may be what is desired, since the first two sessions most likely represent transfers of large files, while the latter two sessions could be two quite distinct types of sessions.

Variable transformations should be applied with caution because they change the nature of the data. While this is what is desired, there can be problems if the nature of the transformation is not fully appreciated. For instance, the transformation $1/x$ reduces the magnitude of values that are 1 or larger, but increases the magnitude of values between 0 and 1. To illustrate, the values $\{1, 2, 3\}$ go to $\{1, \frac{1}{2}, \frac{1}{3}\}$, but the values $\{1, \frac{1}{2}, \frac{1}{3}\}$ go to $\{1, 2, 3\}$. Thus, for all sets of values, the transformation $1/x$ reverses the order. To help clarify the effect of a transformation, it is important to ask questions such as the following: What is the desired property of the transformed attribute? Does the order need to be maintained? Does the transformation apply to all values, especially negative values and 0? What is the effect of the transformation on the values between 0 and 1? Exercise 21 on page 129 explores other aspects of variable transformation.

Normalization or Standardization

The goal of standardization or normalization is to make an entire set of values have a particular property. A traditional example is that of "standardizing a variable" in statistics. If $\overline{x}$ is the mean (average) of the attribute values and s_x is their standard deviation, then the transformation $x' = (x - \overline{x})/s_x$ creates a new variable that has a mean of 0 and a standard deviation of 1. If different variables are to be used together, e.g., for clustering, then such a transformation is often necessary to avoid having a variable with large values

dominate the results of the analysis. To illustrate, consider comparing people based on two variables: age and income. For any two people, the difference in income will likely be much higher in absolute terms (hundreds or thousands of dollars) than the difference in age (less than 150). If the differences in the range of values of age and income are not taken into account, then the comparison between people will be dominated by differences in income. In particular, if the similarity or dissimilarity of two people is calculated using the similarity or dissimilarity measures defined later in this chapter, then in many cases, such as that of Euclidean distance, the income values will dominate the calculation.

The mean and standard deviation are strongly affected by outliers, so the above transformation is often modified. First, the mean is replaced by the **median**, i.e., the middle value. Second, the standard deviation is replaced by the **absolute standard deviation**. Specifically, if x is a variable, then the absolute standard deviation of x is given by $\sigma_A = \sum_{i=1}^{m} |x_i - \mu|$, where x_i is the i^{th} value of the variable, m is the number of objects, and μ is either the mean or median. Other approaches for computing estimates of the location (center) and spread of a set of values in the presence of outliers are described in statistics books. These more robust measures can also be used to define a standardization transformation.

2.4 Measures of Similarity and Dissimilarity

Similarity and dissimilarity are important because they are used by a number of data mining techniques, such as clustering, nearest neighbor classification, and anomaly detection. In many cases, the initial data set is not needed once these similarities or dissimilarities have been computed. Such approaches can be viewed as transforming the data to a similarity (dissimilarity) space and then performing the analysis. Indeed, **kernel methods** are a powerful realization of this idea. These methods are introduced in Section 2.4.7 and are discussed more fully in the context of classification in Section 6.9.4.

We begin with a discussion of the basics: high-level definitions of similarity and dissimilarity, and a discussion of how they are related. For convenience, the term **proximity** is used to refer to either similarity or dissimilarity. Since the proximity between two objects is a function of the proximity between the corresponding attributes of the two objects, we first describe how to measure the proximity between objects having only one attribute.

We then consider proximity measures for objects with multiple attributes. This includes measures such as the Jaccard and cosine similarity measures, which are useful for sparse data, such as documents, as well as correlation

and Euclidean distance, which are useful for non-sparse (dense) data, such as time series or multi-dimensional points. We also consider mutual information, which can be applied to many types of data and is good for detecting nonlinear relationships. In this discussion, we restrict ourselves to objects with relatively homogeneous attribute types, typically binary or continuous.

Next, we consider several important issues concerning proximity measures. This includes how to compute proximity between objects when they have heterogeneous types of attributes, and approaches to account for differences of scale and correlation among variables when computing distance between numerical objects. The section concludes with a brief discussion of how to select the right proximity measure.

Although this section focuses on the computation of proximity between data objects, proximity can also be computed between attributes. For example, for the document-term matrix of Figure 2.2(d), the cosine measure can be used to compute similarity between a pair of documents or a pair of terms (words). Knowing that two variables are strongly related can, for example, be helpful for eliminating redundancy. In particular, the correlation and mutual information measures discussed later are often used for that purpose.

2.4.1 Basics

Definitions

Informally, the **similarity** between two objects is a numerical measure of the degree to which the two objects are alike. Consequently, similarities are *higher* for pairs of objects that are more alike. Similarities are usually non-negative and are often between 0 (no similarity) and 1 (complete similarity).

The **dissimilarity** between two objects is a numerical measure of the degree to which the two objects are different. Dissimilarities are *lower* for more similar pairs of objects. Frequently, the term **distance** is used as a synonym for dissimilarity, although, as we shall see, distance often refers to a special class of dissimilarities. Dissimilarities sometimes fall in the interval $[0, 1]$, but it is also common for them to range from 0 to ∞.

Transformations

Transformations are often applied to convert a similarity to a dissimilarity, or vice versa, or to transform a proximity measure to fall within a particular range, such as [0,1]. For instance, we may have similarities that range from 1 to 10, but the particular algorithm or software package that we want to use may be designed to work only with dissimilarities, or it may work only with

similarities in the interval [0,1]. We discuss these issues here because we will employ such transformations later in our discussion of proximity. In addition, these issues are relatively independent of the details of specific proximity measures.

Frequently, proximity measures, especially similarities, are defined or transformed to have values in the interval [0,1]. Informally, the motivation for this is to use a scale in which a proximity value indicates the fraction of similarity (or dissimilarity) between two objects. Such a transformation is often relatively straightforward. For example, if the similarities between objects range from 1 (not at all similar) to 10 (completely similar), we can make them fall within the range $[0, 1]$ by using the transformation $s' = (s - 1)/9$, where s and s' are the original and new similarity values, respectively. In the more general case, the transformation of similarities to the interval $[0, 1]$ is given by the expression $s' = (s - min\_s)/(max\_s - min\_s)$, where max_s and min_s are the maximum and minimum similarity values, respectively. Likewise, dissimilarity measures with a finite range can be mapped to the interval [0,1] by using the formula $d' = (d - min\_d)/(max\_d - min\_d)$. This is an example of a linear transformation, which preserves the relative distances between points. In other words, if points, x_1 and x_2, are twice as far apart as points, x_3 and x_4, the same will be true after a linear transformation.

However, there can be complications in mapping proximity measures to the interval $[0, 1]$ using a linear transformation. If, for example, the proximity measure originally takes values in the interval [0,∞], then max_d is not defined and a nonlinear transformation is needed. Values will not have the same relationship to one another on the new scale. Consider the transformation $d' = d/(1 + d)$ for a dissimilarity measure that ranges from 0 to ∞. The dissimilarities 0, 0.5, 2, 10, 100, and 1000 will be transformed into the new dissimilarities 0, 0.33, 0.67, 0.90, 0.99, and 0.999, respectively. Larger values on the original dissimilarity scale are compressed into the range of values near 1, but whether this is desirable depends on the application.

Note that mapping proximity measures to the interval $[0, 1]$ can also change the meaning of the proximity measure. For example, correlation, which is discussed later, is a measure of similarity that takes values in the interval $[-1, 1]$. Mapping these values to the interval [0,1] by taking the absolute value loses information about the sign, which can be important in some applications. See Exercise 27 on page 131.

Transforming similarities to dissimilarities and vice versa is also relatively straightforward, although we again face the issues of preserving meaning and changing a linear scale into a nonlinear scale. If the similarity (or dissimilarity) falls in the interval [0,1], then the dissimilarity can be defined as $d = 1 - s$

($s = 1 - d$). Another simple approach is to define similarity as the negative of the dissimilarity (or vice versa). To illustrate, the dissimilarities 0, 1, 10, and 100 can be transformed into the similarities 0, -1, -10, and -100, respectively.

The similarities resulting from the negation transformation are not restricted to the range $[0, 1]$, but if that is desired, then transformations such as $s = \frac{1}{d+1}$, $s = e^{-d}$, or $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$ can be used. For the transformation $s = \frac{1}{d+1}$, the dissimilarities 0, 1, 10, 100 are transformed into 1, 0.5, 0.09, 0.01, respectively. For $s = e^{-d}$, they become 1.00, 0.37, 0.00, 0.00, respectively, while for $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$ they become 1.00, 0.99, 0.90, 0.00, respectively. In this discussion, we have focused on converting dissimilarities to similarities. Conversion in the opposite direction is considered in Exercise 28 on page 131.

In general, any monotonic decreasing function can be used to convert dissimilarities to similarities, or vice versa. Of course, other factors also must be considered when transforming similarities to dissimilarities, or vice versa, or when transforming the values of a proximity measure to a new scale. We have mentioned issues related to preserving meaning, distortion of scale, and requirements of data analysis tools, but this list is certainly not exhaustive.

2.4.2 Similarity and Dissimilarity between Simple Attributes

The proximity of objects with a number of attributes is typically defined by combining the proximities of individual attributes, and thus, we first discuss proximity between objects having a single attribute. Consider objects described by one nominal attribute. What would it mean for two such objects to be similar? Because nominal attributes convey only information about the distinctness of objects, all we can say is that two objects either have the same value or they do not. Hence, in this case similarity is traditionally defined as 1 if attribute values match, and as 0 otherwise. A dissimilarity would be defined in the opposite way: 0 if the attribute values match, and 1 if they do not.

For objects with a single ordinal attribute, the situation is more complicated because information about order should be taken into account. Consider an attribute that measures the quality of a product, e.g., a candy bar, on the scale {*poor*, *fair*, *OK*, *good*, *wonderful*}. It would seem reasonable that a product, P1, which is rated *wonderful*, would be closer to a product P2, which is rated *good*, than it would be to a product P3, which is rated *OK*. To make this observation quantitative, the values of the ordinal attribute are often mapped to successive integers, beginning at 0 or 1, e.g., {poor=0, fair=1, OK=2, $good = 3$, $wonderful = 4$}. Then, $d(\text{P1}, \text{P2}) = 3 - 2 = 1$ or, if we want the dissimilarity to fall between 0 and 1, $d(\text{P1}, \text{P2}) = \frac{3-2}{4} = 0.25$. A similarity for ordinal attributes can then be defined as $s = 1 - d$.

Table 2.7. Similarity and dissimilarity for simple attributes

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$	$s = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$
Ordinal	$d = \lvert x - y \rvert / (n-1)$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - d$
Interval or Ratio	$d = \lvert x - y \rvert$	$s = -d$, $s = \frac{1}{1+d}$, $s = e^{-d}$, $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$

This definition of similarity (dissimilarity) for an ordinal attribute should make the reader a bit uneasy since this assumes equal intervals between successive values of the attribute, and this is not necessarily so. Otherwise, we would have an interval or ratio attribute. Is the difference between the values *fair* and *good* really the same as that between the values *OK* and *wonderful*? Probably not, but in practice, our options are limited, and in the absence of more information, this is the standard approach for defining proximity between ordinal attributes.

For interval or ratio attributes, the natural measure of dissimilarity between two objects is the absolute difference of their values. For example, we might compare our current weight and our weight a year ago by saying "I am ten pounds heavier." In cases such as these, the dissimilarities typically range from 0 to ∞, rather than from 0 to 1. The similarity of interval or ratio attributes is typically expressed by transforming a dissimilarity into a similarity, as previously described.

Table 2.7 summarizes this discussion. In this table, x and y are two objects that have one attribute of the indicated type. Also, $d(x, y)$ and $s(x, y)$ are the dissimilarity and similarity between x and y, respectively. Other approaches are possible; these are the most common ones.

The following two sections consider more complicated measures of proximity between objects that involve multiple attributes: (1) dissimilarities between data objects and (2) similarities between data objects. This division allows us to more naturally display the underlying motivations for employing various proximity measures. We emphasize, however, that similarities can be transformed into dissimilarities and vice versa using the approaches described earlier.

2.4.3 Dissimilarities between Data Objects

In this section, we discuss various kinds of dissimilarities. We begin with a discussion of distances, which are dissimilarities with certain properties, and then provide examples of more general kinds of dissimilarities.

Distances

We first present some examples, and then offer a more formal description of distances in terms of the properties common to all distances. The **Euclidean distance**, d, between two points, $\mathbf{x}$ and $\mathbf{y}$, in one-, two-, three-, or higher-dimensional space, is given by the following familiar formula:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}, \tag{2.1}$$

where n is the number of dimensions and x_k and y_k are, respectively, the k^{th} attributes (components) of $\mathbf{x}$ and $\mathbf{y}$. We illustrate this formula with Figure 2.15 and Tables 2.8 and 2.9, which show a set of points, the x and y coordinates of these points, and the **distance matrix** containing the pairwise distances of these points.

The Euclidean distance measure given in Equation 2.1 is generalized by the **Minkowski** distance metric shown in Equation 2.2,

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{1/r}, \tag{2.2}$$

where r is a parameter. The following are the three most common examples of Minkowski distances.

- $r = 1$. City block (Manhattan, taxicab, L_1 norm) distance. A common example is the **Hamming distance**, which is the number of bits that is different between two objects that have only binary attributes, i.e., between two binary vectors.
- $r = 2$. Euclidean distance (L_2 norm).
- $r = \infty$. Supremum (L_{max} or L_∞ norm) distance. This is the maximum difference between any attribute of the objects. More formally, the L_∞

distance is defined by Equation 2.3

$$d(\mathbf{x}, \mathbf{y}) = \lim_{r \to \infty} \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{1/r}. \tag{2.3}$$

The r parameter should not be confused with the number of dimensions (attributes) n. The Euclidean, Manhattan, and supremum distances are defined for all values of n: $1, 2, 3, \ldots$, and specify different ways of combining the differences in each dimension (attribute) into an overall distance.

Tables 2.10 and 2.11, respectively, give the proximity matrices for the L_1 and L_∞ distances using data from Table 2.8. Notice that all these distance matrices are symmetric; i.e., the ij^{th} entry is the same as the ji^{th} entry. In Table 2.9, for instance, the fourth row of the first column and the fourth column of the first row both contain the value 5.1.

Distances, such as the Euclidean distance, have some well-known properties. If $d(\mathbf{x}, \mathbf{y})$ is the distance between two points, $\mathbf{x}$ and $\mathbf{y}$, then the following properties hold.

1. **Positivity**
   (a) $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all $\mathbf{x}$ and $\mathbf{y}$,
   (b) $d(\mathbf{x}, \mathbf{y}) = 0$ only if $\mathbf{x} = \mathbf{y}$.
2. **Symmetry**
   $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}$ and $\mathbf{y}$.
3. **Triangle Inequality**
   $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$ for all points $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$.

Measures that satisfy all three properties are known as **metrics**. Some people use the term distance only for dissimilarity measures that satisfy these properties, but that practice is often violated. The three properties described here are useful, as well as mathematically pleasing. Also, if the triangle inequality holds, then this property can be used to increase the efficiency of techniques (including clustering) that depend on distances possessing this property. (See Exercise 30.) Nonetheless, many dissimilarities do not satisfy one or more of the metric properties. Example 2.14 illustrates such a measure.

Example 2.14 (Non-metric Dissimilarities: Set Differences)**.** This example is based on the notion of the difference of two sets, as defined in set theory. Given two sets A and B, $A - B$ is the set of elements of A that are not in

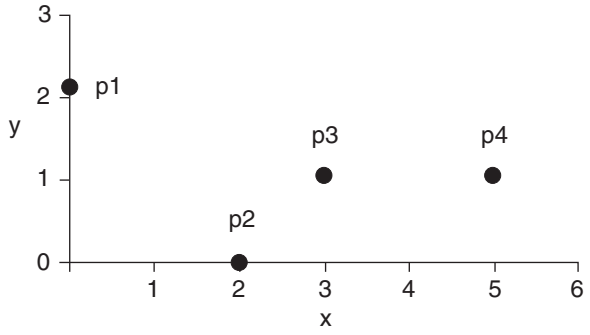


Figure 2.15. Four two-dimensional points.

Table 2.8. x and y coordinates of four points.

point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Table 2.9. Euclidean distance matrix for Table 2.8.

	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

Table 2.10. L_1 distance matrix for Table 2.8.

L_1	p1	p2	p3	p4
p1	0.0	4.0	4.0	6.0
p2	4.0	0.0	2.0	4.0
p3	4.0	2.0	0.0	2.0
p4	6.0	4.0	2.0	0.0

Table 2.11. L_∞ distance matrix for Table 2.8.

L_∞	p1	p2	p3	p4
p1	0.0	2.0	3.0	5.0
p2	2.0	0.0	1.0	3.0
p3	3.0	1.0	0.0	2.0
p4	5.0	3.0	2.0	0.0

B. For example, if $A = \{1, 2, 3, 4\}$ and $B = \{2, 3, 4\}$, then $A - B = \{1\}$ and $B - A = \emptyset$, the empty set. We can define the distance d between two sets A and B as $d(A, B) = size(A - B)$, where $size$ is a function returning the number of elements in a set. This distance measure, which is an integer value greater than or equal to 0, does not satisfy the second part of the positivity property, the symmetry property, or the triangle inequality. However, these properties can be made to hold if the dissimilarity measure is modified as follows: $d(A, B) = size(A-B)+size(B-A)$. See Exercise 26 on page 131. ■

2.4.4 Similarities between Data Objects

For similarities, the triangle inequality (or the analogous property) typically does not hold, but symmetry and positivity typically do. To be explicit, if

$s(\mathbf{x}, \mathbf{y})$ is the similarity between points $\mathbf{x}$ and $\mathbf{y}$, then the typical properties of similarities are the following:

1. $s(\mathbf{x}, \mathbf{y}) = 1$ only if $\mathbf{x} = \mathbf{y}$. $(0 \leq s \leq 1)$
2. $s(\mathbf{x}, \mathbf{y}) = s(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}$ and $\mathbf{y}$. (Symmetry)

There is no general analog of the triangle inequality for similarity measures. It is sometimes possible, however, to show that a similarity measure can easily be converted to a metric distance. The cosine and Jaccard similarity measures, which are discussed shortly, are two examples. Also, for specific similarity measures, it is possible to derive mathematical bounds on the similarity between two objects that are similar in spirit to the triangle inequality.

Example 2.15 (A Non-symmetric Similarity Measure)**.** Consider an experiment in which people are asked to classify a small set of characters as they flash on a screen. The **confusion matrix** for this experiment records how often each character is classified as itself, and how often each is classified as another character. Using the confusion matrix, we can define a similarity measure between a character x and a character y as the number of times that x is misclassified as y, but note that this measure is not symmetric. For example, suppose that "0" appeared 200 times and was classified as a "0" 160 times, but as an "o" 40 times. Likewise, suppose that "o" appeared 200 times and was classified as an "o" 170 times, but as "0" only 30 times. Then, s(0,o) = 40, but s(o, 0) = 30. In such situations, the similarity measure can be made symmetric by setting $s'(x, y) = s'(y, x) = (s(x, y) + s(y, x))/2$, where s' indicates the new similarity measure. ■

2.4.5 Examples of Proximity Measures

This section provides specific examples of some similarity and dissimilarity measures.

Similarity Measures for Binary Data

Similarity measures between objects that contain only binary attributes are called **similarity coefficients**, and typically have values between 0 and 1. A value of 1 indicates that the two objects are completely similar, while a value of 0 indicates that the objects are not at all similar. There are many rationales for why one coefficient is better than another in specific instances.

Let $\mathbf{x}$ and $\mathbf{y}$ be two objects that consist of n binary attributes. The comparison of two such objects, i.e., two binary vectors, leads to the following

four quantities (frequencies):

f_{00} = the number of attributes where **x** is 0 and **y** is 0
f_{01} = the number of attributes where **x** is 0 and **y** is 1
f_{10} = the number of attributes where **x** is 1 and **y** is 0
f_{11} = the number of attributes where **x** is 1 and **y** is 1

Simple Matching Coefficient One commonly used similarity coefficient is the **simple matching coefficient** (SMC), which is defined as

$$SMC = \frac{\text{number of matching attribute values}}{\text{number of attributes}} = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}}. \quad (2.4)$$

This measure counts both presences and absences equally. Consequently, the SMC could be used to find students who had answered questions similarly on a test that consisted only of true/false questions.

Jaccard Coefficient Suppose that **x** and **y** are data objects that represent two rows (two transactions) of a transaction matrix (see Section 2.1.2). If each asymmetric binary attribute corresponds to an item in a store, then a 1 indicates that the item was purchased, while a 0 indicates that the product was not purchased. Because the number of products not purchased by any customer far outnumbers the number of products that were purchased, a similarity measure such as SMC would say that all transactions are very similar. As a result, the Jaccard coefficient is frequently used to handle objects consisting of asymmetric binary attributes. The **Jaccard coefficient**, which is often symbolized by J, is given by the following equation:

$$J = \frac{\text{number of matching presences}}{\text{number of attributes not involved in 00 matches}} = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}. \quad (2.5)$$

Example 2.16 (The SMC and Jaccard Similarity Coefficients)**.** To illustrate the difference between these two similarity measures, we calculate SMC and J for the following two binary vectors.

$\mathbf{x} = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
$\mathbf{y} = (0, 0, 0, 0, 0, 0, 1, 0, 0, 1)$

$f_{01} = 2$ the number of attributes where **x** was 0 and **y** was 1

$f_{10} = 1$ the number of attributes where $\mathbf{x}$ was 1 and $\mathbf{y}$ was 0
$f_{00} = 7$ the number of attributes where $\mathbf{x}$ was 0 and $\mathbf{y}$ was 0
$f_{11} = 0$ the number of attributes where $\mathbf{x}$ was 1 and $\mathbf{y}$ was 1

$$SMC = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}} = \frac{0 + 7}{2 + 1 + 0 + 7} = 0.7$$

$$J = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} = \frac{0}{2 + 1 + 0} = 0$$

■

Cosine Similarity

Documents are often represented as vectors, where each component (attribute) represents the frequency with which a particular term (word) occurs in the document. Even though documents have thousands or tens of thousands of attributes (terms), each document is sparse since it has relatively few non-zero attributes. Thus, as with transaction data, similarity should not depend on the number of shared 0 values because any two documents are likely to "not contain" many of the same words, and therefore, if 0–0 matches are counted, most documents will be highly similar to most other documents. Therefore, a similarity measure for documents needs to ignores 0–0 matches like the Jaccard measure, but also must be able to handle non-binary vectors. The **cosine similarity**, defined next, is one of the most common measures of document similarity. If $\mathbf{x}$ and $\mathbf{y}$ are two document vectors, then

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \, \|\mathbf{y}\|} = \frac{\mathbf{x}'\mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|}, \tag{2.6}$$

where $'$ indicates vector or matrix transpose and $\langle \mathbf{x}, \mathbf{y} \rangle$ indicates the inner product of the two vectors,

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=1}^{n} x_k y_k = \mathbf{x}'\mathbf{y}, \tag{2.7}$$

and $\|\mathbf{x}\|$ is the length of vector $\mathbf{x}$, $\|\mathbf{x}\| = \sqrt{\sum_{k=1}^{n} x_k^2} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\mathbf{x}'\mathbf{x}}$.

The inner product of two vectors works well for asymmetric attributes since it depends only on components that are non-zero in both vectors. Hence, the similarity between two documents depends only upon the words that appear in both of them.

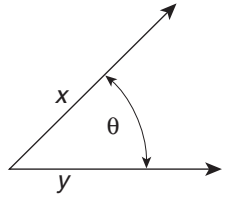


Figure 2.16. Geometric illustration of the cosine measure.

Example 2.17 (Cosine Similarity between Two Document Vectors)**.** This example calculates the cosine similarity for the following two data objects, which might represent document vectors:

$\mathbf{x} = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$
$\mathbf{y} = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$

$\langle \mathbf{x}, \mathbf{y} \rangle = 3 \times 1 + 2 \times 0 + 0 \times 0 + 5 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 2 \times 1 + 0 \times 0 + 0 \times 2 = 5$
$\|\mathbf{x}\| = \sqrt{3 \times 3 + 2 \times 2 + 0 \times 0 + 5 \times 5 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 2 \times 2 + 0 \times 0 + 0 \times 0} = 6.48$
$\|\mathbf{y}\| = \sqrt{1 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 + 0 \times 0 + 2 \times 2} = 2.45$
$\mathbf{cos}(\mathbf{x}, \mathbf{y}) = \mathbf{0.31}$

■

As indicated by Figure 2.16, cosine similarity really is a measure of the (cosine of the) angle between $\mathbf{x}$ and $\mathbf{y}$. Thus, if the cosine similarity is 1, the angle between $\mathbf{x}$ and $\mathbf{y}$ is 0°, and $\mathbf{x}$ and $\mathbf{y}$ are the same except for length. If the cosine similarity is 0, then the angle between $\mathbf{x}$ and $\mathbf{y}$ is 90°, and they do not share any terms (words).

Equation 2.6 also can be written as Equation 2.8.

$$\cos(\mathbf{x}, \mathbf{y}) = \left\langle \frac{\mathbf{x}}{\|\mathbf{x}\|}, \frac{\mathbf{y}}{\|\mathbf{y}\|} \right\rangle = \langle \mathbf{x}', \mathbf{y}' \rangle, \tag{2.8}$$

where $\mathbf{x}' = \mathbf{x}/\|\mathbf{x}\|$ and $\mathbf{y}' = \mathbf{y}/\|\mathbf{y}\|$. Dividing $\mathbf{x}$ and $\mathbf{y}$ by their lengths normalizes them to have a length of 1. This means that cosine similarity does not take the *length* of the two data objects into account when computing similarity. (Euclidean distance might be a better choice when length is important.) For vectors with a length of 1, the cosine measure can be calculated by taking a simple inner product. Consequently, when many cosine similarities between objects are being computed, normalizing the objects to have unit length can reduce the time required.

Extended Jaccard Coefficient (Tanimoto Coefficient)

The extended Jaccard coefficient can be used for document data and that reduces to the Jaccard coefficient in the case of binary attributes. This coefficient, which we shall represent as EJ, is defined by the following equation:

$$EJ(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \langle \mathbf{x}, \mathbf{y} \rangle} = \frac{\mathbf{x}'\mathbf{y}}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \mathbf{x}'\mathbf{y}}. \tag{2.9}$$

Correlation

Correlation is frequently used to measure the linear relationship between two sets of values that are observed together. Thus, correlation can measure the relationship between two variables (height and weight) or between two objects (a pair of temperature time series). Correlation is used much more frequently to measure the similarity between attributes since the values in two data objects come from different attributes, which can have very different attribute types and scales. There are many types of correlation, and indeed correlation is sometimes used in a general sense to mean the relationship between two sets of values that are observed together. In this discussion, we will focus on a measure appropriate for numerical values.

Specifically, **Pearson's correlation** between two sets of numerical values, i.e., two vectors, $\mathbf{x}$ and $\mathbf{y}$, is defined by the following equation:

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\text{covariance}(\mathbf{x}, \mathbf{y})}{\text{standard\_deviation}(\mathbf{x}) \times \text{standard\_deviation}(\mathbf{y})} = \frac{s_{xy}}{s_x \; s_y}, \tag{2.10}$$

where we use the following standard statistical notation and definitions:

$$\text{covariance}(\mathbf{x}, \mathbf{y}) = s_{xy} = \frac{1}{n-1} \sum_{k=1}^{n} (x_k - \overline{x})(y_k - \overline{y}) \tag{2.11}$$

$$\begin{aligned} \text{standard\_deviation}(\mathbf{x}) &= s_x = \sqrt{\frac{1}{n-1} \sum_{k=1}^{n} (x_k - \overline{x})^2} \\ \text{standard\_deviation}(\mathbf{y}) &= s_y = \sqrt{\frac{1}{n-1} \sum_{k=1}^{n} (y_k - \overline{y})^2} \end{aligned}$$

$$\overline{x} = \frac{1}{n}\sum_{k=1}^{n} x_k \text{ is the mean of } \mathbf{x}$$
$$\overline{y} = \frac{1}{n}\sum_{k=1}^{n} y_k \text{ is the mean of } \mathbf{y}$$

Example 2.18 (Perfect Correlation)**.** Correlation is always in the range -1 to 1. A correlation of 1 (-1) means that $\mathbf{x}$ and $\mathbf{y}$ have a perfect positive (negative) linear relationship; that is, $x_k = ay_k + b$, where a and b are constants. The following two vectors $\mathbf{x}$ and $\mathbf{y}$ illustrate cases where the correlation is -1 and $+1$, respectively. In the first case, the means of $\mathbf{x}$ and $\mathbf{y}$ were chosen to be 0, for simplicity.

$\mathbf{x} = (-3,\ \ 6,\ 0,\ \ 3, -6)$
$\mathbf{y} = (\ \ 1, -2,\ 0, -1,\ \ 2)$
$corr(\mathbf{x}, \mathbf{y}) = -1 \quad x_k = -3y_k$

$\mathbf{x} = (3, 6, 0, 3, 6)$
$\mathbf{y} = (1, 2, 0, 1, 2)$
$corr(\mathbf{x}, \mathbf{y}) = 1 \quad x_k = 3y_k$ ■

Example 2.19 (Nonlinear Relationships)**.** If the correlation is 0, then there is no linear relationship between the two sets of values. However, nonlinear relationships can still exist. In the following example, $y_k = x_k^2$, but their correlation is 0.

$\mathbf{x} = (-3, -2, -1,\ 0,\ 1,\ 2,\ 3)$
$\mathbf{y} = (\ \ 9,\ \ \ 4,\ \ \ 1,\ 0,\ 1,\ 4,\ 9)$

■

Example 2.20 (Visualizing Correlation)**.** It is also easy to judge the correlation between two vectors $\mathbf{x}$ and $\mathbf{y}$ by plotting pairs of corresponding values of $\mathbf{x}$ and $\mathbf{y}$ in a scatter plot. Figure 2.17 shows a number of these scatter plots when $\mathbf{x}$ and $\mathbf{y}$ consist of a set of 30 pairs of values that are randomly generated (with a normal distribution) so that the correlation of $\mathbf{x}$ and $\mathbf{y}$ ranges from -1 to 1. Each circle in a plot represents one of the 30 pairs of $\mathbf{x}$ and $\mathbf{y}$ values; its x coordinate is the value of that pair for $\mathbf{x}$, while its y coordinate is the value of the same pair for $\mathbf{y}$. ■

If we transform $\mathbf{x}$ and $\mathbf{y}$ by subtracting off their means and then normalizing them so that their lengths are 1, then their correlation can be calculated by taking the dot product. Let us refer to these transformed vectors of $\mathbf{x}$ and $\mathbf{y}$ as $\mathbf{x}'$ and $\mathbf{y}'$, respectively. (Notice that this transformation is not the same

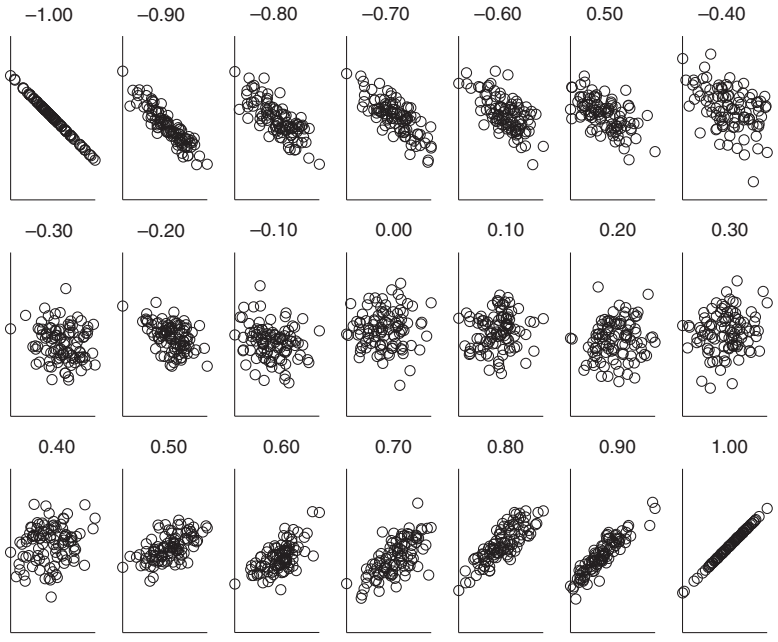


Figure 2.17. Scatter plots illustrating correlations from -1 to 1.

as the standardization used in other contexts, where we subtract the means and divide by the standard deviations, as discussed in Section 2.3.7.) This transformation highlights an interesting relationship between the correlation measure and the cosine measure. Specifically, the correlation between $\mathbf{x}$ and $\mathbf{y}$ is identical to the cosine between $\mathbf{x}'$ and $\mathbf{y}'$. However, the cosine between $\mathbf{x}$ and $\mathbf{y}$ is not the same as the cosine between $\mathbf{x}'$ and $\mathbf{y}'$, even though they both have the same correlation measure. In general, the correlation between two vectors is equal to the cosine measure only in the special case when the means of the two vectors are 0.

Differences Among Measures For Continuous Attributes

In this section, we illustrate the difference among the three proximity measures for continuous attributes that we have just defined: cosine, correlation, and Minkowski distance. Specifically, we consider two types of data transformations that are commonly used, namely, scaling (multiplication) by a constant factor and translation (addition) by a constant value. A proximity measure is considered to be invariant to a data transformation if its value remains unchanged even after performing the transformation. Table 2.12 compares the

Table 2.12. Properties of cosine, correlation, and Minkowski distance measures.

Property	Cosine	Correlation	Minkowski Distance
Invariant to scaling (multiplication)	Yes	Yes	No
Invariant to translation (addition)	No	Yes	No

behavior of cosine, correlation, and Minkowski distance measures regarding their invariance to scaling and translation operations. It can be seen that while correlation is invariant to both scaling and translation, cosine is only invariant to scaling but not to translation. Minkowski distance measures, on the other hand, are sensitive to both scaling and translation and are thus invariant to neither.

Let us consider an example to demonstrate the significance of these differences among different proximity measures.

Example 2.21 (Comparing proximity measures)**.** Consider the following two vectors $\mathbf{x}$ and $\mathbf{y}$ with seven numeric attributes.

$\mathbf{x} = (1, 2, 4, 3, 0, 0, 0)$
$\mathbf{y} = (1, 2, 3, 4, 0, 0, 0)$

It can be seen that both $\mathbf{x}$ and $\mathbf{y}$ have 4 non-zero values, and the values in the two vectors are mostly the same, except for the third and the fourth components. The cosine, correlation, and Euclidean distance between the two vectors can be computed as follows.

$$\begin{aligned} \mathbf{cos}(\mathbf{x}, \mathbf{y}) &= \frac{29}{\sqrt{30} \times \sqrt{30}} &&= 0.9667 \\ \mathbf{correlation}(\mathbf{x}, \mathbf{y}) &= \frac{2.3571}{1.5811 \times 1.5811} &&= 0.9429 \\ \mathbf{Euclidean\ distance}(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}|| &&= 1.4142 \end{aligned}$$

Not surprisingly, $\mathbf{x}$ and $\mathbf{y}$ have a cosine and correlation measure close to 1, while the Euclidean distance between them is small, indicating that they are quite similar. Now let us consider the vector $\mathbf{y_s}$, which is a scaled version of $\mathbf{y}$ (multiplied by a constant factor of 2), and the vector $\mathbf{y_t}$, which is constructed by translating $\mathbf{y}$ by 5 units as follows.

$\mathbf{y_s} = 2 \times \mathbf{y} = (2, 4, 6, 8, 0, 0, 0)$

Table 2.13. Similarity between $(\mathbf{x}, \mathbf{y})$, $(\mathbf{x}, \mathbf{y_s})$, and $(\mathbf{x}, \mathbf{y_t})$.

Measure	$(\mathbf{x}, \mathbf{y})$	$(\mathbf{x}, \mathbf{y_s})$	$(\mathbf{x}, \mathbf{y_t})$
Cosine	0.9667	0.9667	0.7940
Correlation	0.9429	0.9429	0.9429
Euclidean Distance	1.4142	5.8310	14.2127

$\mathbf{y_t} = \mathbf{y} + 5 = (6, 7, 8, 9, 5, 5, 5)$

We are interested in finding whether $\mathbf{y_s}$ and $\mathbf{y_t}$ show the same proximity with $\mathbf{x}$ as shown by the original vector $\mathbf{y}$. Table 2.13 shows the different measures of proximity computed for the pairs $(\mathbf{x}, \mathbf{y})$, $(\mathbf{x}, \mathbf{y_s})$, and $(\mathbf{x}, \mathbf{y_t})$. It can be seen that the value of correlation between $\mathbf{x}$ and $\mathbf{y}$ remains unchanged even after replacing $\mathbf{y}$ with $\mathbf{y_s}$ or $\mathbf{y_t}$. However, the value of cosine remains equal to 0.9667 when computed for $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}, \mathbf{y_s})$, but significantly reduces to 0.7940 when computed for $(\mathbf{x}, \mathbf{y_t})$. This highlights the fact that cosine is invariant to the scaling operation but not to the translation operation, in contrast with the correlation measure. The Euclidean distance, on the other hand, shows different values for all three pairs of vectors, as it is sensitive to both scaling and translation.

We can observe from this example that different proximity measures behave differently when scaling or translation operations are applied on the data. The choice of the right proximity measure thus depends on the desired notion of similarity between data objects that is meaningful for a given application. For example, if $\mathbf{x}$ and $\mathbf{y}$ represented the frequencies of different words in a document-term matrix, it would be meaningful to use a proximity measure that remains unchanged when $\mathbf{y}$ is replaced by $\mathbf{y_s}$, because $\mathbf{y_s}$ is just a scaled version of $\mathbf{y}$ with the same distribution of words occurring in the document. However, $\mathbf{y_t}$ is different from $\mathbf{y}$, since it contains a large number of words with non-zero frequencies that do not occur in $\mathbf{y}$. Because cosine is invariant to scaling but not to translation, it will be an ideal choice of proximity measure for this application.

Consider a different scenario in which $\mathbf{x}$ represents a location's temperature measured on the Celsius scale for seven days. Let $\mathbf{y}$, $\mathbf{y_s}$, and $\mathbf{y_t}$ be the temperatures measured on those days at a different location, but using three different measurement scales. Note that different units of temperature have different offsets (e.g., Celsius and Kelvin) and different scaling factors (e.g., Celsius and Fahrenheit). It is thus desirable to use a proximity measure that captures the proximity between temperature values without being affected by

the measurement scale. Correlation would then be the ideal choice of proximity measure for this application, as it is invariant to both scaling and translation.

As another example, consider a scenario where $\mathbf{x}$ represents the amount of precipitation (in cm) measured at seven locations. Let $\mathbf{y}$, $\mathbf{y_s}$, and $\mathbf{y_t}$ be estimates of the precipitation at these locations, which are predicted using three different models. Ideally, we would like to choose a model that accurately reconstructs the measurements in $\mathbf{x}$ without making any error. It is evident that $\mathbf{y}$ provides a good approximation of the values in $\mathbf{x}$, whereas $\mathbf{y_s}$ and $\mathbf{y_t}$ provide poor estimates of precipitation, even though they do capture the trend in precipitation across locations. Hence, we need to choose a proximity measure that penalizes any difference in the model estimates from the actual observations, and is sensitive to both the scaling and translation operations. The Euclidean distance satisfies this property and thus would be the right choice of proximity measure for this application. Indeed, the Euclidean distance is commonly used in computing the accuracy of models, which will be discussed later in Chapter 3.

■

2.4.6 Mutual Information

Like correlation, mutual information is used as a measure of similarity between two sets of paired values that is sometimes used as an alternative to correlation, particularly when a nonlinear relationship is suspected between the pairs of values. This measure comes from information theory, which is the study of how to formally define and quantify information. Indeed, mutual information is a measure of how much information one set of values provides about another, given that the values come in pairs, e.g., height and weight. If the two sets of values are independent, i.e., the value of one tells us nothing about the other, then their mutual information is 0. On the other hand, if the two sets of values are completely dependent, i.e., knowing the value of one tells us the value of the other and vice-versa, then they have maximum mutual information. Mutual information does not have a maximum value, but we will define a normalized version of it that ranges between 0 and 1.

To define mutual information, we consider two sets of values, X and Y, which occur in pairs (X, Y). We need to measure the average information in a single set of values, i.e., either in X or in Y, and in the pairs of their values. This is commonly measured by entropy. More specifically, assume X and Y are discrete, that is, X can take m distinct values, $u_1, u_2, \ldots, u_m$ and Y can take n distinct values, $v_1, v_2, \ldots, v_n$. Then their individual and joint entropy

can be defined in terms of the probabilities of each value and pair of values as follows:

$$H(X) = -\sum_{j=1}^{m} P(X = u_j) \log_2 P(X = u_j) \tag{2.12}$$

$$H(Y) = -\sum_{k=1}^{n} P(Y = v_k) \log_2 P(Y = v_k) \tag{2.13}$$

$$H(X, Y) = -\sum_{j=1}^{m} \sum_{k=1}^{n} P(X = u_j, Y = v_k) \log_2 P(X = u_j, Y = v_k) \tag{2.14}$$

where if the probability of a value or combination of values is 0, then $0 \log_2(0)$ is conventionally taken to be 0.

The mutual information of X and Y can now be defined straightforwardly:

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \tag{2.15}$$

Note that $H(X, Y)$ is symmetric, i.e., $H(X, Y) = H(Y, X)$, and thus mutual information is also symmetric, i.e., $I(X, Y) = I(Y)$.

Practically, X and Y are either the values in two attributes or two rows of the same data set. In Example 2.22, we will represent those values as two vectors $\mathbf{x}$ and $\mathbf{y}$ and calculate the probability of each value or pair of values from the frequency with which values or pairs of values occur in $\mathbf{x}$, $\mathbf{y}$ and (x_i, y_i), where x_i is the i^{th} component of $\mathbf{x}$ and y_i is the i^{th} component of $\mathbf{y}$. Let us illustrate using a previous example.

Example 2.22 (Evaluating Nonlinear Relationships with Mutual Information)**.** Recall Example 2.19 where $y_k = x_k^2$, but their correlation was 0.

$\mathbf{x} = (-3, -2, -1,\ 0,\ 1,\ 2,\ 3)$
$\mathbf{y} = (\ 9,\ \ 4,\ \ 1,\ 0,\ 1,\ 4,\ 9)$

From Figure 2.22, $I(\mathbf{x}, \mathbf{y}) = H(\mathbf{x}) + H(\mathbf{y}) - H(\mathbf{x}, \mathbf{y}) = 1.9502$. Although a variety of approaches to normalize mutual information are possible—see Bibliographic Notes—for this example, we will apply one that divides the mutual information by $\log_2(\min(m, n))$ and produces a result between 0 and 1. This yields a value of $1.9502/\log_2(4)) = 0.9751$. Thus, we can see that $\mathbf{x}$ and $\mathbf{y}$ are strongly related. They are not perfectly related because given a value of $\mathbf{y}$ there is, except for $\mathbf{y} = 0$, some ambiguity about the value of $\mathbf{x}$. Notice that for $\mathbf{y} = -\mathbf{x}$, the normalized mutual information would be 1. ■

Figure 2.18. Computation of mutual information.

Table 2.14. Entropy for $\mathbf{x}$

x_j	$P(\mathbf{x} = x_j)$	$-P(\mathbf{x} = x_j) \log_2 P(\mathbf{x} = x_j)$
-3	1/7	0.4011
-2	1/7	0.4011
-1	1/7	0.4011
0	1/7	0.4011
1	1/7	0.4011
2	1/7	0.4011
3	1/7	0.4011
$H(\mathbf{x})$		2.8074

Table 2.15. Entropy for $\mathbf{y}$

y_k	$P(\mathbf{y} = y_k)$	$-P(\mathbf{y} = y_k) \log_2(P(\mathbf{y} = y_k)$
9	2/7	0.5164
4	2/7	0.5164
1	2/7	0.5164
0	1/7	0.4011
$H(\mathbf{y})$		1.9502

Table 2.16. Joint entropy for $\mathbf{x}$ and $\mathbf{y}$

x_j	y_k	$P(\mathbf{x} = x_j, \mathbf{y} = x_k)$	$-P(\mathbf{x} = x_j, \mathbf{y} = x_k) \log_2 P(\mathbf{x} = x_j, \mathbf{y} = x_k)$
-3	9	1/7	0.4011
-2	4	1/7	0.4011
-1	1	1/7	0.4011
0	0	1/7	0.4011
1	1	1/7	0.4011
2	4	1/7	0.4011
3	9	1/7	0.4011
$H(\mathbf{x}, \mathbf{y})$			2.8074

2.4.7 Kernel Functions*

It is easy to understand how similarity and distance might be useful in an application such as clustering, which tries to group similar objects together. What is much less obvious is that many other data analysis tasks, including predictive modeling and dimensionality reduction, can be expressed in terms of pairwise "proximities" of data objects. More specifically, many data analysis problems can be mathematically formulated to take as input, a **kernel matrix**, $\mathbf{K}$, which can be considered a type of proximity matrix. Thus, an initial preprocessing step is used to convert the input data into a kernel matrix, which is the input to the data analysis algorithm.

More formally, if a data set has m data objects, then $\mathbf{K}$ is an m by m matrix. If $\mathbf{x}_i$ and $\mathbf{x}_j$ are the i^{th} and j^{th} data objects, respectively, then k_{ij}, the ij^{th} entry of $\mathbf{K}$, is computed by a **kernel function**:

$$k_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{2.16}$$

As we will see in the material that follows, the use of a kernel matrix allows both wider applicability of an algorithm to various kinds of data and an ability to model nonlinear relationships with algorithms that are designed only for detecting linear relationships.

Kernels make an algorithm data independent If an algorithm uses a kernel matrix, then it can be used with any type of data for which a kernel function can be designed. This is illustrated by Algorithm 2.1. Although only some data analysis algorithms can be modified to use a kernel matrix as input, this approach is extremely powerful because it allows such an algorithm to be used with almost any type of data for which an appropriate kernel function can be defined. Thus, a classification algorithm can be used, for example, with record data, string data, or graph data. If an algorithm can be reformulated to use a kernel matrix, then its applicability to different types of data increases dramatically. As we will see in later chapters, many clustering, classification, and anomaly detection algorithms work only with similarities or distances, and thus, can be easily modified to work with kernels.

Algorithm 2.1 Basic kernel algorithm.

1: Read in the m data objects in the data set.
2: Compute the kernel matrix, $\mathbf{K}$ by applying the kernel function, κ, to each pair of data objects.
3: Run the data analysis algorithm with $\mathbf{K}$ as input.
4: Return the analysis result, e.g., predicted class or cluster labels.

Mapping data into a higher dimensional data space can allow modeling of nonlinear relationships There is yet another, equally important, aspect of kernel based data algorithms—their ability to model nonlinear relationships with algorithms that model only linear relationships. Typically, this works by first transforming (mapping) the data from a lower dimensional data space to a higher dimensional space.

Example 2.23 (Mapping Data to a Higher Dimensional Space)**.** Consider the relationship between two variables x and y given by the following equation, which defines an ellipse in two dimensions (Figure 2.19(a)):

$$4x^2 + 9xy + 7y^2 = 10 \tag{2.17}$$

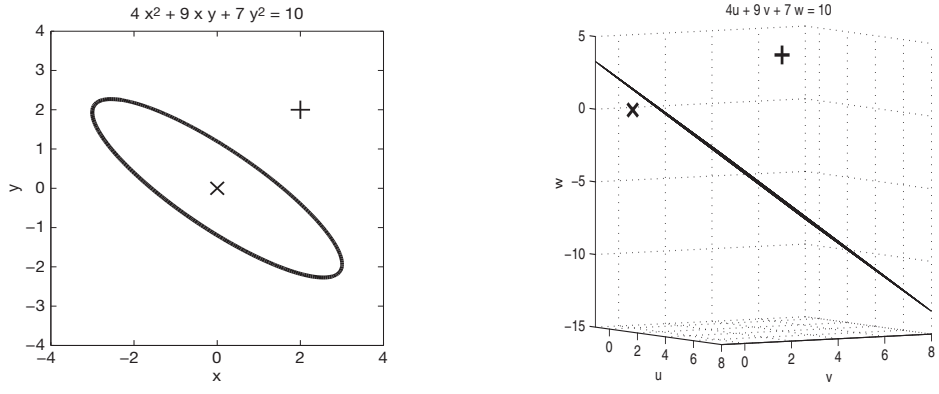
We can map our two dimensional data to three dimensions by creating three new variables, u, v, and w, which are defined as follows:

$$\begin{aligned} w &= x^2 \\ u &= xy \\ v &= y^2 \end{aligned}$$

As a result, we can now express Equation 2.17 as a linear one. This equation describes a plane in three dimensions. Points on the ellipse will lie on that plane, while points inside and outside the ellipse will lie on opposite sides of the plane. See Figure 2.19(b). The viewpoint of this 3D plot is along the surface of the separating plane so that the plane appears as a line.

$$4u + 9v + 7w = 10 \tag{2.18}$$

■



(a) Ellipse and two points in 2 dimensions.

(b) Data mapped to 3 dimensions.

Figure 2.19. Mapping data to a higher dimensional space: two to three dimensions.

The Kernel Trick The approach illustrated above shows the value in mapping data to higher dimensional space, an operation that is integral to kernel-based methods. Conceptually, we first define a function φ that maps data points $\mathbf{x}$ and $\mathbf{y}$ to data points $\varphi(\mathbf{x})$ and $\varphi(\mathbf{y})$ in a higher dimensional space

such that the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ gives the desired measure of proximity of $\mathbf{x}$ and $\mathbf{y}$. It may seem that we have potentially sacrificed a great deal by using such an approach, because we can greatly expand the size of our data, increase the computational complexity of our analysis, and encounter problems with the curse of dimensionality by computing similarity in a high-dimensional space. However, this is not the case since these problems can be avoided by defining a kernel function κ that can compute the same similarity value, but with the data points in the original space, i.e., $\kappa(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle$. This is known as the *kernel trick*. Despite the name, the kernel trick has a very solid mathematical foundation and is a remarkably powerful approach for data analysis.

Not every function of a pair of data objects satisfies the properties needed for a kernel function, but it has been possible to design many useful kernels for a wide variety of data types. For example, three common kernel functions are the polynomial, Gaussian (radial basis function (RBF)), and sigmoid kernels. If $\mathbf{x}$ and $\mathbf{y}$ are two data objects, specifically, two data vectors, then these two kernel functions can be expressed as follows, respectively:

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}'\mathbf{y} + c)^d \tag{2.19}$$

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||/2\sigma^2) \tag{2.20}$$

$$\kappa(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x}'\mathbf{y} + c) \tag{2.21}$$

where α and $c \geq 0$ are constants, d is an integer parameter that gives the polynomial degree, $||\mathbf{x} - \mathbf{y}||$ is the length of the vector $\mathbf{x} - \mathbf{y}$ and $\sigma > 0$ is a parameter that governs the "spread" of a Gaussian.

Example 2.24 (The Polynomial Kernel)**.** Note that the kernel functions presented in the previous section are computing the same similarity value as would be computed if we actually mapped the data to a higher dimensional space and then computed an inner product there. For example, for the polynomial kernel of degree 2, let φ be the function that maps a two-dimensional data vector $\mathbf{x} = (x_1, x_2)$ to the higher dimensional space. Specifically, let

$$\varphi(\mathbf{x}) = (x_1^2,\ x_2^2,\ \sqrt{2}x_1x_2,\ \sqrt{2c}x_1, \sqrt{2c}x_2,\ c). \tag{2.22}$$

For the higher dimensional space, let the proximity be defined as the inner product of $\varphi(\mathbf{x})$ and $\varphi(\mathbf{y})$, i.e., $\langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle$. Then, as previously mentioned, it can be shown that

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle \tag{2.23}$$

where κ is defined by Equation 2.19 above. Specifically, if $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$, then

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}'\mathbf{y} = (x_1^2 y_1^2,\ x_2^2 y_2^2,\ 2x_1 x_2 y_1 y_2,\ 2cx_1 y_1, 2cx_2 y_2,\ c^2). \quad (2.24)$$

More generally, the kernel trick depends on defining κ and φ so that Equation 2.23 holds. This has been done for a wide variety of kernels. ■

This discussion of kernel-based approaches was intended only to provide a brief introduction to this topic and has omitted many details. A fuller discussion of the kernel-based approach is provided in Section 6.9.4, which discusses these issues in the context of nonlinear support vector machines for classification. More general references for the kernel based analysis can be found in the Bibliographic Notes of this chapter.

2.4.8 Bregman Divergence*

This section provides a brief description of Bregman divergences, which are a family of proximity functions that share some common properties. As a result, it is possible to construct general data mining algorithms, such as clustering algorithms, that work with any Bregman divergence. A concrete example is the K-means clustering algorithm (Section 5.2). Note that this section requires knowledge of vector calculus.

Bregman divergences are loss or distortion functions. To understand the idea of a loss function, consider the following. Let $\mathbf{x}$ and $\mathbf{y}$ be two points, where $\mathbf{y}$ is regarded as the original point and $\mathbf{x}$ is some distortion or approximation of it. For example, $\mathbf{x}$ may be a point that was generated by adding random noise to $\mathbf{y}$. The goal is to measure the resulting distortion or loss that results if $\mathbf{y}$ is approximated by $\mathbf{x}$. Of course, the more similar $\mathbf{x}$ and $\mathbf{y}$ are, the smaller the loss or distortion. Thus, Bregman divergences can be used as dissimilarity functions.

More formally, we have the following definition.

Definition 2.6 (Bregman divergence)**.** Given a strictly convex function ϕ (with a few modest restrictions that are generally satisfied), the Bregman divergence (loss function) $D(\mathbf{x}, \mathbf{y})$ generated by that function is given by the following equation:

$$D(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle \quad (2.25)$$

where $\nabla\phi(\mathbf{y})$ is the gradient of ϕ evaluated at $\mathbf{y}$, $\mathbf{x} - \mathbf{y}$, is the vector difference between $\mathbf{x}$ and $\mathbf{y}$, and $\langle \nabla\phi(\mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle$ is the inner product between $\nabla\phi(\mathbf{y})$

and $(\mathbf{x} - \mathbf{y})$. For points in Euclidean space, the inner product is just the dot product.

$D(\mathbf{x}, \mathbf{y})$ can be written as $D(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - L(\mathbf{x})$, where $L(\mathbf{x}) = \phi(\mathbf{y}) + \langle \nabla\phi(\mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle$ and represents the equation of a plane that is tangent to the function ϕ at $\mathbf{y}$. Using calculus terminology, $L(\mathbf{x})$ is the linearization of ϕ around the point $\mathbf{y}$, and the Bregman divergence is just the difference between a function and a linear approximation to that function. Different Bregman divergences are obtained by using different choices for ϕ.

Example 2.25. We provide a concrete example using squared Euclidean distance, but restrict ourselves to one dimension to simplify the mathematics. Let x and y be real numbers and $\phi(t)$ be the real-valued function, $\phi(t) = t^2$. In that case, the gradient reduces to the derivative, and the dot product reduces to multiplication. Specifically, Equation 2.25 becomes Equation 2.26.

$$D(x, y) = x^2 - y^2 - 2y(x - y) = (x - y)^2 \tag{2.26}$$

The graph for this example, with $y = 1$, is shown in Figure 2.20. The Bregman divergence is shown for two values of x: $x = 2$ and $x = 3$. ■

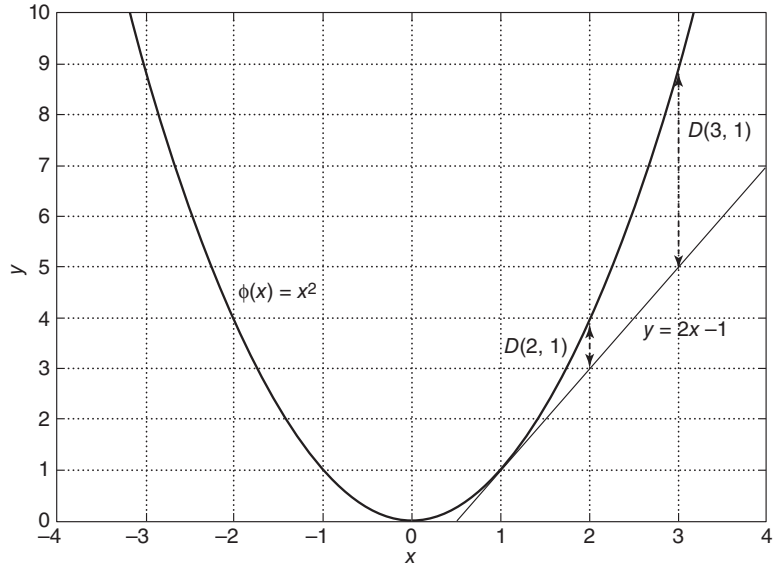


Figure 2.20. Illustration of Bregman divergence.

2.4.9 Issues in Proximity Calculation

This section discusses several important issues related to proximity measures: (1) how to handle the case in which attributes have different scales and/or are correlated, (2) how to calculate proximity between objects that are composed of different types of attributes, e.g., quantitative and qualitative, (3) and how to handle proximity calculations when attributes have different weights; i.e., when not all attributes contribute equally to the proximity of objects.

Standardization and Correlation for Distance Measures

An important issue with distance measures is how to handle the situation when attributes do not have the same range of values. (This situation is often described by saying that "the variables have different scales.") In a previous example, Euclidean distance was used to measure the distance between people based on two attributes: age and income. Unless these two attributes are standardized, the distance between two people will be dominated by income.

A related issue is how to compute distance when there is correlation between some of the attributes, perhaps in addition to differences in the ranges of values. A generalization of Euclidean distance, the **Mahalanobis distance**, is useful when attributes are correlated, have different ranges of values (different variances), and the distribution of the data is approximately Gaussian (normal). Correlated variables have a large impact on standard distance measures since a change in any of the correlated variables is reflected in a change in all the correlated variables. Specifically, the Mahalanobis distance between two objects (vectors) $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\text{Mahalanobis}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})' \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})}, \tag{2.27}$$

where $\boldsymbol{\Sigma}^{-1}$ is the inverse of the covariance matrix of the data. Note that the covariance matrix $\boldsymbol{\Sigma}$ is the matrix whose ij^{th} entry is the covariance of the i^{th} and j^{th} attributes as defined by Equation 2.11.

Example 2.26. In Figure 2.21, there are 1000 points, whose x and y attributes have a correlation of 0.6. The distance between the two large points at the opposite ends of the long axis of the ellipse is 14.7 in terms of Euclidean distance, but only 6 with respect to Mahalanobis distance. This is because the Mahalanobis distance gives less emphasis to the direction of largest variance. In practice, computing the Mahalanobis distance is expensive, but can be worthwhile for data whose attributes are correlated. If the attributes are relatively uncorrelated, but have different ranges, then standardizing the variables is sufficient.

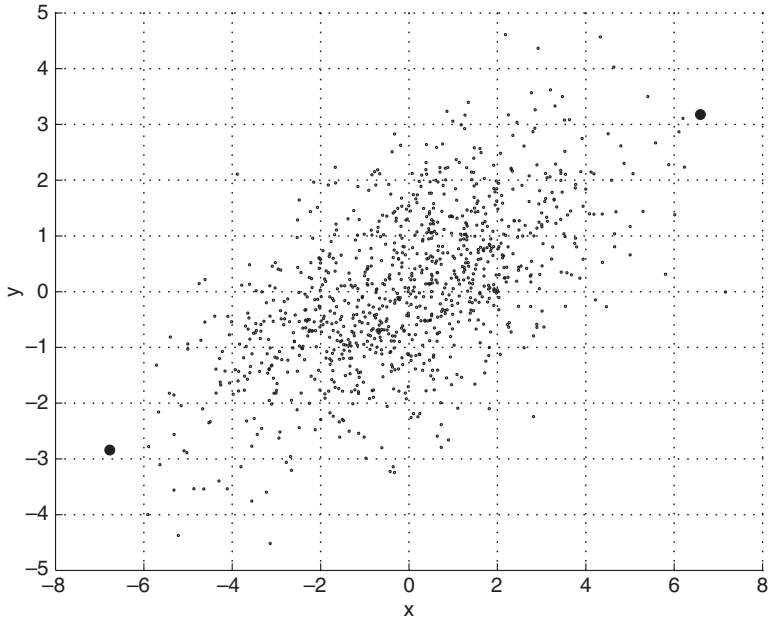


Figure 2.21. Set of two-dimensional points. The Mahalanobis distance between the two points represented by large dots is 6; their Euclidean distance is 14.7.

■

Combining Similarities for Heterogeneous Attributes

The previous definitions of similarity were based on approaches that assumed all the attributes were of the same type. A general approach is needed when the attributes are of different types. One straightforward approach is to compute the similarity between each attribute separately using Table 2.7, and then combine these similarities using a method that results in a similarity between 0 and 1. One possible approach is to define the overall similarity as the average of all the individual attribute similarities. Unfortunately, this approach does not work well if some of the attributes are asymmetric attributes. For example, if all the attributes are asymmetric binary attributes, then the similarity measure suggested previously reduces to the simple matching coefficient, a measure that is not appropriate for asymmetric binary attributes. The easiest way to fix this problem is to omit asymmetric attributes from the similarity calculation when their values are 0 for both of the objects whose similarity

is being computed. A similar approach also works well for handling missing values.

In summary, Algorithm 2.2 is effective for computing an overall similarity between two objects, $\mathbf{x}$ and $\mathbf{y}$, with different types of attributes. This procedure can be easily modified to work with dissimilarities.

Algorithm 2.2 Similarities of heterogeneous objects.

1: For the k^{th} attribute, compute a similarity, $s_k(\mathbf{x}, \mathbf{y})$, in the range $[0, 1]$.
2: Define an indicator variable, δ_k, for the k^{th} attribute as follows:

$$\delta_k = \begin{cases} 0 & \text{if the } k^{th} \text{ attribute is an asymmetric attribute and} \\ & \text{both objects have a value of 0, or if one of the objects} \\ & \text{has a missing value for the } k^{th} \text{ attribute} \\ 1 & \text{otherwise} \end{cases}$$

3: Compute the overall similarity between the two objects using the following formula:

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^{n} \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^{n} \delta_k} \tag{2.28}$$

Using Weights

In much of the previous discussion, all attributes were treated equally when computing proximity. This is not desirable when some attributes are more important to the definition of proximity than others. To address these situations, the formulas for proximity can be modified by weighting the contribution of each attribute.

With attribute weights, w_k, (2.28) becomes

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^{n} w_k \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^{n} w_k \delta_k}. \tag{2.29}$$

The definition of the Minkowski distance can also be modified as follows:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{n} w_k |x_k - y_k|^r \right)^{1/r}. \tag{2.30}$$

2.4.10 Selecting the Right Proximity Measure

A few general observations may be helpful. First, the type of proximity measure should fit the type of data. For many types of dense, continuous data,

metric distance measures such as Euclidean distance are often used. Proximity between continuous attributes is most often expressed in terms of differences, and distance measures provide a well-defined way of combining these differences into an overall proximity measure. Although attributes can have different scales and be of differing importance, these issues can often be dealt with as described earlier, such as normalization and weighting of attributes.

For sparse data, which often consists of asymmetric attributes, we typically employ similarity measures that ignore 0–0 matches. Conceptually, this reflects the fact that, for a pair of complex objects, similarity depends on the number of characteristics they both share, rather than the number of characteristics they both lack. The cosine, Jaccard, and extended Jaccard measures are appropriate for such data.

There are other characteristics of data vectors that often need to be considered. Invariance to scaling (multiplication) and to translation (addition) were previously discussed with respect to Euclidean distance and the cosine and correlation measures. The practical implications of such considerations are that, for example, cosine is more suitable for sparse document data where only scaling is important, while correlation works better for time series, where both scaling and translation are important. Euclidean distance or other types of Minkowski distance are most appropriate when two data vectors are to match as closely as possible across all components (features).

In some cases, transformation or normalization of the data is needed to obtain a proper similarity measure. For instance, time series can have trends or periodic patterns that significantly impact similarity. Also, a proper computation of similarity often requires that time lags be taken into account. Finally, two time series may be similar only over specific periods of time. For example, there is a strong relationship between temperature and the use of natural gas, but only during the heating season.

Practical consideration can also be important. Sometimes, one or more proximity measures are already in use in a particular field, and thus, others will have answered the question of which proximity measures should be used. Other times, the software package or clustering algorithm being used can drastically limit the choices. If efficiency is a concern, then we may want to choose a proximity measure that has a property, such as the triangle inequality, that can be used to reduce the number of proximity calculations. (See Exercise 30.)

However, if common practice or practical restrictions do not dictate a choice, then the proper choice of a proximity measure can be a time-consuming task that requires careful consideration of both domain knowledge and the purpose for which the measure is being used. A number of different similarity

measures may need to be evaluated to see which ones produce results that make the most sense.

2.5 Bibliographic Notes

It is essential to understand the nature of the data that is being analyzed, and at a fundamental level, this is the subject of measurement theory. In particular, one of the initial motivations for defining types of attributes was to be precise about which statistical operations were valid for what sorts of data. We have presented the view of measurement theory that was initially described in a classic paper by S. S. Stevens [112]. (Tables 2.2 and 2.3 are derived from those presented by Stevens [113].) While this is the most common view and is reasonably easy to understand and apply, there is, of course, much more to measurement theory. An authoritative discussion can be found in a three-volume series on the foundations of measurement theory [88, 94, 114]. Also of interest is a wide-ranging article by Hand [77], which discusses measurement theory and statistics, and is accompanied by comments from other researchers in the field. Numerous critiques and extensions of the approach of Stevens have been made [66, 97, 117]. Finally, many books and articles describe measurement issues for particular areas of science and engineering.

Data quality is a broad subject that spans every discipline that uses data. Discussions of precision, bias, accuracy, and significant figures can be found in many introductory science, engineering, and statistics textbooks. The view of data quality as "fitness for use" is explained in more detail in the book by Redman [103]. Those interested in data quality may also be interested in MIT's Information Quality (MITIQ) Program [95, 118]. However, the knowledge needed to deal with specific data quality issues in a particular domain is often best obtained by investigating the data quality practices of researchers in that field.

Aggregation is a less well-defined subject than many other preprocessing tasks. However, aggregation is one of the main techniques used by the database area of Online Analytical Processing (OLAP) [68, 76, 102]. There has also been relevant work in the area of symbolic data analysis (Bock and Diday [64]). One of the goals in this area is to summarize traditional record data in terms of symbolic data objects whose attributes are more complex than traditional attributes. Specifically, these attributes can have values that are sets of values (categories), intervals, or sets of values with weights (histograms). Another goal of symbolic data analysis is to be able to perform clustering, classification, and other kinds of data analysis on data that consists of symbolic data objects.

Sampling is a subject that has been well studied in statistics and related fields. Many introductory statistics books, such as the one by Lindgren [90], have some discussion about sampling, and entire books are devoted to the subject, such as the classic text by Cochran [67]. A survey of sampling for data mining is provided by Gu and Liu [74], while a survey of sampling for databases is provided by Olken and Rotem [98]. There are a number of other data mining and database-related sampling references that may be of interest, including papers by Palmer and Faloutsos [100], Provost et al. [101], Toivonen [115], and Zaki et al. [119].

In statistics, the traditional techniques that have been used for dimensionality reduction are multidimensional scaling (MDS) (Borg and Groenen [65], Kruskal and Uslaner [89]) and principal component analysis (PCA) (Jolliffe [80]), which is similar to singular value decomposition (SVD) (Demmel [70]). Dimensionality reduction is discussed in more detail in Appendix B.

Discretization is a topic that has been extensively investigated in data mining. Some classification algorithms work only with categorical data, and association analysis requires binary data, and thus, there is a significant motivation to investigate how to best binarize or discretize continuous attributes. For association analysis, we refer the reader to work by Srikant and Agrawal [111], while some useful references for discretization in the area of classification include work by Dougherty et al. [71], Elomaa and Rousu [72], Fayyad and Irani [73], and Hussain et al. [78].

Feature selection is another topic well investigated in data mining. A broad coverage of this topic is provided in a survey by Molina et al. [96] and two books by Liu and Motada [91, 92]. Other useful papers include those by Blum and Langley [63], Kohavi and John [87], and Liu et al. [93].

It is difficult to provide references for the subject of feature transformations because practices vary from one discipline to another. Many statistics books have a discussion of transformations, but typically the discussion is restricted to a particular purpose, such as ensuring the normality of a variable or making sure that variables have equal variance. We offer two references: Osborne [99] and Tukey [116].

While we have covered some of the most commonly used distance and similarity measures, there are hundreds of such measures and more are being created all the time. As with so many other topics in this chapter, many of these measures are specific to particular fields, e.g., in the area of time series see papers by Kalpakis et al. [81] and Keogh and Pazzani [83]. Clustering books provide the best general discussions. In particular, see the books by Anderberg [62], Jain and Dubes [79], Kaufman and Rousseeuw [82], and Sneath and Sokal [109].

Information-based measures of similarity have become more popular lately despite the computational difficulties and expense of calculating them. A good introduction to information theory is provided by Cover and Thomas [69]. Computing the mutual information for continuous variables can be straightforward if they follow a well-know distribution, such as Gaussian. However, this is often not the case, and many techniques have been developed. As one example, the article by Khan, et al. [85] compares various methods in the context of comparing short time series. See also the information and mutual information packages for R and Matlab. Mutual information has been the subject of considerable recent attention due to paper by Reshef, et al. [104, 105] that introduced an alternative measure, albeit one based on mutual information, which was claimed to have superior properties. Although this approach had some early support, e.g., [110], others have pointed out various limitations [75, 86, 108].

Two popular books on the topic of kernel methods are [106] and [107]. The latter also has a website with links to kernel-related materials [84]. In addition, many current data mining, machine learning, and statistical learning textbooks have some material about kernel methods. Further references for kernel methods in the context of support vector machine classifiers are provided in the Bibliographic Notes of Section 6.9.4.

Bibliography

[62] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, December 1973.

[63] A. Blum and P. Langley. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, 97(1–2):245–271, 1997.

[64] H. H. Bock and E. Diday. *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data (Studies in Classification, Data Analysis, and Knowledge Organization)*. Springer-Verlag Telos, January 2000.

[65] I. Borg and P. Groenen. *Modern Multidimensional Scaling—Theory and Applications*. Springer-Verlag, February 1997.

[66] N. R. Chrisman. Rethinking levels of measurement for cartography. *Cartography and Geographic Information Systems*, 25(4):231–242, 1998.

[67] W. G. Cochran. *Sampling Techniques*. John Wiley & Sons, 3rd edition, July 1977.

[68] E. F. Codd, S. B. Codd, and C. T. Smalley. Providing OLAP (On-line Analytical Processing) to User- Analysts: An IT Mandate. White Paper, E.F. Codd and Associates, 1993.

[69] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[70] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial & Applied Mathematics, September 1997.

[71] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 194–202, 1995.

[72] T. Elomaa and J. Rousu. General and Efficient Multisplitting of Numerical Attributes. *Machine Learning*, 36(3):201–244, 1999.

[73] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuousvalued attributes for classification learning. In *Proc. 13th Int. Joint Conf. on Artificial Intelligence*, pages 1022–1027. Morgan Kaufman, 1993.

[74] F. H. Gaohua Gu and H. Liu. Sampling and Its Application in Data Mining: A Survey. Technical Report TRA6/00, National University of Singapore, Singapore, 2000.

[75] M. Gorfine, R. Heller, and Y. Heller. Comment on Detecting novel associations in large data sets. *Unpublished (available at http://emotion. technion. ac. il/ gorfinm/files/science6. pdf on 11 Nov. 2012)*, 2012.

[76] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Journal Data Mining and Knowledge Discovery*, 1(1): 29–53, 1997.

[77] D. J. Hand. Statistics and the Theory of Measurement. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 159(3):445–492, 1996.

[78] F. Hussain, H. Liu, C. L. Tan, and M. Dash. TRC6/99: Discretization: an enabling technique. Technical report, National University of Singapore, Singapore, 1999.

[79] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, March 1988.

[80] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2nd edition, October 2002.

[81] K. Kalpakis, D. Gada, and V. Puttagunta. Distance Measures for Effective Clustering of ARIMA Time-Series. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 273–280. IEEE Computer Society, 2001.

[82] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley and Sons, New York, November 1990.

[83] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *KDD*, pages 285–289, 2000.

[84] Kernel Methods for Pattern Analysis Website. http://www.kernel-methods.net/, 2014.

[85] S. Khan, S. Bandyopadhyay, A. R. Ganguly, S. Saigal, D. J. Erickson III, V. Protopopescu, and G. Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical Review E*, 76(2):026209, 2007.

[86] J. B. Kinney and G. S. Atwal. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences*, 2014.

[87] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.

[88] D. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurements: Volume 1: Additive and polynomial representations*. Academic Press, New York, 1971.

[89] J. B. Kruskal and E. M. Uslaner. *Multidimensional Scaling*. Sage Publications, August 1978.

[90] B. W. Lindgren. *Statistical Theory*. CRC Press, January 1993.

[91] H. Liu and H. Motoda, editors. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer International Series in Engineering and Computer Science, 453. Kluwer Academic Publishers, July 1998.

[92] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer International Series in Engineering and Computer Science, 454. Kluwer Academic Publishers, July 1998.

[93] H. Liu, H. Motoda, and L. Yu. Feature Extraction, Selection, and Construction. In N. Ye, editor, *The Handbook of Data Mining*, pages 22–41. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 2003.

[94] R. D. Luce, D. Krantz, P. Suppes, and A. Tversky. *Foundations of Measurements: Volume 3: Representation, Axiomatization, and Invariance*. Academic Press, New York, 1990.

[95] MIT Information Quality (MITIQ) Program. http://mitiq.mit.edu/, 2014.

[96] L. C. Molina, L. Belanche, and A. Nebot. Feature Selection Algorithms: A Survey and Experimental Evaluation. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, 2002.

[97] F. Mosteller and J. W. Tukey. *Data analysis and regression: a second course in statistics*. Addison-Wesley, 1977.

[98] F. Olken and D. Rotem. Random Sampling from Databases—A Survey. *Statistics & Computing*, 5(1):25–42, March 1995.

[99] J. Osborne. Notes on the Use of Data Transformations. *Practical Assessment, Research & Evaluation*, 28(6), 2002.

[100] C. R. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. *ACM SIGMOD Record*, 29(2):82–92, 2000.

[101] F. J. Provost, D. Jensen, and T. Oates. Efficient Progressive Sampling. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 23–32, 1999.

[102] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, August 2002.

[103] T. C. Redman. *Data Quality: The Field Guide*. Digital Press, January 2001.

[104] D. Reshef, Y. Reshef, M. Mitzenmacher, and P. Sabeti. Equitability analysis of the maximal information coefficient, with comparisons. *arXiv preprint arXiv:1301.6314*, 2013.

[105] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti. Detecting novel associations in large data sets. *science*, 334(6062):1518–1524, 2011.

[106] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[107] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[108] N. Simon and R. Tibshirani. Comment on" Detecting Novel Associations In Large Data Sets" by Reshef Et Al, Science Dec 16, 2011. *arXiv preprint arXiv:1401.7645*, 2014.

[109] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1971.

[110] T. Speed. A correlation for the 21st century. *Science*, 334(6062):1502–1503, 2011.

[111] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Montreal, Quebec, Canada, August 1996.

[112] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, June 1946.

[113] S. S. Stevens. Measurement. In G. M. Maranell, editor, *Scaling: A Sourcebook for Behavioral Scientists*, pages 22–41. Aldine Publishing Co., Chicago, 1974.

[114] P. Suppes, D. Krantz, R. D. Luce, and A. Tversky. *Foundations of Measurements: Volume 2: Geometrical, Threshold, and Probabilistic Representations*. Academic Press, New York, 1989.

[115] H. Toivonen. Sampling Large Databases for Association Rules. In *VLDB96*, pages 134–145. Morgan Kaufman, September 1996.

[116] J. W. Tukey. On the Comparative Anatomy of Transformations. *Annals of Mathematical Statistics*, 28(3):602–632, September 1957.

[117] P. F. Velleman and L. Wilkinson. Nominal, ordinal, interval, and ratio typologies are misleading. *The American Statistician*, 47(1):65–72, 1993.

[118] R. Y. Wang, M. Ziad, Y. W. Lee, and Y. R. Wang. *Data Quality*. The Kluwer International Series on Advances in Database Systems, Volume 23. Kluwer Academic Publishers, January 2001.

[119] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of Sampling for Data Mining of Association Rules. Technical Report TR617, Rensselaer Polytechnic Institute, 1996.

2.6 Exercises

1. In the initial example of Chapter 2, the statistician says, "Yes, fields 2 and 3 are basically the same." Can you tell from the three lines of sample data that are shown why she says that?

2. Classify the following attributes as binary, discrete, or continuous. Also classify them as qualitative (nominal or ordinal) or quantitative (interval or ratio). Some cases may have more than one interpretation, so briefly indicate your reasoning if you think there may be some ambiguity.

   Example: Age in years. **Answer:** Discrete, quantitative, ratio

   (a) Time in terms of AM or PM.

   (b) Brightness as measured by a light meter.

   (c) Brightness as measured by people's judgments.

   (d) Angles as measured in degrees between 0 and 360.

   (e) Bronze, Silver, and Gold medals as awarded at the Olympics.

   (f) Height above sea level.

   (g) Number of patients in a hospital.

   (h) ISBN numbers for books. (Look up the format on the Web.)

   (i) Ability to pass light in terms of the following values: opaque, translucent, transparent.

   (j) Military rank.

   (k) Distance from the center of campus.

(l) Density of a substance in grams per cubic centimeter.

(m) Coat check number. (When you attend an event, you can often give your coat to someone who, in turn, gives you a number that you can use to claim your coat when you leave.)

3. What is aggregation? What are the motivations for aggregation? How are the values of attributes handled when aggregating data?

4. What is sampling? In what situations should a simple random sample be used? When should stratified sampling be used? Why is the size of the sample important? How does it affect results?

5. What is the difference between feature extraction and feature creation?

6. You are approached by the marketing director of a local company, who believes that he has devised a foolproof way to measure customer satisfaction. He explains his scheme as follows: "It's so simple that I can't believe that no one has thought of it before. I just keep track of the number of customer complaints for each product. I read in a data mining book that counts are ratio attributes, and so, my measure of product satisfaction must be a ratio attribute. But when I rated the products based on my new customer satisfaction measure and showed them to my boss, he told me that I had overlooked the obvious, and that my measure was worthless. I think that he was just mad because our best-selling product had the worst satisfaction since it had the most complaints. Could you help me set him straight?"

   (a) Who is right, the marketing director or his boss? If you answered, his boss, what would you do to fix the measure of satisfaction?

   (b) What can you say about the attribute type of the original product satisfaction attribute?

7. A few months later, you are again approached by the same marketing director as in Exercise 6. This time, he has devised a better approach to measure the extent to which a customer prefers one product over other similar products. He explains, "When we develop new products, we typically create several variations and evaluate which one customers prefer. Our standard procedure is to give our test subjects all of the product variations at one time and then ask them to rank the product variations in order of preference. However, our test subjects are very indecisive, especially when there are more than two products. As a result, testing takes forever. I suggested that we perform the comparisons in pairs and then use these comparisons to get the rankings. Thus, if we have three product variations, we have the customers compare variations 1 and 2, then 2 and 3, and finally 3 and 1. Our testing time with my new procedure is a third of what it was for the old procedure, but the employees conducting the tests complain that they cannot come up with a consistent ranking from

the results. And my boss wants the latest product evaluations, yesterday. I should also mention that he was the person who came up with the old product evaluation approach. Can you help me?"

(a) Is the marketing director in trouble? Will his approach work for generating an ordinal ranking of the product variations in terms of customer preference? Explain.

(b) Is there a way to fix the marketing director's approach? More generally, what can you say about trying to create an ordinal measurement scale based on pairwise comparisons?

(c) For the original product evaluation scheme, the overall rankings of each product variation are found by computing its average over all test subjects. Comment on whether you think that this is a reasonable approach. What other approaches might you take?

8. Can you think of a situation in which identification numbers would be useful for prediction?

9. When discretizing continuous attributes, when would you choose unsupervised discretization and when would you choose supervised discretization? Why?

10. An educational psychologist wants to use association analysis to analyze test results. The test consists of 100 questions with four possible answers each.

    (a) How would you convert this data into a form suitable for association analysis?

    (b) In particular, what type of attributes would you have and how many of them are there?

11. Which of the following quantities is likely to show more temporal autocorrelation: daily rainfall or daily temperature? Why?

12. Discuss why a document-term matrix is an example of a data set that has asymmetric discrete or asymmetric continuous features.

13. Many sciences rely on observation instead of (or in addition to) designed experiments. Compare the data quality issues involved in observational science with those of experimental science and data mining.

14. Discuss the difference between the precision of a measurement and the terms single and double precision, as they are used in computer science, typically to represent floating-point numbers that require 32 and 64 bits, respectively.

15. Give at least two advantages to working with data stored in text files instead of in a binary format.

16. Distinguish between noise and outliers. Be sure to consider the following questions.

    (a) Is noise ever interesting or desirable? Outliers?

    (b) Can noise objects be outliers?

    (c) Are noise objects always outliers?

    (d) Are outliers always noise objects?

    (e) Can noise make a typical value into an unusual one, or vice versa?

Algorithm 2.3 Algorithm for finding k-nearest neighbors.

1: **for** $i = 1$ to *number of data objects* **do**
2: Find the distances of the i^{th} object to all other objects.
3: Sort these distances in decreasing order.
(Keep track of which object is associated with each distance.)
4: **return** the objects associated with the first K distances of the sorted list
5: **end for**

17. Consider the problem of finding the K-nearest neighbors of a data object. A programmer designs Algorithm 2.3 for this task.

    (a) Describe the potential problems with this algorithm if there are duplicate objects in the data set. Assume the distance function will return a distance of 0 only for objects that are the same.

    (b) How would you fix this problem?

18. The following attributes are measured for members of a herd of Asian elephants: *weight, height, tusk length, trunk length,* and *ear area*. Based on these measurements, what sort of proximity measure from Section 2.4 would you use to compare or group these elephants? Justify your answer and explain any special circumstances.

19. You are given a set of m objects that is divided into K groups, where the i^{th} group is of size m_i. If the goal is to obtain a sample of size $n < m$, what is the difference between the following two sampling schemes? (Assume sampling with replacement.)

    (a) We randomly select $n \times m_i/m$ elements from each group.

    (b) We randomly select n elements from the data set, without regard for the group to which an object belongs.

20. Consider a document-term matrix, where tf_{ij} is the frequency of the i^{th} word (term) in the j^{th} document and m is the number of documents. Consider the variable transformation that is defined by

$$tf'_{ij} = tf_{ij} \times \log \frac{m}{df_i}, \tag{2.31}$$

where df_i is the number of documents in which the i^{th} term appears, which is known as the **document frequency** of the term. This transformation is known as the **inverse document frequency** transformation.

(a) What is the effect of this transformation if a term occurs in one document? In every document?

(b) What might be the purpose of this transformation?

21. Assume that we apply a square root transformation to a ratio attribute x to obtain the new attribute x^*. As part of your analysis, you identify an interval (a, b) in which x^* has a linear relationship to another attribute y.

(a) What is the corresponding interval (a, b) in terms of x?

(b) Give an equation that relates y to x.

22. For two points $P_1 = (2, 0)$ and $P_2 = (3, 1)$, compute the following.

(a) Hamming distance

(b) Euclidean distance

(c) Supremum distance

23. This exercise compares and contrasts some similarity and distance measures.

(a) For binary data, the L1 distance corresponds to the Hamming distance; that is, the number of bits that are different between two binary vectors. The Jaccard similarity is a measure of the similarity between two binary vectors. Compute the Hamming distance and the Jaccard similarity between the following two binary vectors.

$\mathbf{x} = 0101010001$
$\mathbf{y} = 0100011000$

(b) Which approach, Jaccard or Hamming distance, is more similar to the Simple Matching Coefficient, and which approach is more similar to the cosine measure? Explain. (Note: The Hamming measure is a distance, while the other three measures are similarities, but don't let this confuse you.)

(c) Suppose that you are comparing how similar two organisms of different species are in terms of the number of genes they share. Describe which measure, Hamming or Jaccard, you think would be more appropriate for comparing the genetic makeup of two organisms. Explain. (Assume that each animal is represented as a binary vector, where each attribute is 1 if a particular gene is present in the organism and 0 otherwise.)

(d) If you wanted to compare the genetic makeup of two organisms of the same species, e.g., two human beings, would you use the Hamming distance, the Jaccard coefficient, or a different measure of similarity or distance? Explain. (Note that two human beings share $> 99.9\%$ of the same genes.)

24. For the following vectors, $\mathbf{x}$ and $\mathbf{y}$, calculate the indicated similarity or distance measures.

(a) $\mathbf{x} = (1, 1, 1, 1),\ \mathbf{y} = (2, 2, 2, 2)$ cosine, correlation, Euclidean

(b) $\mathbf{x} = (0, 1, 0, 1),\ \mathbf{y} = (1, 0, 1, 0)$ cosine, correlation, Euclidean, Jaccard

(c) $\mathbf{x} = (0, -1, 0, 1),\ \mathbf{y} = (1, 0, -1, 0)$ cosine, correlation, Euclidean

(d) $\mathbf{x} = (1, 1, 0, 1, 0, 1),\ \mathbf{y} = (1, 1, 1, 0, 0, 1)$ cosine, correlation, Jaccard

(e) $\mathbf{x} = (2, -1, 0, 2, 0, -3),\ \mathbf{y} = (-1, 1, -1, 0, 0, -1)$ cosine, correlation

25. Here, we further explore the cosine and correlation measures.

(a) What is the range of values possible for the cosine measure?

(b) If two objects have a cosine measure of 1, are they identical? Explain.

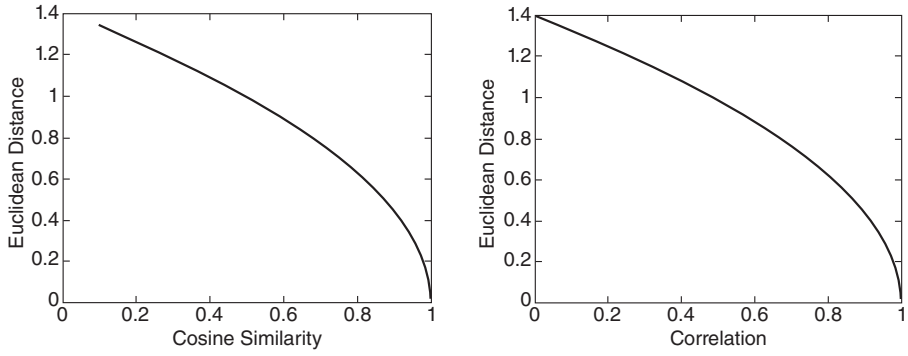
(c) What is the relationship of the cosine measure to correlation, if any? (Hint: Look at statistical measures such as mean and standard deviation in cases where cosine and correlation are the same and different.)

(d) Figure 2.22(a) shows the relationship of the cosine measure to Euclidean distance for 100,000 randomly generated points that have been normalized to have an L2 length of 1. What general observation can you make about the relationship between Euclidean distance and cosine similarity when vectors have an L2 norm of 1?

(e) Figure 2.22(b) shows the relationship of correlation to Euclidean distance for 100,000 randomly generated points that have been standardized to have a mean of 0 and a standard deviation of 1. What general observation can you make about the relationship between Euclidean distance and correlation when the vectors have been standardized to have a mean of 0 and a standard deviation of 1?

(f) Derive the mathematical relationship between cosine similarity and Euclidean distance when each data object has an L_2 length of 1.

(g) Derive the mathematical relationship between correlation and Euclidean distance when each data point has been standardized by subtracting its mean and dividing by its standard deviation.



(a) Relationship between Euclidean distance and the cosine measure.

(b) Relationship between Euclidean distance and correlation.

Figure 2.22. Graphs for Exercise 25.

26. Show that the set difference metric given by

$$d(A, B) = size(A - B) + size(B - A) \tag{2.32}$$

satisfies the metric axioms given on page 97. A and B are sets and $A - B$ is the set difference.

27. Discuss how you might map correlation values from the interval [−1,1] to the interval [0,1]. Note that the type of transformation that you use might depend on the application that you have in mind. Thus, consider two applications: clustering time series and predicting the behavior of one time series given another.

28. Given a similarity measure with values in the interval [0,1], describe two ways to transform this similarity value into a dissimilarity value in the interval [0,∞].

29. Proximity is typically defined between a pair of objects.

    (a) Define two ways in which you might define the proximity among a group of objects.

    (b) How might you define the distance between two sets of points in Euclidean space?

    (c) How might you define the proximity between two sets of data objects? (Make no assumption about the data objects, except that a proximity measure is defined between any pair of objects.)

30. You are given a set of points S in Euclidean space, as well as the distance of each point in S to a point $\mathbf{x}$. (It does not matter if $\mathbf{x} \in S$.)

(a) If the goal is to find all points within a specified distance ε of point $\mathbf{y}$, $\mathbf{y} \neq \mathbf{x}$, explain how you could use the triangle inequality and the already calculated distances to $\mathbf{x}$ to potentially reduce the number of distance calculations necessary? Hint: The triangle inequality, $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{x})$, can be rewritten as $d(\mathbf{x}, \mathbf{y}) \geq d(\mathbf{x}, \mathbf{z}) - d(\mathbf{y}, \mathbf{z})$.

(b) In general, how would the distance between $\mathbf{x}$ and $\mathbf{y}$ affect the number of distance calculations?

(c) Suppose that you can find a small subset of points S', from the original data set, such that every point in the data set is within a specified distance ε of at least one of the points in S', and that you also have the pairwise distance matrix for S'. Describe a technique that uses this information to compute, with a minimum of distance calculations, the set of all points within a distance of β of a specified point from the data set.

31. Show that 1 minus the Jaccard similarity is a distance measure between two data objects, $\mathbf{x}$ and $\mathbf{y}$, that satisfies the metric axioms given on page 97. Specifically, $d(\mathbf{x}, \mathbf{y}) = 1 - \mathrm{J}(\mathbf{x}, \mathbf{y})$.

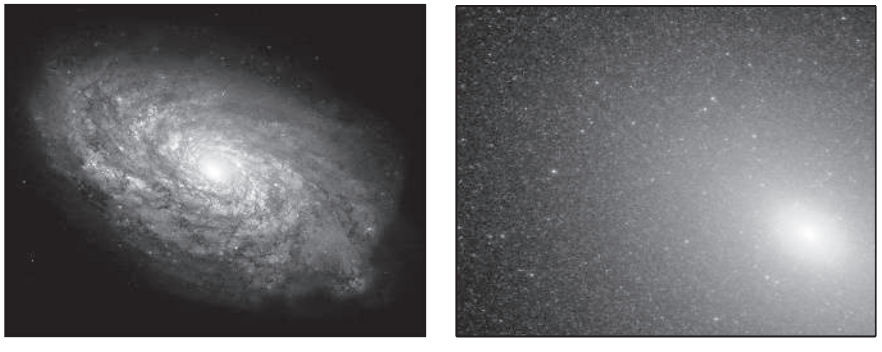
32. Show that the distance measure defined as the angle between two data vectors, $\mathbf{x}$ and $\mathbf{y}$, satisfies the metric axioms given on page 97. Specifically, $d(\mathbf{x}, \mathbf{y}) = \arccos(\cos(\mathbf{x}, \mathbf{y}))$.

33. Explain why computing the proximity between two attributes is often simpler than computing the similarity between two objects.

3

Classification: Basic Concepts and Techniques

Humans have an innate ability to classify things into categories, e.g., mundane tasks such as filtering spam email messages or more specialized tasks such as recognizing celestial objects in telescope images (see Figure 3.1). While manual classification often suffices for small and simple data sets with only a few attributes, larger and more complex data sets require an automated solution.



(a) A spiral galaxy. (b) An elliptical galaxy.

Figure 3.1. Classification of galaxies from telescope images taken from the NASA website.

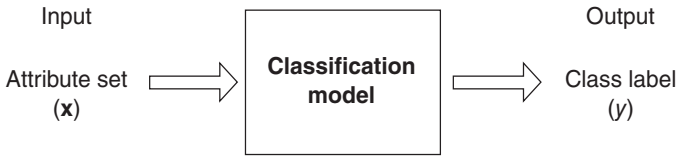


Figure 3.2. A schematic illustration of a classification task.

This chapter introduces the basic concepts of classification and describes some of its key issues such as model overfitting, model selection, and model evaluation. While these topics are illustrated using a classification technique known as decision tree induction, most of the discussion in this chapter is also applicable to other classification techniques, many of which are covered in Chapter 6.

3.1 Basic Concepts

Figure 3.2 illustrates the general idea behind classification. The data for a classification task consists of a collection of instances (records). Each such instance is characterized by the tuple $(\mathbf{x}, y)$, where $\mathbf{x}$ is the set of attribute values that describe the instance and y is the class label of the instance. The attribute set $\mathbf{x}$ can contain attributes of any type, while the class label y must be categorical.

A **classification model** is an abstract representation of the relationship between the attribute set and the class label. As will be seen in the next two chapters, the model can be represented in many ways, e.g., as a tree, a probability table, or simply, a vector of real-valued parameters. More formally, we can express it mathematically as a target function f that takes as input the attribute set $\mathbf{x}$ and produces an output corresponding to the predicted class label. The model is said to classify an instance $(\mathbf{x}, y)$ correctly if $f(\mathbf{x}) = y$.

Table 3.1. Examples of classification tasks.

Task	Attribute set	Class label
Spam filtering	Features extracted from email message header and content	spam or non-spam
Tumor identification	Features extracted from magnetic resonance imaging (MRI) scans	malignant or benign
Galaxy classification	Features extracted from telescope images	elliptical, spiral, or irregular-shaped

Table 3.2. A sample data for the vertebrate classification problem.

Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

Table 3.1 shows examples of attribute sets and class labels for various classification tasks. Spam filtering and tumor identification are examples of binary classification problems, in which each data instance can be categorized into one of two classes. If the number of classes is larger than 2, as in the galaxy classification example, then it is called a multiclass classification problem.

We illustrate the basic concepts of classification in this chapter with the following two examples.

Example 3.1. [Vertebrate Classification] Table 3.2 shows a sample data set for classifying vertebrates into mammals, reptiles, birds, fishes, and amphibians. The attribute set includes characteristics of the vertebrate such as its body temperature, skin cover, and ability to fly. The data set can also be used for a binary classification task such as mammal classification, by grouping the reptiles, birds, fishes, and amphibians into a single category called non-mammals. ■

Example 3.2. [Loan Borrower Classification] Consider the problem of predicting whether a loan borrower will repay the loan or default on the loan payments. The data set used to build the classification model is shown in Table 3.3. The attribute set includes personal information of the borrower such as marital status and annual income, while the class label indicates whether the borrower had defaulted on the loan payments. ■

Table 3.3. A sample data for the loan borrower classification problem.

ID	Home Owner	Marital Status	Annual Income	Defaulted?
1	Yes	Single	125000	No
2	No	Married	100000	No
3	No	Single	70000	No
4	Yes	Married	120000	No
5	No	Divorced	95000	Yes
6	No	Single	60000	No
7	Yes	Divorced	220000	No
8	No	Single	85000	Yes
9	No	Married	75000	No
10	No	Single	90000	Yes

A classification model serves two important roles in data mining. First, it is used as a **predictive model** to classify previously unlabeled instances. A good classification model must provide accurate predictions with a fast response time. Second, it serves as a **descriptive model** to identify the characteristics that distinguish instances from different classes. This is particularly useful for critical applications, such as medical diagnosis, where it is insufficient to have a model that makes a prediction without justifying how it reaches such a decision.

For example, a classification model induced from the vertebrate data set shown in Table 3.2 can be used to predict the class label of the following vertebrate:

Vertebrate Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

In addition, it can be used as a descriptive model to help determine characteristics that define a vertebrate as a mammal, a reptile, a bird, a fish, or an amphibian. For example, the model may identify mammals as warm-blooded vertebrates that give birth to their young.

There are several points worth noting regarding the previous example. First, although all the attributes shown in Table 3.2 are qualitative, there are no restrictions on the type of attributes that can be used as predictor variables. The class label, on the other hand, must be of nominal type. This distinguishes classification from other predictive modeling tasks such as regression, where the predicted value is often quantitative. More information about regression can be found in Appendix D.

Another point worth noting is that not all attributes may be relevant to the classification task. For example, the average length or weight of a

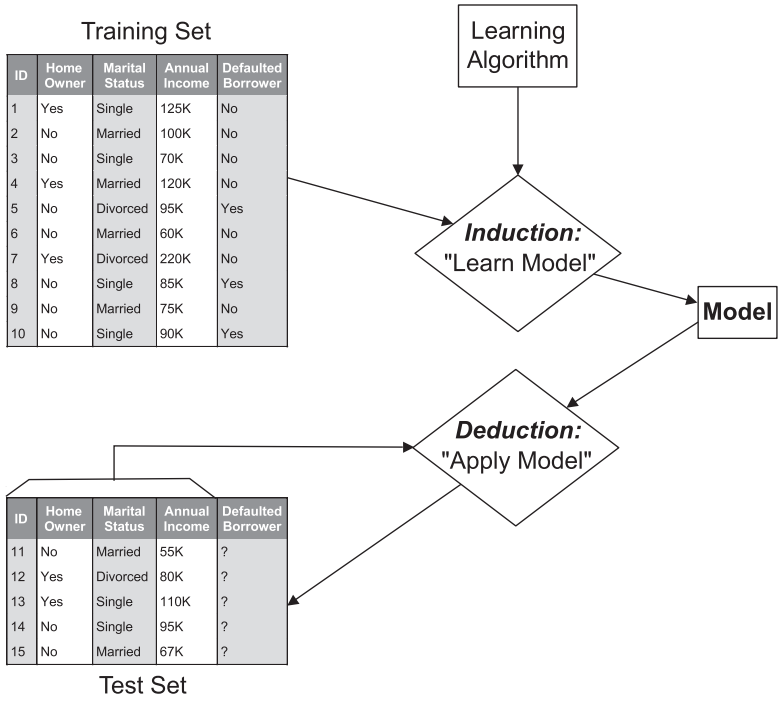
vertebrate may not be useful for classifying mammals, as these attributes can show same value for both mammals and non-mammals. Such an attribute is typically discarded during preprocessing. The remaining attributes might not be able to distinguish the classes by themselves, and thus, must be used in concert with other attributes. For instance, the `Body Temperature` attribute is insufficient to distinguish mammals from other vertebrates. When it is used together with `Gives Birth`, the classification of mammals improves significantly. However, when additional attributes, such as `Skin Cover` are included, the model becomes overly specific and no longer covers all mammals. Finding the optimal combination of attributes that best discriminates instances from different classes is the key challenge in building classification models.

3.2 General Framework for Classification

Classification is the task of assigning labels to unlabeled data instances and a **classifier** is used to perform such a task. A classifier is typically described in terms of a model as illustrated in the previous section. The model is created using a given a set of instances, known as the **training set**, which contains attribute values as well as class labels for each instance. The systematic approach for learning a classification model given a training set is known as a **learning algorithm**. The process of using a learning algorithm to build a classification model from the training data is known as **induction**. This process is also often described as "learning a model" or "building a model." This process of applying a classification model on unseen test instances to predict their class labels is known as **deduction**. Thus, the process of classification involves two steps: applying a learning algorithm to training data to learn a model, and then applying the model to assign labels to unlabeled instances. Figure 3.3 illustrates the general framework for classification.

A **classification technique** refers to a general approach to classification, e.g., the decision tree technique that we will study in this chapter. This classification technique like most others, consists of a family of related models and a number of algorithms for learning these models. In Chapter 6, we will study additional classification techniques, including neural networks and support vector machines.

A couple notes on terminology. First, the terms "classifier" and "model" are often taken to be synonymous. If a classification technique builds a single, global model, then this is fine. However, while every model defines a classifier, not every classifier is defined by a single model. Some classifiers, such as k-nearest neighbor classifiers, do not build an explicit model (Section 6.3), while



ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
11	No	Married	55K	?
12	Yes	Divorced	80K	?
13	Yes	Single	110K	?
14	No	Single	95K	?
15	No	Married	67K	?

Figure 3.3. General framework for building a classification model.

other classifiers, such as ensemble classifiers, combine the output of a collection of models (Section 6.10). Second, the term "classifier" is often used in a more general sense to refer to a classification technique. Thus, for example, "decision tree classifier" can refer to the decision tree classification technique or a specific classifier built using that technique. Fortunately, the meaning of "classifier" is usually clear from the context.

In the general framework shown in Figure 3.3, the induction and deduction steps should be performed separately. In fact, as will be discussed later in Section 3.6, the training and test sets should be independent of each other to ensure that the induced model can accurately predict the class labels of instances it has never encountered before. Models that deliver such predictive insights are said to have good **generalization performance**. The performance of a model (classifier) can be evaluated by comparing the predicted labels against the true labels of instances. This information can be summarized in a table called a **confusion matrix**. Table 3.4 depicts the confusion matrix for a binary classification problem. Each entry f_{ij} denotes the number of instances from class i predicted to be of class j. For example, f_{01} is the

Table 3.4. Confusion matrix for a binary classification problem.

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

number of instances from class 0 incorrectly predicted as class 1. The number of correct predictions made by the model is $(f_{11} + f_{00})$ and the number of incorrect predictions is $(f_{10} + f_{01})$.

Although a confusion matrix provides the information needed to determine how well a classification model performs, summarizing this information into a single number makes it more convenient to compare the relative performance of different models. This can be done using an **evaluation metric** such as **accuracy**, which is computed in the following way:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{3.1}$$

For binary classification problems, the accuracy of a model is given by

$$\text{Accuracy} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}. \tag{3.2}$$

Error rate is another related metric, which is defined as follows for binary classification problems:

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}. \tag{3.3}$$

The learning algorithms of most classification techniques are designed to learn models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set. We will revisit the topic of model evaluation in Section 3.6.

3.3 Decision Tree Classifier

This section introduces a simple classification technique known as the **decision tree** classifier. To illustrate how a decision tree works, consider the classification problem of distinguishing mammals from non-mammals using

the vertebrate data set shown in Table 3.2. Suppose a new species is discovered by scientists. How can we tell whether it is a mammal or a non-mammal? One approach is to pose a series of questions about the characteristics of the species. The first question we may ask is whether the species is cold- or warm-blooded. If it is cold-blooded, then it is definitely not a mammal. Otherwise, it is either a bird or a mammal. In the latter case, we need to ask a follow-up question: Do the females of the species give birth to their young? Those that do give birth are definitely mammals, while those that do not are likely to be non-mammals (with the exception of egg-laying mammals such as the platypus and spiny anteater).

The previous example illustrates how we can solve a classification problem by asking a series of carefully crafted questions about the attributes of the test instance. Each time we receive an answer, we could ask a follow-up question until we can conclusively decide on its class label. The series of questions and their possible answers can be organized into a hierarchical structure called a decision tree. Figure 3.4 shows an example of the decision tree for the mammal classification problem. The tree has three types of nodes:

- A **root node**, with no incoming links and zero or more outgoing links.
- **Internal nodes**, each of which has exactly one incoming link and two or more outgoing links.
- **Leaf** or **terminal** nodes, each of which has exactly one incoming link and no outgoing links.

Every leaf node in the decision tree is associated with a class label. The **non-terminal** nodes, which include the root and internal nodes, contain **attribute test conditions** that are typically defined using a single attribute. Each possible outcome of the attribute test condition is associated with exactly one child of this node. For example, the root node of the tree shown in Figure 3.4 uses the attribute `Body Temperature` to define an attribute test condition that has two outcomes, warm and cold, resulting in two child nodes.

Given a decision tree, classifying a test instance is straightforward. Starting from the root node, we apply its attribute test condition and follow the appropriate branch based on the outcome of the test. This will lead us either to another internal node, for which a new attribute test condition is applied, or to a leaf node. Once a leaf node is reached, we assign the class label associated with the node to the test instance. As an illustration, Figure 3.5 traces the path used to predict the class label of a flamingo. The path terminates at a leaf node labeled as `Non-mammals`.

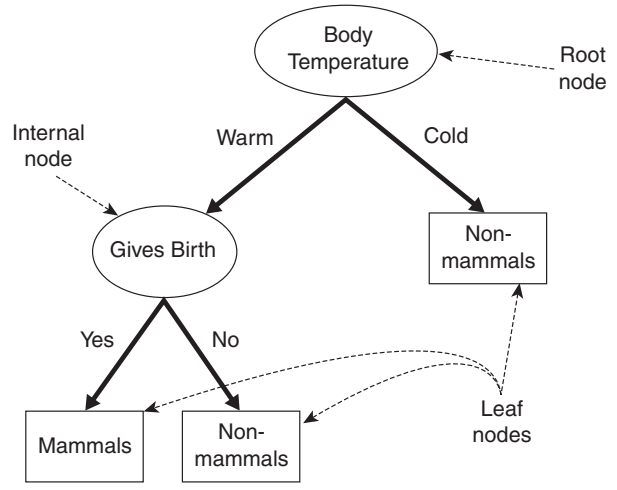


Figure 3.4. A decision tree for the mammal classification problem.

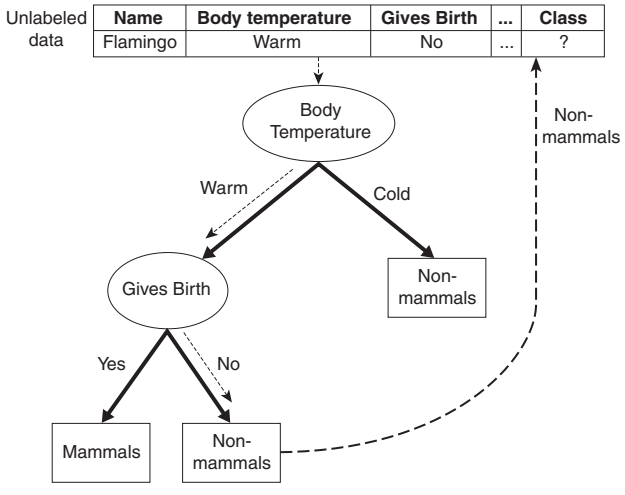


Figure 3.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the `Non-mammals` class.

3.3.1 A Basic Algorithm to Build a Decision Tree

Many possible decision trees that can be constructed from a particular data set. While some trees are better than others, finding an optimal one is computationally expensive due to the exponential size of the search space. Efficient algorithms have been developed to induce a reasonably accurate, albeit

suboptimal, decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy to grow the decision tree in a top-down fashion by making a series of locally optimal decisions about which attribute to use when partitioning the training data. One of the earliest method is **Hunt's algorithm**, which is the basis for many current implementations of decision tree classifiers, including ID3, C4.5, and CART. This subsection presents Hunt's algorithm and describes some of the design issues that must be considered when building a decision tree.

Hunt's Algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion. The tree initially contains a single root node that is associated with all the training instances. If a node is associated with instances from more than one class, it is expanded using an attribute test condition that is determined using a **splitting criterion**. A child leaf node is created for each outcome of the attribute test condition and the instances associated with the parent node are distributed to the children based on the test outcomes. This node expansion step can then be recursively applied to each child node, as long as it has labels of more than one class. If all the instances associated with a leaf node have identical class labels, then the node is not expanded any further. Each leaf node is assigned a class label that occurs most frequently in the training instances associated with the node.

To illustrate how the algorithm works, consider the training set shown in Table 3.3 for the loan borrower classification problem. Suppose we apply Hunt's algorithm to fit the training data. The tree initially contains only a single leaf node as shown in Figure 3.6(a). This node is labeled as `Defaulted = No`, since the majority of the borrowers did not default on their loan payments. The training error of this tree is 30% as three out of the ten training instances have the class label `Defaulted = Yes`. The leaf node can therefore be further expanded because it contains training instances from more than one class.

Let `Home Owner` be the attribute chosen to split the training instances. The justification for choosing this attribute as the attribute test condition will be discussed later. The resulting binary split on the `Home Owner` attribute is shown in Figure 3.6(b). All the training instances for which `Home Owner = Yes` are propagated to the left child of the root node and the rest are propagated to the right child. Hunt's algorithm is then recursively applied to each child. The left child becomes a leaf node labeled `Defaulted = No`, since all instances associated with this node have identical class label `Defaulted = No`. The right child has instances from each class label. Hence, we split it

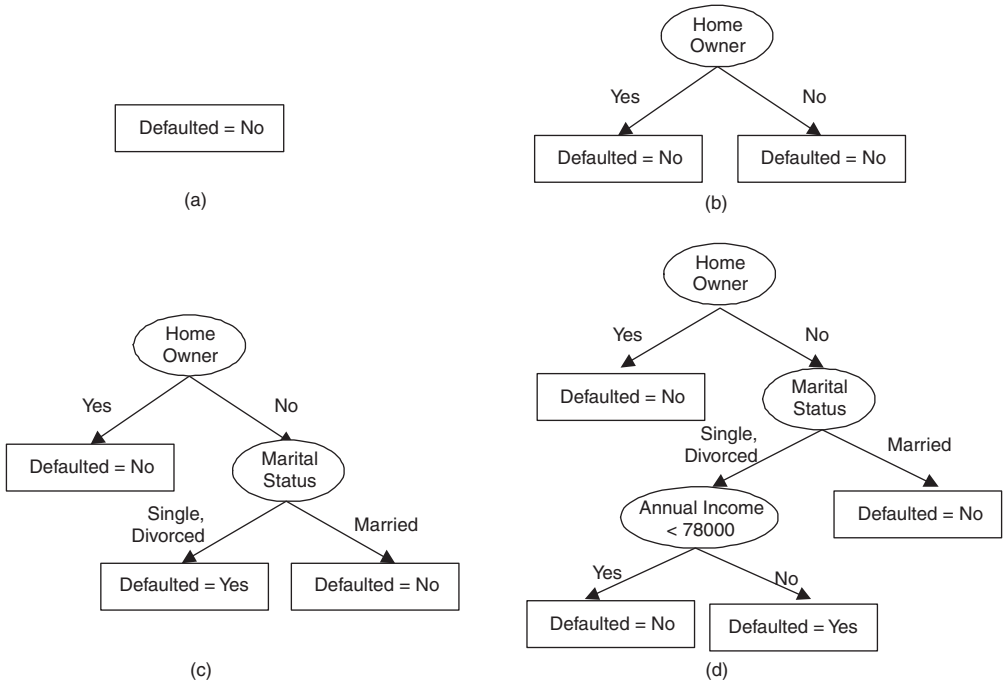


Figure 3.6. Hunt's algorithm for building decision trees.

further. The resulting subtrees after recursively expanding the right child are shown in Figures 3.6(c) and (d).

Hunt's algorithm, as described above, makes some simplifying assumptions that are often not true in practice. In the following, we describe these assumptions and briefly discuss some of the possible ways for handling them.

1. Some of the child nodes created in Hunt's algorithm can be empty if none of the training instances have the particular attribute values. One way to handle this is by declaring each of them as a leaf node with a class label that occurs most frequently among the training instances associated with their parent nodes.

2. If all training instances associated with a node have identical attribute values but different class labels, it is not possible to expand this node any further. One way to handle this case is to declare it a leaf node and assign it the class label that occurs most frequently in the training instances associated with this node.

Design Issues of Decision Tree Induction

Hunt's algorithm is a generic procedure for growing decision trees in a greedy fashion. To implement the algorithm, there are two key design issues that must be addressed.

1. **What is the splitting criterion?** At each recursive step, an attribute must be selected to partition the training instances associated with a node into smaller subsets associated with its child nodes. The splitting criterion determines which attribute is chosen as the test condition and how the training instances should be distributed to the child nodes. This will be discussed in Sections 3.3.2 and 3.3.3.

2. **What is the stopping criterion?** The basic algorithm stops expanding a node only when all the training instances associated with the node have the same class labels or have identical attribute values. Although these conditions are sufficient, there are reasons to stop expanding a node much earlier even if the leaf node contains training instances from more than one class. This process is called early termination and the condition used to determine when a node should be stopped from expanding is called a stopping criterion. The advantages of early termination are discussed in Section 3.4.

3.3.2 Methods for Expressing Attribute Test Conditions

Decision tree induction algorithms must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types.

Binary Attributes The test condition for a binary attribute generates two potential outcomes, as shown in Figure 3.7.

Nominal Attributes Since a nominal attribute can have many values, its attribute test condition can be expressed in two ways, as a multiway split or a binary split as shown in Figure 3.8. For a multiway split (Figure 3.8(a)), the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values—single, married, or divorced—its test condition will produce a three-way split. It is also possible to create a binary split by partitioning all values taken by the nominal attribute into two groups. For example, some decision tree algorithms, such as CART, produce only binary

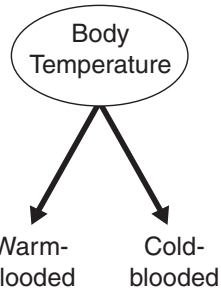


Figure 3.7. Attribute test condition for a binary attribute.

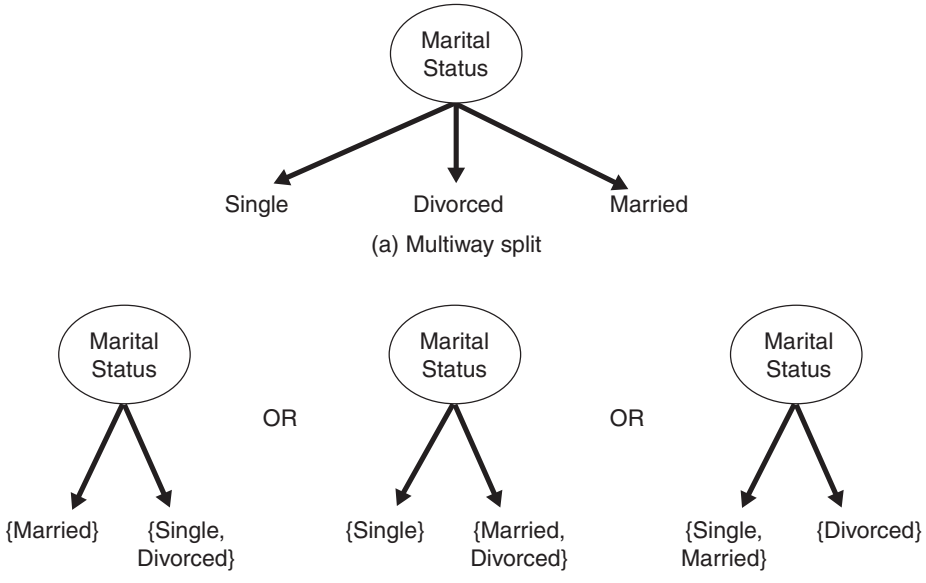


Figure 3.8. Attribute test conditions for nominal attributes.

splits by considering all $2^{k-1} - 1$ ways of creating a binary partition of k attribute values. Figure 3.8(b) illustrates three different ways of grouping the attribute values for marital status into two subsets.

Ordinal Attributes Ordinal attributes can also produce binary or multiway splits. Ordinal attribute values can be grouped as long as the grouping

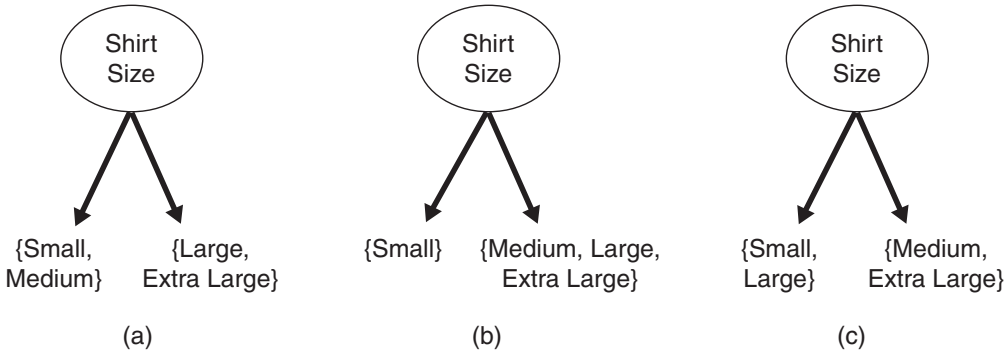


Figure 3.9. Different ways of grouping ordinal attribute values.

does not violate the order property of the attribute values. Figure 3.9 illustrates various ways of splitting training records based on the `Shirt Size` attribute. The groupings shown in Figures 3.9(a) and (b) preserve the order among the attribute values, whereas the grouping shown in Figure 3.9(c) violates this property because it combines the attribute values `Small` and `Large` into the same partition while `Medium` and `Extra Large` are combined into another partition.

Continuous Attributes For continuous attributes, the attribute test condition can be expressed as a comparison test (e.g., $A < v$) producing a binary split, or as a range query of the form $v_i \leq A < v_{i+1}$, for $i = 1, \ldots, k$, producing a multiway split. The difference between these approaches is shown in Figure 3.10. For the binary split, any possible value v between the minimum and maximum attribute values in the training data can be used for constructing the comparison test $A < v$. However, it is sufficient to only consider distinct attribute values in the training set as candidate split positions. For the multiway split, any possible collection of attribute value ranges can be used, as long as they are mutually exclusive and cover the entire range of attribute values between the minimum and maximum values observed in the training set. One approach for constructing multiway splits is to apply the discretization strategies described in Section 2.3.6 on page 83. After discretization, a new ordinal value is assigned to each discretized interval, and the attribute test condition is then defined using this newly constructed ordinal attribute.

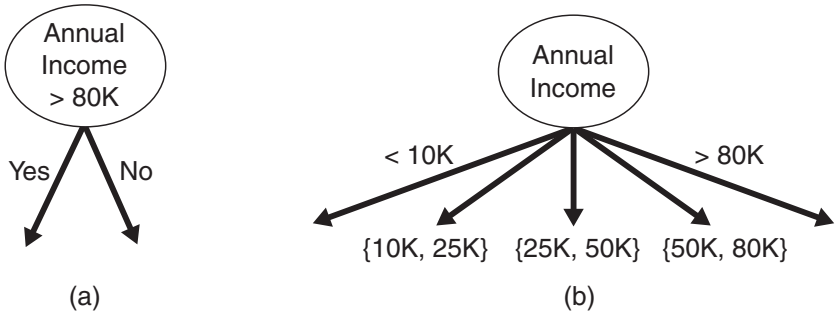


Figure 3.10. Test condition for continuous attributes.

3.3.3 Measures for Selecting an Attribute Test Condition

There are many measures that can be used to determine the goodness of an attribute test condition. These measures try to give preference to attribute test conditions that partition the training instances into *purer* subsets in the child nodes, which mostly have the same class labels. Having purer nodes is useful since a node that has all of its training instances from the same class does not need to be expanded further. In contrast, an impure node containing training instances from multiple classes is likely to require several levels of node expansions, thereby increasing the depth of the tree considerably. Larger trees are less desirable as they are more susceptible to model overfitting, a condition that may degrade the classification performance on unseen instances, as will be discussed in Section 3.4. They are also difficult to interpret and incur more training and test time as compared to smaller trees.

In the following, we present different ways of measuring the impurity of a node and the collective impurity of its child nodes, both of which will be used to identify the best attribute test condition for a node.

Impurity Measure for a Single Node

The impurity of a node measures how dissimilar the class labels are for the data instances belonging to a common node. Following are examples of measures

that can be used to evaluate the impurity of a node t:

$$\text{Entropy} = -\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t), \tag{3.4}$$

$$\text{Gini index} = 1 - \sum_{i=0}^{c-1} p_i(t)^2, \tag{3.5}$$

$$\text{Classification error} = 1 - \max_i [p_i(t)], \tag{3.6}$$

where $p_i(t)$ is the relative frequency of training instances that belong to class i at node t, c is the total number of classes, and $0 \log_2 0 = 0$ in entropy calculations. All three measures give a zero impurity value if a node contains instances from a single class and maximum impurity if the node has equal proportion of instances from multiple classes.

Figure 3.11 compares the relative magnitude of the impurity measures when applied to binary classification problems. Since there are only two classes, $p_0(t) + p_1(t) = 1$. The horizontal axis p refers to the fraction of instances that belong to one of the two classes. Observe that all three measures attain their maximum value when the class distribution is uniform (i.e., $p_0(t) = p_1(t) = 0.5$) and minimum value when all the instances belong to a single class (i.e., either $p_0(t)$ or $p_1(t)$ equals to 1). The following examples illustrate how the values of the impurity measures vary as we alter the class distribution.

Node N_1	Count
Class=0	0
Class=1	6

Gini $= 1 - (0/6)^2 - (6/6)^2 = 0$
Entropy $= -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0$
Error $= 1 - \max[0/6, 6/6] = 0$

Node N_2	Count
Class=0	1
Class=1	5

Gini $= 1 - (1/6)^2 - (5/6)^2 = 0.278$
Entropy $= -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650$
Error $= 1 - \max[1/6, 5/6] = 0.167$

Node N_3	Count
Class=0	3
Class=1	3

Gini $= 1 - (3/6)^2 - (3/6)^2 = 0.5$
Entropy $= -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1$
Error $= 1 - \max[3/6, 3/6] = 0.5$

Based on these calculations, node N_1 has the lowest impurity value, followed by N_2 and N_3. This example, along with Figure 3.11, shows the consistency among the impurity measures, i.e., if a node N_1 has lower entropy than node N_2, then the Gini index and error rate of N_1 will also be lower than that

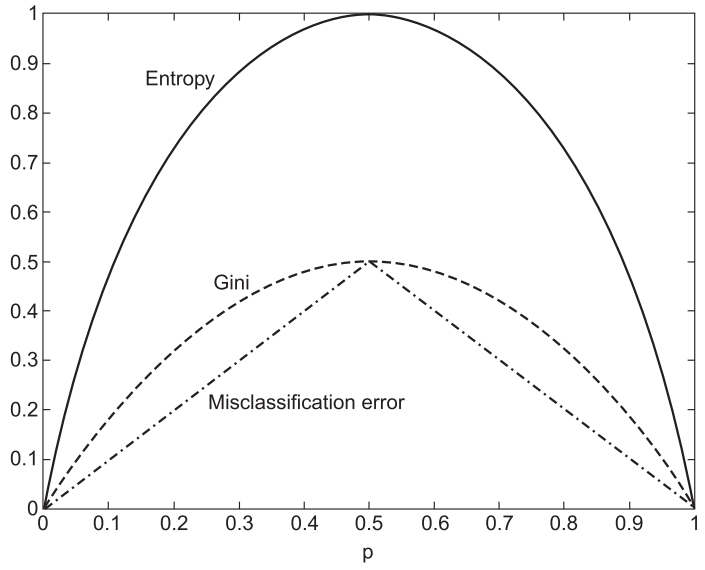


Figure 3.11. Comparison among the impurity measures for binary classification problems.

of N_2. Despite their agreement, the attribute chosen as splitting criterion by the impurity measures can still be different (see Exercise 7 on pages 206–207).

Collective Impurity of Child Nodes

Consider an attribute test condition that splits a node containing N training instances into k children, $\{v_1, v_2, \cdots, v_k\}$, where every child node represents a partition of the data resulting from one of the k outcomes of the attribute test condition. Let $N(v_j)$ be the number of training instances associated with a child node v_j, whose impurity value is $I(v_j)$. Since a training instance in the parent node reaches node v_j for a fraction of $N(v_j)/N$ times, the collective impurity of the child nodes can be computed by taking a weighted sum of the impurities of the child nodes, as follows:

$$I(\text{children}) = \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j), \tag{3.7}$$

Example 3.3. [Weighted Entropy] Consider the candidate attribute test condition shown in Figures 3.12(a) and (b) for the loan borrower classification problem. Splitting on the `Home Owner` attribute will generate two child nodes

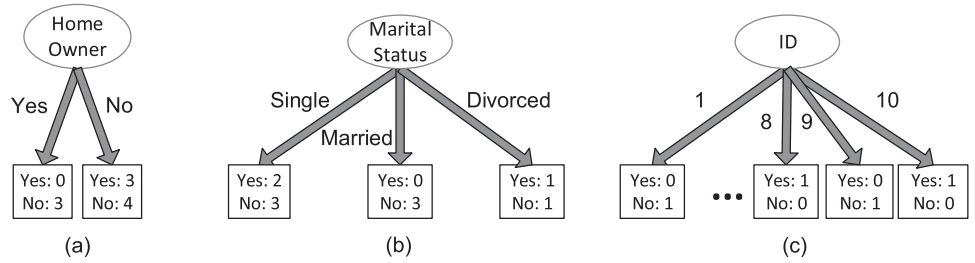


Figure 3.12. Examples of candidate attribute test conditions.

whose weighted entropy can be calculated as follows:

$$I(\texttt{Home Owner = yes}) = -\frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0$$
$$I(\texttt{Home Owner = no}) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985$$
$$I(\texttt{Home Owner}) = \frac{3}{10}\times 0 + \frac{7}{10}\times 0.985 = 0.690$$

Splitting on `Marital Status`, on the other hand, leads to three child nodes with a weighted entropy given by

$$I(\texttt{Marital Status = Single}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$$
$$I(\texttt{Marital Status = Married}) = -\frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0$$
$$I(\texttt{Marital Status = Divorced}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1.000$$
$$I(\texttt{Marital Status}) = \frac{5}{10}\times 0.971 + \frac{3}{10}\times 0 + \frac{2}{10}\times 1 = 0.686$$

Thus, `Marital Status` has a lower weighted entropy than `Home Owner`. ■

Identifying the best attribute test condition

To determine the goodness of an attribute test condition, we need to compare the degree of impurity of the parent node (before splitting) with the weighted degree of impurity of the child nodes (after splitting). The larger their difference, the better the test condition. This difference, Δ, also termed as the **gain** in purity of an attribute test condition, can be defined as follows:

$$\Delta = I(\text{parent}) - I(\text{children}), \tag{3.8}$$

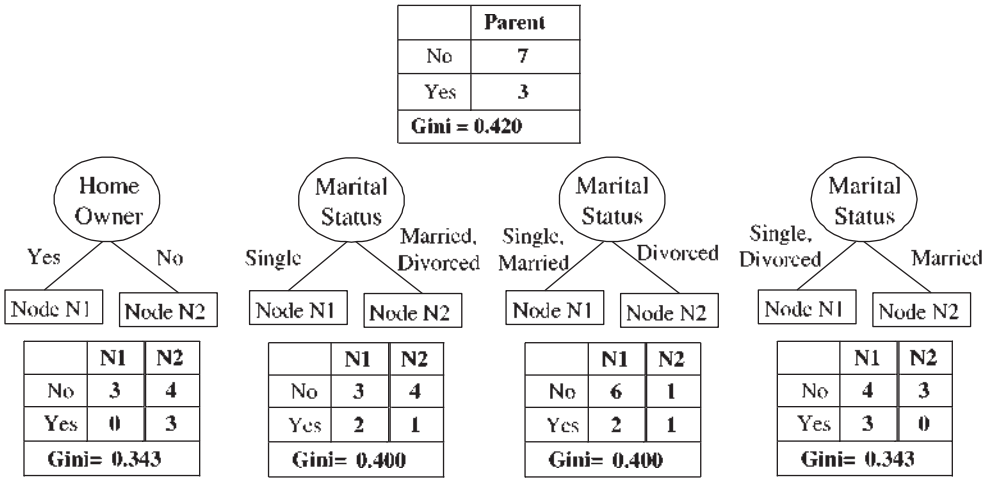


Figure 3.13. Splitting criteria for the loan borrower classification problem using Gini index.

where $I(\text{parent})$ is the impurity of a node before splitting and $I(\text{children})$ is the weighted impurity measure after splitting. It can be shown that the gain is non-negative since $I(\text{parent}) \geq I(\text{children})$ for any reasonable measure such as those presented above. The higher the gain, the purer are the classes in the child nodes relative to the parent node. The splitting criterion in the decision tree learning algorithm selects the attribute test condition that shows the maximum gain. Note that maximizing the gain at a given node is equivalent to minimizing the weighted impurity measure of its children since $I(\text{parent})$ is the same for all candidate attribute test conditions. Finally, when entropy is used as the impurity measure, the difference in entropy is commonly known as **information gain**, Δ_{info}.

In the following, we present illustrative approaches for identifying the best attribute test condition given qualitative or quantitative attributes.

Splitting of Qualitative Attributes

Consider the first two candidate splits shown in Figure 3.12 involving qualitative attributes `Home Owner` and `Marital Status`. The initial class distribution at the parent node is $(0.3, 0.7)$, since there are 3 instances of class `Yes` and 7 instances of class `No` in the training data. Thus,

$$I(\text{parent}) = -\frac{3}{10}\log_2\frac{3}{10} - \frac{7}{10}\log_2\frac{7}{10} = 0.881$$

The information gains for `Home Owner` and `Marital Status` are each given by

$$\begin{aligned} \Delta_{\text{info}}(\texttt{Home Owner}) &= 0.881 - 0.690 = 0.191 \\ \Delta_{\text{info}}(\texttt{Marital Status}) &= 0.881 - 0.686 = 0.195 \end{aligned}$$

The information gain for `Marital Status` is thus higher due to its lower weighted entropy, which will thus be considered for splitting.

Binary Splitting of Qualitative Attributes

Consider building a decision tree using only binary splits and the Gini index as the impurity measure. Figure 3.13 shows examples of four candidate splitting criteria for the `Home Owner` and `Marital Status` attributes. Since there are 3 borrowers in the training set who defaulted and 7 others who repaid their loan (see Table in Figure 3.13), the Gini index of the parent node before splitting is

$$1 - \left(\frac{3}{10}\right)^2 - \left(\frac{7}{10}\right)^2 = 0.420.$$

If `Home Owner` is chosen as the splitting attribute, the Gini index for the child nodes $\texttt{N}_1$ and $\texttt{N}_2$ are 0 and 0.490, respectively. The weighted average Gini index for the children is

$$(3/10) \times 0 + (7/10) \times 0.490 = 0.343,$$

where the weights represent the proportion of training instances assigned to each child. The gain using `Home Owner` as splitting attribute is 0.420 - 0.343 = 0.077. Similarly, we can apply a binary split on the `Marital Status` attribute. However, since `Marital Status` is a nominal attribute with three outcomes, there are three possible ways to group the attribute values into a binary split. The weighted average Gini index of the children for each candidate binary split is shown in Figure 3.13. Based on these results, `Home Owner` and the last binary split using `Marital Status` are clearly the best candidates, since they both produce the lowest weighted average Gini index. Binary splits can also be used for ordinal attributes, if the binary partitioning of the attribute values does not violate the ordering property of the values.

Binary Splitting of Quantitative Attributes

Consider the problem of identifying the best binary split $\texttt{Annual Income} \leq \tau$ for the preceding loan approval classification problem. As discussed previously,

Class	No		No		No		Yes		Yes		Yes		No		No		No		No	
	Annual Income (in '000s)																			
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	55		65		72.5		80		87.5		92.5		97.5		110		122.5		172.5		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

Figure 3.14. Splitting continuous attributes.

even though τ can take any value between the minimum and maximum values of annual income in the training set, it is sufficient to only consider the annual income values observed in the training set as candidate split positions. For each candidate τ, the training set is scanned once to count the number of borrowers with annual income less than or greater than τ along with their class proportions. We can then compute the Gini index at each candidate split position and choose the τ that produces the lowest value. Computing the Gini index at each candidate split position requires $O(N)$ operations, where N is the number of training instances. Since there are at most N possible candidates, the overall complexity of this brute-force method is $O(N^2)$. It is possible to reduce the complexity of this problem to $O(N \log N)$ by using a method described as follows (see illustration in Figure 3.14). In this method, we first sort the training instances based on their annual income, a one-time cost that requires $O(N \log N)$ operations. The candidate split positions are given by the midpoints between every two adjacent sorted values: \$55,000, \$65,000, \$72,500, and so on. For the first candidate, since none of the instances has an annual income less than or equal to \$55,000, the Gini index for the child node with `Annual Income` $<$ \$55,000 is equal to zero. In contrast, there are 3 training instances of class `Yes` and 7 instances of class `No` with annual income greater than \$55,000. The Gini index for this node is 0.420. The weighted average Gini index for the first candidate split position, $\tau = \$55,000$, is equal to $0 \times 0 + 1 \times 0.420 = 0.420$.

For the next candidate, $\tau = \$65,000$, the class distribution of its child nodes can be obtained with a simple update of the distribution for the previous candidate. This is because, as τ increases from \$55,000 to \$65,000, there is only one training instance affected by the change. By examining the class label of the affected training instance, the new class distribution is obtained. For example, as τ increases to \$65,000, there is only one borrower in the training

set, with an annual income of \$60,000, affected by this change. Since the class label for the borrower is `No`, the count for class `No` increases from 0 to 1 (for `Annual Income` $\leq$ \$65,000) and decreases from 7 to 6 (for `Annual Income` $>$ \$65,000), as shown in Figure 3.14. The distribution for the `Yes` class remains unaffected. The updated Gini index for this candidate split position is 0.400.

This procedure is repeated until the Gini index for all candidates are found. The best split position corresponds to the one that produces the lowest Gini index, which occurs at $\tau = \$97,500$. Since the Gini index at each candidate split position can be computed in $O(1)$ time, the complexity of finding the best split position is $O(N)$ once all the values are kept sorted, a one-time operation that takes $O(N \log N)$ time. The overall complexity of this method is thus $O(N \log N)$, which is much smaller than the $O(N^2)$ time taken by the brute-force method. The amount of computation can be further reduced by considering only candidate split positions located between two adjacent sorted instances with different class labels. For example, we do not need to consider candidate split positions located between \$60,000 and \$75,000 because all three instances with annual income in this range (\$60,000, \$70,000, and \$75,000) have the same class labels. Choosing a split position within this range only increases the degree of impurity, compared to a split position located outside this range. Therefore, the candidate split positions at $\tau = \$65,000$ and $\tau = \$72,500$ can be ignored. Similarly, we do not need to consider the candidate split positions at \$87,500, \$92,500, \$110,000, \$122,500, and \$172,500 because they are located between two adjacent instances with the same labels. This strategy reduces the number of candidate split positions to consider from 9 to 2 (excluding the two boundary cases $\tau = \$55,000$ and $\tau = \$230,000$).

Gain Ratio

One potential limitation of impurity measures such as entropy and Gini index is that they tend to favor qualitative attributes with large number of distinct values. Figure 3.12 shows three candidate attributes for partitioning the data set given in Table 3.3. As previously mentioned, the attribute `Marital Status` is a better choice than the attribute `Home Owner`, because it provides a larger information gain. However, if we compare them against `Customer ID`, the latter produces the purest partitions with the maximum information gain, since the weighted entropy and Gini index is equal to zero for its children. Yet, `Customer ID` is not a good attribute for splitting because it has a unique value for each instance. Even though a test condition involving `Customer ID` will accurately classify every instance in the training data, we cannot use such a test condition on new test instances with `Customer ID` values that haven't

been seen before during training. This example suggests having a low impurity value alone is insufficient to find a good attribute test condition for a node. As we will see later in Section 3.4, having more number of child nodes can make a decision tree more complex and consequently more susceptible to overfitting. Hence, the number of children produced by the splitting attribute should also be taken into consideration while deciding the best attribute test condition.

There are two ways to overcome this problem. One way is to generate only binary decision trees, thus avoiding the difficulty of handling attributes with varying number of partitions. This strategy is employed by decision tree classifiers such as CART. Another way is to modify the splitting criterion to take into account the number of partitions produced by the attribute. For example, in the C4.5 decision tree algorithm, a measure known as **gain ratio** is used to compensate for attributes that produce a large number of child nodes. This measure is computed as follows:

$$\text{Gain ratio} = \frac{\Delta_{\text{info}}}{\text{Split Info}} = \frac{\text{Entropy(Parent)} - \sum_{i=1}^{k} \frac{N(v_i)}{N} \text{Entropy}(v_i)}{-\sum_{i=1}^{k} \frac{N(v_i)}{N} \log_2 \frac{N(v_i)}{N}} \quad (3.9)$$

where $N(v_i)$ is the number of instances assigned to node v_i and k is the total number of splits. The split information measures the entropy of splitting a node into its child nodes and evaluates if the split results in a larger number of equally-sized child nodes or not. For example, if every partition has the same number of instances, then $\forall i : N(v_i)/N = 1/k$ and the split information would be equal to $\log_2 k$. Thus, if an attribute produces a large number of splits, its split information is also large, which in turn, reduces the gain ratio.

Example 3.4. [Gain Ratio] Consider the data set given in Exercise 3 on page 205. We want to select the best attribute test condition among the following three attributes: `Gender`, `Car Type`, and `Customer ID`. The entropy before splitting is

$$\text{Entropy(parent)} = -\frac{10}{20}\log_2\frac{10}{20} - \frac{10}{20}\log_2\frac{10}{20} = 1.$$

If `Gender` is used as attribute test condition:

$$\begin{aligned} \text{Entropy(children)} &= \frac{10}{20}\left[-\frac{6}{10}\log_2\frac{6}{10} - \frac{4}{10}\log_2\frac{4}{10}\right] \times 2 = 0.971 \\ \text{Gain Ratio} &= \frac{1 - 0.971}{-\frac{10}{20}\log_2\frac{10}{20} - \frac{10}{20}\log_2\frac{10}{20}} = \frac{0.029}{1} = 0.029 \end{aligned}$$

If `Car Type` is used as attribute test condition:

$$\begin{aligned}\text{Entropy(children)} &= \frac{4}{20}\left[-\frac{1}{4}\log_2\frac{1}{4}-\frac{3}{4}\log_2\frac{3}{4}\right]+\frac{8}{20}\times 0 \\ &\quad + \frac{8}{20}\left[-\frac{1}{8}\log_2\frac{1}{8}-\frac{7}{8}\log_2\frac{7}{8}\right] = 0.380 \\ \text{Gain Ratio} &= \frac{1-0.380}{-\frac{4}{20}\log_2\frac{4}{20}-\frac{8}{20}\log_2\frac{8}{20}-\frac{8}{20}\log_2\frac{8}{20}} = \frac{0.620}{1.52} = 0.41\end{aligned}$$

Finally, if `Customer ID` is used as attribute test condition:

$$\begin{aligned}\text{Entropy(children)} &= \frac{1}{20}\left[-\frac{1}{1}\log_2\frac{1}{1}-\frac{0}{1}\log_2\frac{0}{1}\right]\times 20 = 0 \\ \text{Gain Ratio} &= \frac{1-0}{-\frac{1}{20}\log_2\frac{1}{20}\times 20} = \frac{1}{4.32} = 0.23\end{aligned}$$

Thus, even though `Customer ID` has the highest information gain, its gain ratio is lower than `Car Type` since it produces a larger number of splits. ■

3.3.4 Algorithm for Decision Tree Induction

Algorithm 3.1 presents a pseudocode for decision tree induction algorithm. The input to this algorithm is a set of training instances E along with the attribute set F. The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the nodes of the tree (Steps 11 and 12) until the stopping criterion is met (Step 1). The details of this algorithm are explained below.

1. The `createNode`() function extends the decision tree by creating a new node. A node in the decision tree either has a test condition, denoted as *node.test_cond*, or a class label, denoted as *node.label*.

2. The `find_best_split`() function determines the attribute test condition for partitioning the training instances associated with a node. The splitting attribute chosen depends on the impurity measure used. The popular measures include entropy and the Gini index.

3. The `Classify`() function determines the class label to be assigned to a leaf node. For each leaf node t, let $p(i|t)$ denote the fraction of training instances from class i associated with the node t. The label assigned

Algorithm 3.1 A skeleton decision tree induction algorithm.

```
TreeGrowth (E, F)
 1: if stopping_cond(E,F) = true then
 2:    leaf = createNode().
 3:    leaf.label = Classify(E).
 4:    return leaf.
 5: else
 6:    root = createNode().
 7:    root.test_cond = find_best_split(E, F).
 8:    let V = {v|v is a possible outcome of root.test_cond }.
 9:    for each v ∈ V do
10:       E_v = {e | root.test_cond(e) = v and e ∈ E}.
11:       child = TreeGrowth(E_v, F).
12:       add child as descendent of root and label the edge (root → child) as v.
13:    end for
14: end if
15: return root.
```

to the leaf node is typically the one that occurs most frequently in the training instances that are associated with this node.

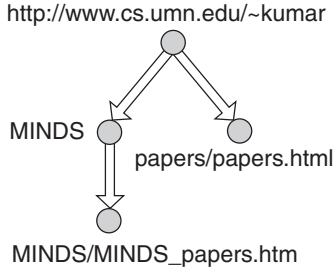
$$leaf.label = \underset{i}{\text{argmax}}\ p(i|t), \tag{3.10}$$

where the argmax operator returns the class i that maximizes $p(i|t)$. Besides providing the information needed to determine the class label of a leaf node, $p(i|t)$ can also be used as a rough estimate of the probability that an instance assigned to the leaf node t belongs to class i. Sections 6.11.2 and 6.11.4 in the next chapter describe how such probability estimates can be used to determine the performance of a decision tree under different cost functions.

4. The `stopping_cond`() function is used to terminate the tree-growing process by checking whether all the instances have identical class label or attribute values. Since decision tree classifiers employ a top-down, recursive partitioning approach for building a model, the number of training instances associated with a node decreases as the depth of the tree increases. As a result, a leaf node may contain too few training instances to make a statistically significant decision about its class label. This is known as the **data fragmentation** problem. One way to avoid this problem is to disallow splitting of a node when the number of instances associated with the node fall below a certain threshold. A

Session	IP Address	Timestamp	Request Method	Requested Web Page	Protocol	Status	Number of Bytes	Referrer	User Agent
1	160.11.11.11	08/Aug/2004 10:15:21	GET	http://www.cs.umn.edu/~kumar	HTTP/1.1	200	6424		Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:34	GET	http://www.cs.umn.edu/~kumar/MINDS	HTTP/1.1	200	41378	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:41	GET	http://www.cs.umn.edu/~kumar/MINDS/MINDS_papers.htm	HTTP/1.1	200	1018516	http://www.cs.umn.edu/~kumar/MINDS	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:16:11	GET	http://www.cs.umn.edu/~kumar/papers/papers.html	HTTP/1.1	200	7463	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
2	35.9.2.2	08/Aug/2004 10:16:15	GET	http://www.cs.umn.edu/~steinbac	HTTP/1.0	200	3149		Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040616

(a) Example of a Web server log.



(b) Graph of a Web session.

Attribute Name	Description
totalPages	Total number of pages retrieved in a Web session
ImagePages	Total number of image pages retrieved in a Web session
TotalTime	Total amount of time spent by Web site visitor
RepeatedAccess	The same page requested more than once in a Web session
ErrorRequest	Errors in requesting for Web pages
GET	Percentage of requests made using GET method
POST	Percentage of requests made using POST method
HEAD	Percentage of requests made using HEAD method
Breadth	Breadth of Web traversal
Depth	Depth of Web traversal
MultiIP	Session with multiple IP addresses
MultiAgent	Session with multiple user agents

(c) Derived attributes for Web robot detection.

Figure 3.15. Input data for web robot detection.

more systematic way to control the size of a decision tree (number of leaf nodes) will be discussed in Section 3.5.4.

3.3.5 Example Application: Web Robot Detection

Consider the task of distinguishing the access patterns of web robots from those generated by human users. A web robot (also known as a web crawler) is a software program that automatically retrieves files from one or more websites by following the hyperlinks extracted from an initial set of seed URLs. These programs have been deployed for various purposes, from gathering web pages on behalf of search engines to more malicious activities such as spamming and committing click frauds in online advertisements.

The web robot detection problem can be cast as a binary classification task. The input data for the classification task is a web server log, a sample of which is shown in Figure 3.15(a). Each line in the log file corresponds to a

request made by a client (i.e., a human user or a web robot) to the web server. The fields recorded in the web log include the client's IP address, timestamp of the request, URL of the requested file, size of the file, and **user agent**, which is a field that contains identifying information about the client. For human users, the user agent field specifies the type of web browser or mobile device used to fetch the files, whereas for web robots, it should technically contain the name of the crawler program. However, web robots may conceal their true identities by declaring their user agent fields to be identical to known browsers. Therefore, user agent is not a reliable field to detect web robots.

The first step toward building a classification model is to precisely define a data instance and associated attributes. A simple approach is to consider each log entry as a data instance and use the appropriate fields in the log file as its attribute set. This approach, however, is inadequate for several reasons. First, many of the attributes are nominal-valued and have a wide range of domain values. For example, the number of unique client IP addresses, URLs, and referrers in a log file can be very large. These attributes are undesirable for building a decision tree because their split information is extremely high (see Equation (3.9)). In addition, it might not be possible to classify test instances containing IP addresses, URLs, or referrers that are not present in the training data. Finally, by considering each log entry as a separate data instance, we disregard the sequence of web pages retrieved by the client—a critical piece of information that can help distinguish web robot accesses from those of a human user.

A better alternative is to consider each web session as a data instance. A web session is a sequence of requests made by a client during a given visit to the website. Each web session can be modeled as a directed graph, in which the nodes correspond to web pages and the edges correspond to hyperlinks connecting one web page to another. Figure 3.15(b) shows a graphical representation of the first web session given in the log file. Every web session can be characterized using some meaningful attributes about the graph that contain discriminatory information. Figure 3.15(c) shows some of the attributes extracted from the graph, including the `depth` and `breadth` of its corresponding tree rooted at the entry point to the website. For example, the `depth` and `breadth` of the tree shown in Figure 3.15(b) are both equal to two.

The derived attributes shown in Figure 3.15(c) are more informative than the original attributes given in the log file because they characterize the behavior of the client at the website. Using this approach, a data set containing 2916 instances was created, with equal numbers of sessions due to web robots (class 1) and human users (class 0). 10% of the data were reserved for training while the remaining 90% were used for testing. The induced decision tree is

shown in Figure 3.16, which has an error rate equal to 3.8% on the training set and 5.3% on the test set. In addition to its low error rate, the tree also reveals some interesting properties that can help discriminate web robots from human users:

1. Accesses by web robots tend to be broad but shallow, whereas accesses by human users tend to be more focused (narrow but deep).

2. Web robots seldom retrieve the image pages associated with a web page.

3. Sessions due to web robots tend to be long and contain a large number of requested pages.

4. Web robots are more likely to make repeated requests for the same web page than human users since the web pages retrieved by human users are often cached by the browser.

3.3.6 Characteristics of Decision Tree Classifiers

The following is a summary of the important characteristics of decision tree induction algorithms.

1. **Applicability:** Decision trees are a nonparametric approach for building classification models. This approach does not require any prior assumption about the probability distribution governing the class and attributes of the data, and thus, is applicable to a wide variety of data sets. It is also applicable to both categorical and continuous data without requiring the attributes to be transformed into a common representation via binarization, normalization, or standardization. Unlike some binary classifiers described in Chapter 6, it can also deal with multiclass problems without the need to decompose them into multiple binary classification tasks. Another appealing feature of decision tree classifiers is that the induced trees, especially the shorter ones, are relatively easy to interpret. The accuracies of the trees are also quite comparable to other classification techniques for many simple data sets.

2. **Expressiveness:** A decision tree provides a universal representation for discrete-valued functions. In other words, it can encode any function of discrete-valued attributes. This is because every discrete-valued function can be represented as an assignment table, where every unique combination of discrete attributes is assigned a class label. Since every

```
Decision Tree:
depth = 1:
| breadth> 7 : class 1
| breadth<= 7:
| | breadth <= 3:
| | | ImagePages> 0.375: class 0
| | | ImagePages<= 0.375:
| | | | totalPages<= 6: class 1
| | | | totalPages> 6:
| | | | | breadth <= 1: class 1
| | | | | breadth > 1: class 0
| | width > 3:
| | | MultiIP = 0:
| | | | ImagePages<= 0.1333: class 1
| | | | ImagePages> 0.1333:
| | | | breadth <= 6: class 0
| | | | breadth > 6: class 1
| | | MultiIP = 1:
| | | | TotalTime <= 361: class 0
| | | | TotalTime > 361: class 1
depth> 1:
| MultiAgent = 0:
| | depth > 2: class 0
| | depth < 2:
| | | MultiIP = 1: class 0
| | | MultiIP = 0:
| | | | breadth <= 6: class 0
| | | | breadth > 6:
| | | | | RepeatedAccess <= 0.322: class 0
| | | | | RepeatedAccess > 0.322: class 1
| MultiAgent = 1:
| | totalPages <= 81: class 0
| | totalPages > 81: class 1
```

Figure 3.16. Decision tree model for web robot detection.

combination of attributes can be represented as a leaf in the decision tree, we can always find a decision tree whose label assignments at the leaf nodes matches with the assignment table of the original function. Decision trees can also help in providing compact representations of functions when some of the unique combinations of attributes can be represented by the same leaf node. For example, Figure 3.17 shows the assignment table of the Boolean function $(A \wedge B) \vee (C \wedge D)$ involving four binary attributes, resulting in a total of $2^4 = 16$ possible assignments. The tree shown in Figure 3.17 shows a compressed encoding of this assignment table. Instead of requiring a fully-grown tree with 16 leaf nodes, it is possible to encode the function using a simpler tree with only 7 leaf nodes. Nevertheless, not all decision trees for discrete-valued attributes can be simplified. One notable example is the parity function,

A	B	C	D	class
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

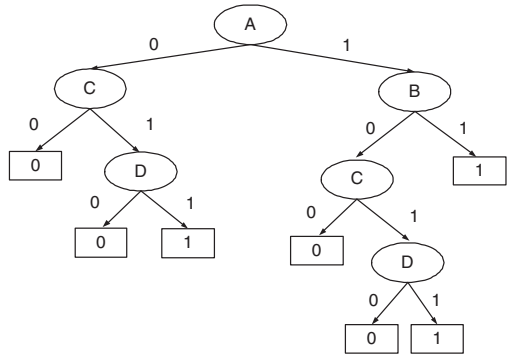


Figure 3.17. Decision tree for the Boolean function $(A \wedge B) \vee (C \wedge D)$.

whose value is 1 when there is an even number of true values among its Boolean attributes, and 0 otherwise. Accurate modeling of such a function requires a full decision tree with 2^d nodes, where d is the number of Boolean attributes (see Exercise 2 on page 205).

3. **Computational Efficiency:** Since the number of possible decision trees can be very large, many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space. For example, the algorithm presented in Section 3.3.4 uses a greedy, top-down, recursive partitioning strategy for growing a decision tree. For many data sets, such techniques quickly construct a reasonably good decision tree even when the training set size is very large. Furthermore, once a decision tree has been built, classifying a test record is extremely fast, with a worst-case complexity of $O(w)$, where w is the maximum depth of the tree.

4. **Handling Missing Values:** A decision tree classifier can handle missing attribute values in a number of ways, both in the training and the test sets. When there are missing values in the test set, the classifier must decide which branch to follow if the value of a splitting node attribute is missing for a given test instance. One approach, known as the **probabilistic split method**, which is employed by the C4.5 decision tree classifier, distributes the data instance to every child of the splitting node according to the probability that the missing attribute has a particular value. In contrast, the CART algorithm uses the **surrogate split method**, where the instance whose splitting attribute value is

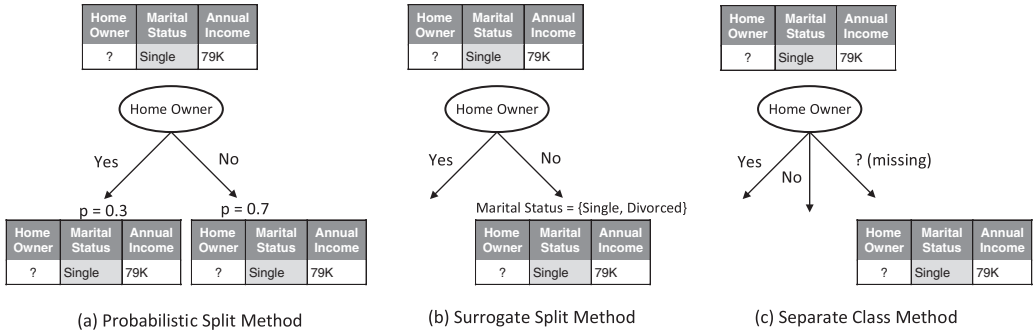


Figure 3.18. Methods for handling missing attribute values in decision tree classifier.

missing is assigned to one of the child nodes based on the value of another non-missing surrogate attribute whose splits most resemble the partitions made by the missing attribute. Another approach, known as the **separate class method** is used by the CHAID algorithm, where the missing value is treated as a separate categorical value distinct from other values of the splitting attribute. Figure 3.18 shows an example of the three different ways for handling missing values in a decision tree classifier. Other strategies for dealing with missing values are based on data preprocessing, where the instance with missing value is either imputed with the mode (for categorical attribute) or mean (for continuous attribute) value or discarded before the classifier is trained.

During training, if an attribute v has missing values in some of the training instances associated with a node, we need a way to measure the gain in purity if v is used for splitting. One simple way is to exclude instances with missing values of v in the counting of instances associated with every child node, generated for every possible outcome of v. Further, if v is chosen as the attribute test condition at a node, training instances with missing values of v can be propagated to the child nodes using any of the methods described above for handling missing values in test instances.

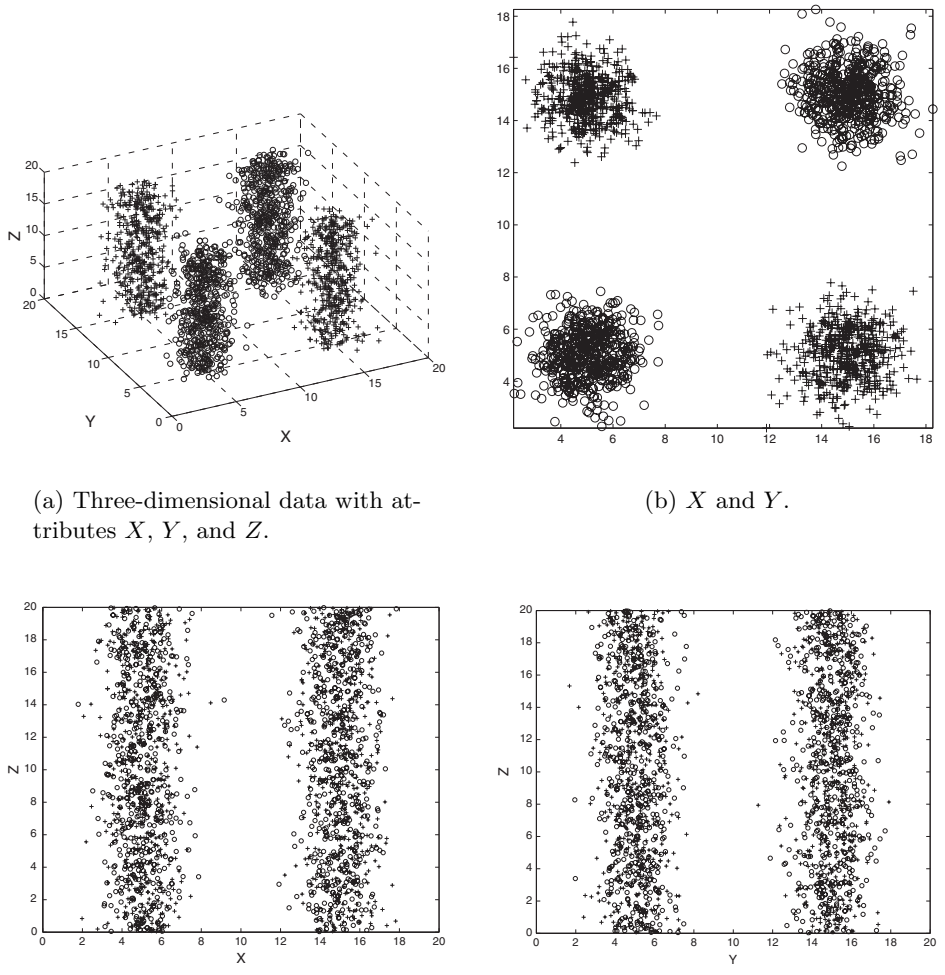
5. **Handling Interactions among Attributes:** Attributes are considered interacting if they are able to distinguish between classes when used together, but individually they provide little or no information. Due to the greedy nature of the splitting criteria in decision trees, such attributes could be passed over in favor of other attributes that are not as useful. This could result in more complex decision trees than necessary.

Hence, decision trees can perform poorly when there are interactions among attributes.

To illustrate this point, consider the three-dimensional data shown in Figure 3.19(a), which contains 2000 data points from one of two classes, denoted as + and ∘ in the diagram. Figure 3.19(b) shows the distribution of the two classes in the two-dimensional space involving attributes X and Y, which is a noisy version of the XOR Boolean function. We can see that even though the two classes are well-separated in this two-dimensional space, neither of the two attributes contain sufficient information to distinguish between the two classes when used alone. For example, the entropies of the following attribute test conditions: $X \leq 10$ and $Y \leq 10$, are close to 1, indicating that neither X nor Y provide any reduction in the impurity measure when used individually. X and Y thus represent a case of interaction among attributes. The data set also contains a third attribute, Z, in which both classes are distributed uniformly, as shown in Figures 3.19(c) and 3.19(d), and hence, the entropy of any split involving Z is close to 1. As a result, Z is as likely to be chosen for splitting as the interacting but useful attributes, X and Y. For further illustration of this issue, readers are referred to Example 3.7 in Section 3.4.1 and Exercise 8 at the end of this chapter.

6. **Handling Irrelevant Attributes:** An attribute is irrelevant if it is not useful for the classification task. Since irrelevant attributes are poorly associated with the target class labels, they will provide little or no gain in purity and thus will be passed over by other more relevant features. Hence, the presence of a small number of irrelevant attributes will not impact the decision tree construction process. However, not all attributes that provide little to no gain are irrelevant (see Figure 3.19). Hence, if the classification problem is complex (e.g., involving interactions among attributes) and there are a large number of irrelevant attributes, then some of these attributes may be accidentally chosen during the tree-growing process, since they may provide a better gain than a relevant attribute just by random chance. Feature selection techniques can help to improve the accuracy of decision trees by eliminating the irrelevant attributes during preprocessing. We will investigate the issue of too many irrelevant attributes in Section 3.4.1.

7. **Handling Redundant Attributes:** An attribute is redundant if it is strongly correlated with another attribute in the data. Since redundant



(a) Three-dimensional data with attributes X, Y, and Z.

(b) X and Y.

(c) X and Z.

(d) Y and Z.

Figure 3.19. Example of a XOR data involving X and Y, along with an irrelevant attribute Z.

attributes show similar gains in purity if they are selected for splitting, only one of them will be selected as an attribute test condition in the decision tree algorithm. Decision trees can thus handle the presence of redundant attributes.

8. **Using Rectilinear Splits:** The test conditions described so far in this chapter involve using only a single attribute at a time. As a consequence,

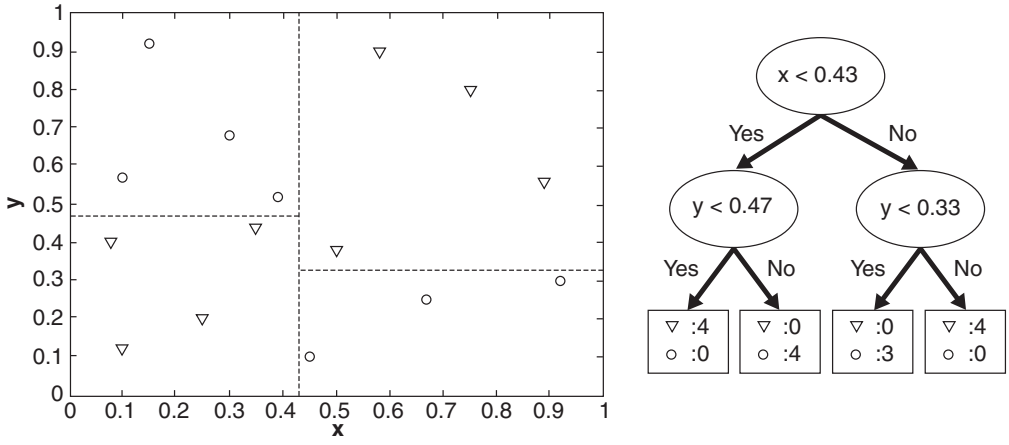


Figure 3.20. Example of a decision tree and its decision boundaries for a two-dimensional data set.

the tree-growing procedure can be viewed as the process of partitioning the attribute space into disjoint regions until each region contains records of the same class. The border between two neighboring regions of different classes is known as a **decision boundary**. Figure 3.20 shows the decision tree as well as the decision boundary for a binary classification problem. Since the test condition involves only a single attribute, the decision boundaries are rectilinear; i.e., parallel to the coordinate axes. This limits the expressiveness of decision trees in representing decision boundaries of data sets with continuous attributes. Figure 3.21 shows a two-dimensional data set involving binary classes that cannot be perfectly classified by a decision tree whose attribute test conditions are defined based on single attributes. The binary classes in the data set are generated from two skewed Gaussian distributions, centered at (8,8) and (12,12), respectively. The true decision boundary is represented by the diagonal dashed line, whereas the rectilinear decision boundary produced by the decision tree classifier is shown by the thick solid line.

In contrast, an **oblique decision tree** may overcome this limitation by allowing the test condition to be specified using more than one attribute. For example, the binary classification data shown in Figure 3.21 can be easily represented by an oblique decision tree with a single root node with test condition

$$x + y < 20.$$

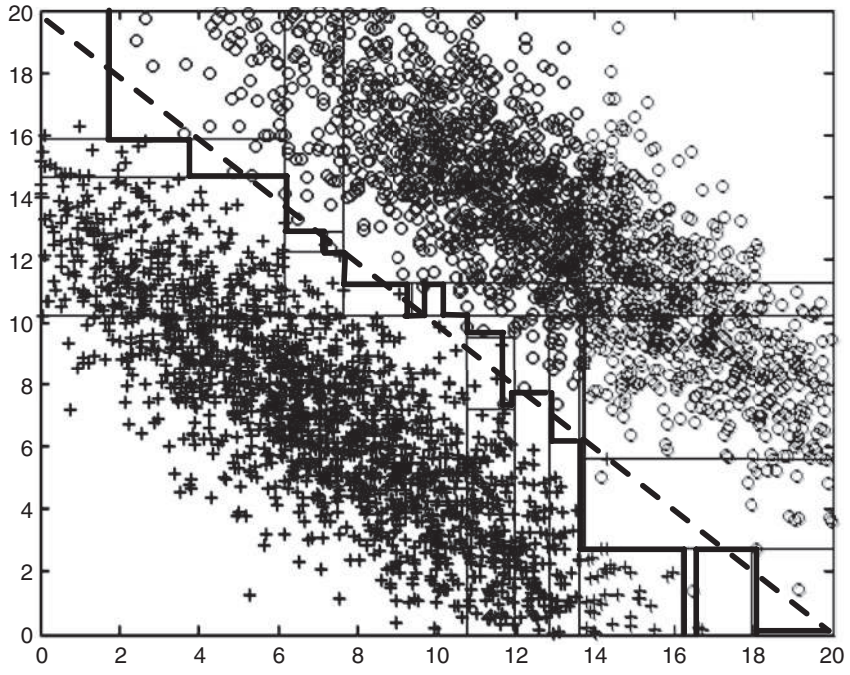


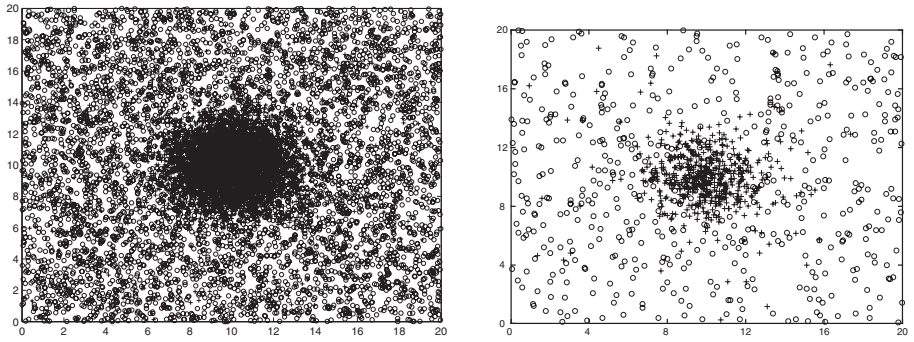
Figure 3.21. Example of data set that cannot be partitioned optimally using a decision tree with single attribute test conditions. The true decision boundary is shown by the dashed line.

Although an oblique decision tree is more expressive and can produce more compact trees, finding the optimal test condition is computationally more expensive.

9. **Choice of Impurity Measure:** It should be noted that the choice of impurity measure often has little effect on the performance of decision tree classifiers since many of the impurity measures are quite consistent with each other, as shown in Figure 3.11 on page 149. Instead, the strategy used to prune the tree has a greater impact on the final tree than the choice of impurity measure.

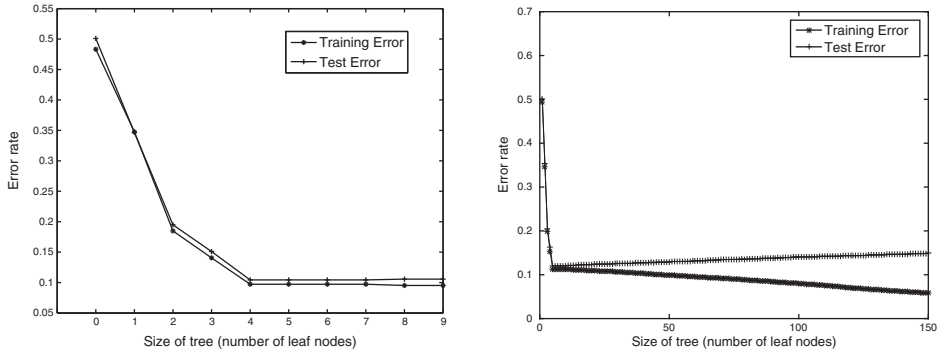
3.4 Model Overfitting

Methods presented so far try to learn classification models that show the lowest error on the training set. However, as we will show in the following example, even if a model fits well over the training data, it can still show poor generalization performance, a phenomenon known as model overfitting.



(a) Example of a 2-D data. (b) Training set using 10% data.

Figure 3.22. Examples of training and test sets of a two-dimensional classification problem.



(a) Varying tree size from 1 to 8. (b) Varying tree size from 1 to 150.

Figure 3.23. Effect of varying tree size (number of leaf nodes) on training and test errors.

Example 3.5. [Overfitting and Underfitting of Decision Trees] Consider the two-dimensional data set shown in Figure 3.22(a). The data set contains instances that belong to two separate classes, represented as + and o, respectively, where each class has 5400 instances. All instances belonging to the o class were generated from a uniform distribution. For the + class, 5000 instances were generated from a Gaussian distribution centered at (10,10) with unit variance, while the remaining 400 instances were sampled from the same uniform distribution as the o class. We can see from Figure 3.22(a) that

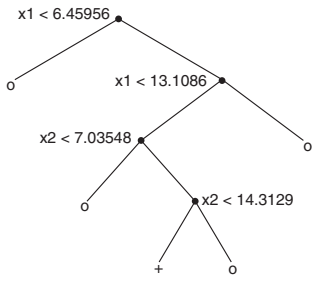
the + class can be largely distinguished from the o class by drawing a circle of appropriate size centered at (10,10). To learn a classifier using this two-dimensional data set, we randomly sampled 10% of the data for training and used the remaining 90% for testing. The training set, shown in Figure 3.22(b), looks quite representative of the overall data. We used Gini index as the impurity measure to construct decision trees of increasing sizes (number of leaf nodes), by recursively expanding a node into child nodes till every leaf node was pure, as described in Section 3.3.4.

Figure 3.23(a) shows changes in the training and test error rates as the size of the tree varies from 1 to 8. Both error rates are initially large when the tree has only one or two leaf nodes. This situation is known as **model underfitting**. Underfitting occurs when the learned decision tree is too simplistic, and thus, incapable of fully representing the true relationship between the attributes and the class labels. As we increase the tree size from 1 to 8, we can observe two effects. First, both the error rates decrease since larger trees are able to represent more complex decision boundaries. Second, the training and test error rates are quite close to each other, which indicates that the performance on the training set is fairly representative of the generalization performance. As we further increase the size of the tree from 8 to 150, the training error continues to steadily decrease till it eventually reaches zero, as shown in Figure 3.23(b). However, in a striking contrast, the test error rate ceases to decrease any further beyond a certain tree size, and then it begins to increase. The training error rate thus grossly under-estimates the test error rate once the tree becomes too large. Further, the gap between the training and test error rates keeps on widening as we increase the tree size. This behavior, which may seem counter-intuitive at first, can be attributed to the phenomena of **model overfitting**.

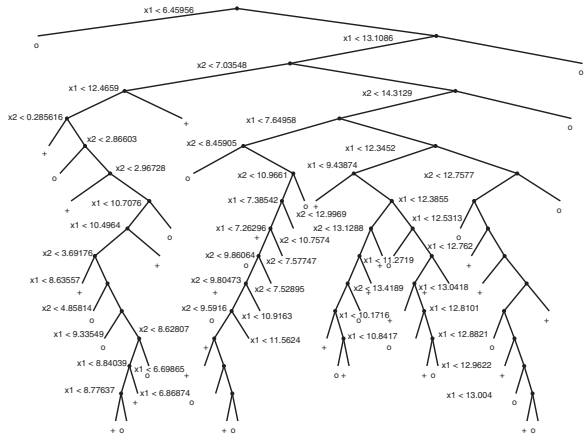
■

3.4.1 Reasons for Model Overfitting

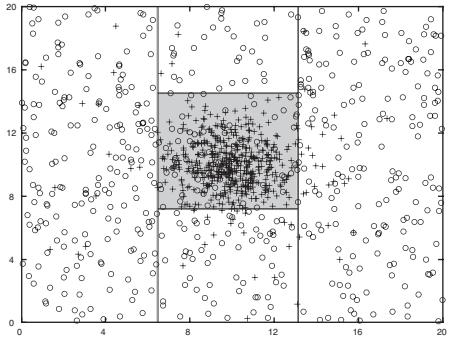
Model overfitting is the phenomena where, in the pursuit of minimizing the training error rate, an overly complex model is selected that captures specific patterns in the training data but fails to learn the *true* nature of relationships between attributes and class labels in the overall data. To illustrate this, Figure 3.24 shows decision trees and their corresponding decision boundaries (shaded rectangles represent regions assigned to the + class) for two trees of sizes 5 and 50. We can see that the decision tree of size 5 appears quite simple and its decision boundaries provide a reasonable approximation to the ideal decision boundary, which in this case corresponds to a circle centered around



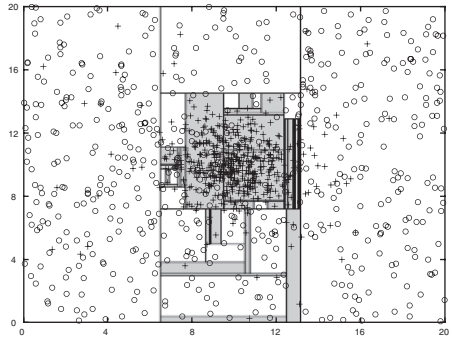
(a) Decision tree with 5 leaf nodes.



(b) Decision tree with 50 leaf nodes.



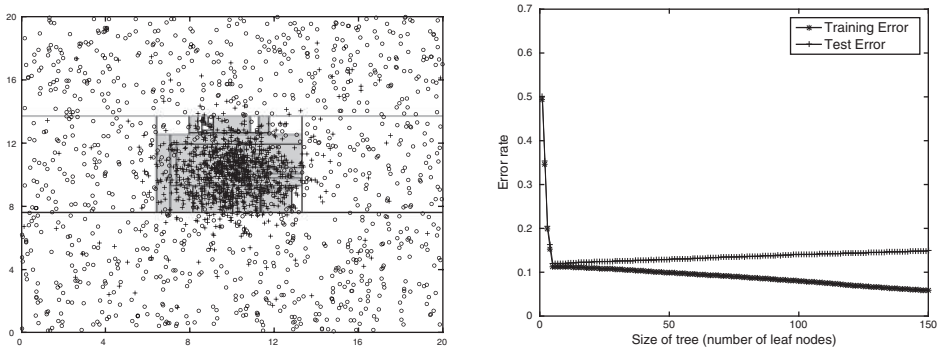
(c) Decision boundary for tree with 5 leaf nodes.



(d) Decision boundary for tree with 50 leaf nodes.

Figure 3.24. Decision trees with different model complexities.

the Gaussian distribution at (10, 10). Although its training and test error rates are non-zero, they are very close to each other, which indicates that the patterns learned in the training set should generalize well over the test set. On the other hand, the decision tree of size 50 appears much more complex than the tree of size 5, with complicated decision boundaries. For example, some of its shaded rectangles (assigned the + class) attempt to cover narrow regions in the input space that contain only one or two + training instances. Note that



(a) Decision boundary for tree with 50 leaf nodes using 20% data for training.

(b) Training and test error rates using 20% data for training.

Figure 3.25. Performance of decision trees using 20% data for training (twice the original training size).

the prevalence of + instances in such regions is highly specific to the training set, as these regions are mostly dominated by − instances in the overall data. Hence, in an attempt to perfectly fit the training data, the decision tree of size 50 starts fine tuning itself to specific patterns in the training data, leading to poor performance on an independently chosen test set.

There are a number of factors that influence model overfitting. In the following, we provide brief descriptions of two of the major factors: limited training size and high model complexity. Though they are not exhaustive, the interplay between them can help explain most of the common model overfitting phenomena in real-world applications.

Limited Training Size

Note that a training set consisting of a finite number of instances can only provide a limited representation of the overall data. Hence, it is possible that the patterns learned from a training set do not fully represent the true patterns in the overall data, leading to model overfitting. In general, as we increase the size of a training set (number of training instances), the patterns learned from the training set start resembling the true patterns in the overall data. Hence, the effect of overfitting can be reduced by increasing the training size, as illustrated in the following example.

Example 3.6. [Effect of Training Size] Suppose that we use twice the number of training instances than what we had used in the experiments conducted in Example 3.5. Specifically, we use 20% data for training and use the remainder for testing. Figure 3.25(b) shows the training and test error rates as the size of the tree is varied from 1 to 150. There are two major differences in the trends shown in this figure and those shown in Figure 3.23(b) (using only 10% of the data for training). First, even though the training error rate decreases with increasing tree size in both figures, its rate of decrease is much smaller when we use twice the training size. Second, for a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size. These differences suggest that the patterns learned using 20% of data for training are more generalizable than those learned using 10% of data for training.

Figure 3.25(a) shows the decision boundaries for the tree of size 50, learned using 20% of data for training. In contrast to the tree of the same size learned using 10% data for training (see Figure 3.24(d)), we can see that the decision tree is not capturing specific patterns of noisy + instances in the training set. Instead, the high model complexity of 50 leaf nodes is being effectively used to learn the boundaries of the + instances centered at (10, 10).

■

High Model Complexity

Generally, a more complex model has a better ability to represent complex patterns in the data. For example, decision trees with larger number of leaf nodes can represent more complex decision boundaries than decision trees with fewer leaf nodes. However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalize well over unseen instances. Models with high complexity should thus be judiciously used to avoid overfitting.

One measure of model complexity is the number of "parameters" that need to be inferred from the training set. For example, in the case of decision tree induction, the attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set. A decision tree with larger number of attribute test conditions (and consequently more leaf nodes) thus involves more "parameters" and hence is more complex.

Given a class of models with a certain number of parameters, a learning algorithm attempts to select the best combination of parameter values that maximizes an evaluation metric (e.g., accuracy) over the training set. If the number of parameter value combinations (and hence the complexity) is large,

the learning algorithm has to select the best combination from a large number of possibilities, using a limited training set. In such cases, there is a high chance for the learning algorithm to pick a *spurious* combination of parameters that maximizes the evaluation metric just by random chance. This is similar to the **multiple comparisons problem** (also referred as multiple testing problem) in statistics.

As an illustration of the multiple comparisons problem, consider the task of predicting whether the stock market will rise or fall in the next ten trading days. If a stock analyst simply makes random guesses, the probability that her prediction is correct on any trading day is 0.5. However, the probability that she will predict correctly at least nine out of ten times is

$$\frac{\binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0107,$$

which is extremely low.

Suppose we are interested in choosing an investment advisor from a pool of 200 stock analysts. Our strategy is to select the analyst who makes the most number of correct predictions in the next ten trading days. The flaw in this strategy is that even if all the analysts make their predictions in a random fashion, the probability that at least one of them makes at least nine correct predictions is

$$1 - (1 - 0.0107)^{200} = 0.8847,$$

which is very high. Although each analyst has a low probability of predicting at least nine times correctly, considered together, we have a high probability of finding at least one analyst who can do so. However, there is no guarantee in the future that such an analyst will continue to make accurate predictions by random guessing.

How does the multiple comparisons problem relate to model overfitting? In the context of learning a classification model, each combination of parameter values corresponds to an analyst, while the number of training instances corresponds to the number of days. Analogous to the task of selecting the best analyst who makes the most accurate predictions on consecutive days, the task of a learning algorithm is to select the best combination of parameters that results in the highest accuracy on the training set. If the number of parameter combinations is large but the training size is small, it is highly likely for the learning algorithm to choose a spurious parameter combination that provides high training accuracy just by random chance. In the following example, we illustrate the phenomena of overfitting due to multiple comparisons in the context of decision tree induction.

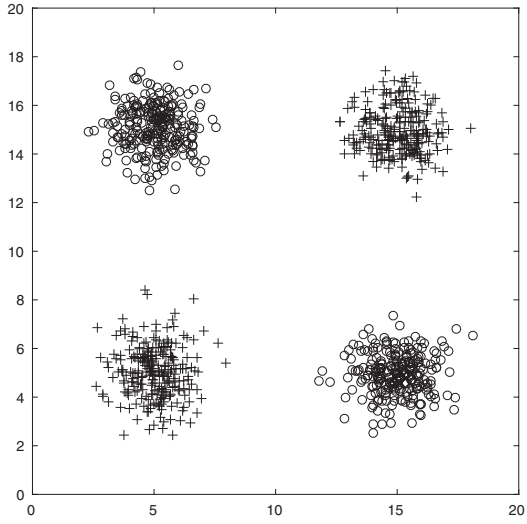
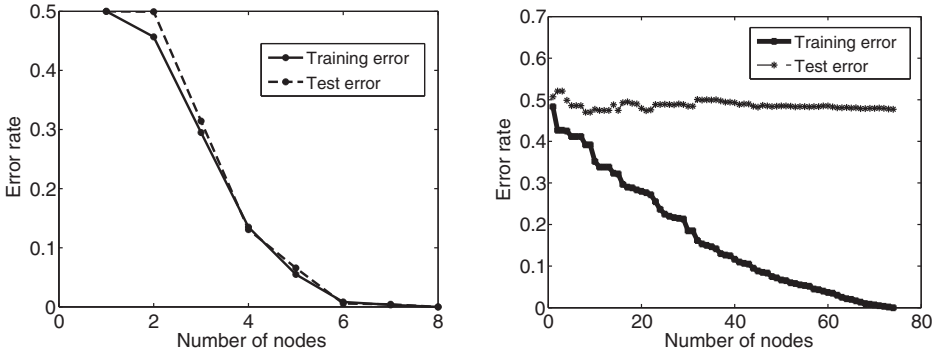


Figure 3.26. Example of a two-dimensional (X-Y) data set.

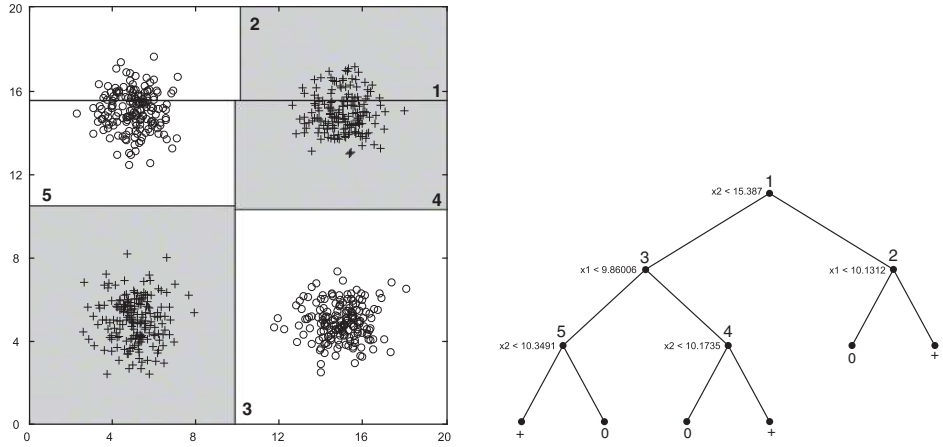


(a) Using X and Y attributes only.

(b) After adding 100 irrelevant attributes.

Figure 3.27. Training and test error rates illustrating the effect of multiple comparisons problem on model overfitting.

Example 3.7. [Multiple Comparisons and Overfitting] Consider the two-dimensional data set shown in Figure 3.26 containing 500 + and 500 o instances, which is similar to the data shown in Figure 3.19. In this data set, the distributions of both classes are well-separated in the two-dimensional (X-Y) attribute space, but none of the two attributes (X or Y) are sufficiently informative to be used alone for separating the two classes. Hence, splitting



(a) Decision boundary for tree with 6 leaf nodes.

(b) Decision tree with 6 leaf nodes.

Figure 3.28. Decision tree with 6 leaf nodes using X and Y as attributes. Splits have been numbered from 1 to 5 in order of other occurrence in the tree.

the data set based on any value of an X or Y attribute will provide close to zero reduction in an impurity measure. However, if X and Y attributes are used together in the splitting criterion (e.g., splitting X at 10 *and* Y at 10), the two classes can be effectively separated.

Figure 3.27(a) shows the training and test error rates for learning decision trees of varying sizes, when 30% of the data is used for training and the remainder of the data for testing. We can see that the two classes can be separated using a small number of leaf nodes. Figure 3.28 shows the decision boundaries for the tree with six leaf nodes, where the splits have been numbered according to their order of appearance in the tree. Note that the even though splits 1 and 3 provide trivial gains, their consequent splits (2, 4, and 5) provide large gains, resulting in effective discrimination of the two classes.

Assume we add 100 irrelevant attributes to the two-dimensional X-Y data. Learning a decision tree from this resultant data will be challenging because the number of candidate attributes to choose for splitting at every internal node will increase from two to 102. With such a large number of candidate attribute test conditions to choose from, it is quite likely that spurious attribute test conditions will be selected at internal nodes because of the multiple comparisons problem. Figure 3.27(b) shows the training and test error rates after adding 100 irrelevant attributes to the training set. We can see that the

test error rate remains close to 0.5 even after using 50 leaf nodes, while the training error rate keeps on declining and eventually becomes 0.

■

3.5 Model Selection

There are many possible classification models with varying levels of model complexity that can be used to capture patterns in the training data. Among these possibilities, we want to select the model that shows lowest generalization error rate. The process of selecting a model with the right level of complexity, which is expected to generalize well over unseen test instances, is known as **model selection**. As described in the previous section, the training error rate cannot be reliably used as the sole criterion for model selection. In the following, we present three generic approaches to estimate the generalization performance of a model that can be used for model selection. We conclude this section by presenting specific strategies for using these approaches in the context of decision tree induction.

3.5.1 Using a Validation Set

Note that we can always estimate the generalization error rate of a model by using "out-of-sample" estimates, i.e. by evaluating the model on a separate **validation set** that is not used for training the model. The error rate on the validation set, termed as the validation error rate, is a better indicator of generalization performance than the training error rate, since the validation set has not been used for training the model. The validation error rate can be used for model selection as follows.

Given a training set $D.train$, we can partition $D.train$ into two smaller subsets, $D.tr$ and $D.val$, such that $D.tr$ is used for training while $D.val$ is used as the validation set. For example, two-thirds of $D.train$ can be reserved as $D.tr$ for training, while the remaining one-third is used as $D.val$ for computing validation error rate. For any choice of classification model m that is trained on $D.tr$, we can estimate its validation error rate on $D.val$, $err_{val}(m)$. The model that shows the lowest value of $err_{val}(m)$ can then be selected as the preferred choice of model.

The use of validation set provides a generic approach for model selection. However, one limitation of this approach is that it is sensitive to the sizes of $D.tr$ and $D.val$, obtained by partitioning $D.train$. If the size of $D.tr$ is too small, it may result in the learning of a poor classification model with sub-standard performance, since a smaller training set will be less representative

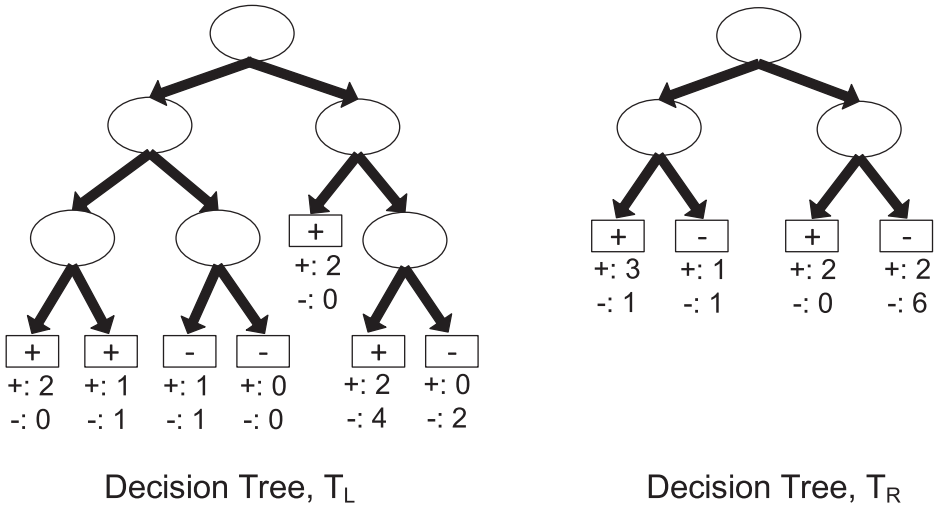


Figure 3.29. Class distribution of validation data for the two decision trees shown in Figure 3.30.

of the overall data. On the other hand, if the size of $D.val$ is too small, the validation error rate might not be reliable for selecting models, as it would be computed over a small number of instances.

Example 3.8. [Validation Error] In the following example, we illustrate one possible approach for using a validation set in decision tree induction. Figure 3.29 shows the predicted labels at the leaf nodes of the decision trees generated in Figure 3.30. The counts given beneath the leaf nodes represent the proportion of data instances in the validation set that reach each of the nodes. Based on the predicted labels of the nodes, the validation error rate for the left tree is $err_{val}(T_L) = 6/16 = 0.375$, while the validation error rate for the right tree is $err_{val}(T_R) = 4/16 = 0.25$. Based on their validation error rates, the right tree is preferred over the left one. ■

3.5.2 Incorporating Model Complexity

Since the chance for model overfitting increases as the model becomes more complex, a model selection approach should not only consider the training error rate but also the model complexity. This strategy is inspired by a well-known principle known as **Occam's razor** or the **principle of parsimony**, which suggests that given two models with the same errors, the simpler model is preferred over the more complex model. A generic approach to account

for model complexity while estimating generalization performance is formally described as follows.

Given a training set $D.train$, let us consider learning a classification model m that belongs to a certain class of models, $\mathcal{M}$. For example, if $\mathcal{M}$ represents the set of all possible decision trees, then m can correspond to a specific decision tree learned from the training set. We are interested in estimating the generalization error rate of m, $gen.error(m)$. As discussed previously, the training error rate of m, $train.error(m, D.train)$, can under-estimate $gen.error(m)$ when the model complexity is high. Hence, we represent $gen.error(m)$ as a function of not just the training error rate but also the model complexity of $\mathcal{M}$, $complexity(\mathcal{M})$, as follows:

$$gen.error(m) = train.error(m, D.train) + \alpha \times complexity(\mathcal{M}), \qquad (3.11)$$

where α is a hyper-parameter that strikes a balance between minimizing training error and reducing model complexity. A higher value of α gives more emphasis to the model complexity in the estimation of generalization performance. To choose the right value of α, we can make use of the validation set in a similar way as described in 3.5.1. For example, we can iterate through a range of values of α and for every possible value, we can learn a model on a subset of the training set, $D.tr$, and compute its validation error rate on a separate subset, $D.val$. We can then select the value of α that provides the lowest validation error rate.

Equation 3.11 provides one possible approach for incorporating model complexity into the estimate of generalization performance. This approach is at the heart of a number of techniques for estimating generalization performance, such as the structural risk minimization principle, the Akaike's Information Criterion (AIC), and the Bayesian Information Criterion (BIC). The structural risk minimization principle serves as the building block for learning support vector machines, which will be discussed later in Chapter 6. For more details on AIC and BIC, see the Bibliographic Notes.

In the following, we present two different approaches for estimating the complexity of a model, $complexity(\mathcal{M})$. While the former is specific to decision trees, the latter is more generic and can be used with any class of models.

Estimating the Complexity of Decision Trees

In the context of decision trees, the complexity of a decision tree can be estimated as the ratio of the number of leaf nodes to the number of training instances. Let k be the number of leaf nodes and N_{train} be the number of

training instances. The complexity of a decision tree can then be described as k/N_{train}. This reflects the intuition that for a larger training size, we can learn a decision tree with larger number of leaf nodes without it becoming overly complex. The generalization error rate of a decision tree T can then be computed using Equation 3.11 as follows:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}},$$

where $err(T)$ is the training error of the decision tree and Ω is a hyper-parameter that makes a trade-off between reducing training error and minimizing model complexity, similar to the use of α in Equation 3.11. Ω can be viewed as the relative cost of adding a leaf node relative to incurring a training error. In the literature on decision tree induction, the above approach for estimating generalization error rate is also termed as the **pessimistic error estimate**. It is called pessimistic as it assumes the generalization error rate to be worse than the training error rate (by adding a penalty term for model complexity). On the other hand, simply using the training error rate as an estimate of the generalization error rate is called the **optimistic error estimate** or the **resubstitution estimate**.

Example 3.9. [Generalization Error Estimates] Consider the two binary decision trees, T_L and T_R, shown in Figure 3.30. Both trees are generated from the same training data and T_L is generated by expanding three leaf nodes of T_R. The counts shown in the leaf nodes of the trees represent the class

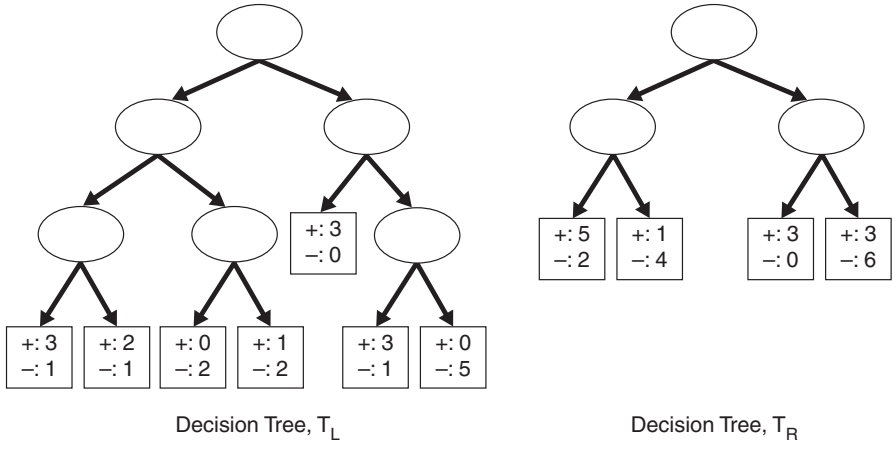


Figure 3.30. Example of two decision trees generated from the same training data.

distribution of the training instances. If each leaf node is labeled according to the majority class of training instances that reach the node, the training error rate for the left tree will be $err(T_L) = 4/24 = 0.167$, while the training error rate for the right tree will be $err(T_R) = 6/24 = 0.25$. Based on their training error rates alone, T_L would preferred over T_R, even though T_L is more complex (contains larger number of leaf nodes) than T_R.

Now, assume that the cost associated with each leaf node is $\Omega = 0.5$. Then, the generalization error estimate for T_L will be

$$err_{gen}(T_L) = \frac{4}{24} + 0.5 \times \frac{7}{24} = \frac{7.5}{24} = 0.3125$$

and the generalization error estimate for T_R will be

$$err_{gen}(T_R) = \frac{6}{24} + 0.5 \times \frac{4}{24} = \frac{8}{24} = 0.3333.$$

Since T_L has a lower generalization error rate, it will still be preferred over T_R. Note that $\Omega = 0.5$ implies that a node should always be expanded into its two child nodes if it improves the prediction of at least one training instance, since expanding a node is less costly than misclassifying a training instance. On the other hand, if $\Omega = 1$, then the generalization error rate for T_L is $err_{gen}(T_L) = 11/24 = 0.458$ and for T_R is $err_{gen}(T_R) = 10/24 = 0.417$. In this case, T_R will be preferred over T_L because it has a lower generalization error rate. This example illustrates that different choices of Ω can change our preference of decision trees based on their generalization error estimates. However, for a given choice of Ω, the pessimistic error estimate provides an approach for modeling the generalization performance on unseen test instances. The value of Ω can be selected with the help of a validation set. ■

Minimum Description Length Principle

Another way to incorporate model complexity is based on an information-theoretic approach known as the minimum description length or MDL principle. To illustrate this approach, consider the example shown in Figure 3.31. In this example, both person `A` and person `B` are given a set of instances with known attribute values $\mathbf{x}$. Assume person `A` knows the class label y for every instance, while person `B` has no such information. `A` would like to share the class information with `B` by sending a message containing the labels. The message would contain $\Theta(N)$ bits of information, where N is the number of instances.

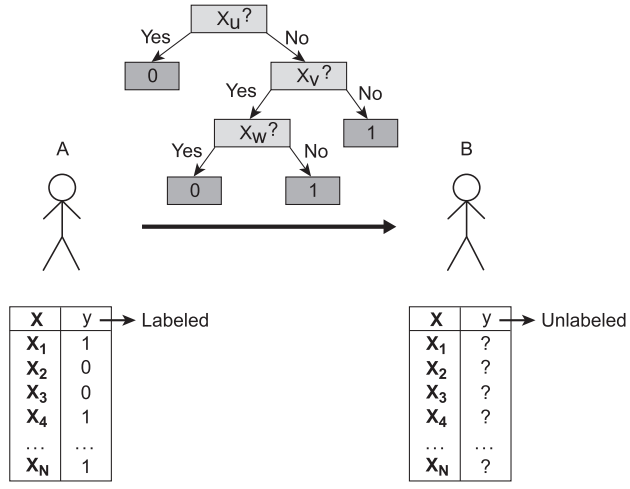


Figure 3.31. An illustration of the minimum description length principle.

Alternatively, instead of sending the class labels explicitly, `A` can build a classification model from the instances and transmit it to `B`. `B` can then apply the model to determine the class labels of the instances. If the model is 100% accurate, then the cost for transmission is equal to the number of bits required to encode the model. Otherwise, `A` must also transmit information about which instances are misclassified by the model so that `B` can reproduce the same class labels. Thus, the overall transmission cost, which is equal to the total description length of the message, is

$$Cost(model, data) = Cost(data|model) + \alpha \times Cost(model), \tag{3.12}$$

where the first term on the right-hand side is the number of bits needed to encode the misclassified instances, while the second term is the number of bits required to encode the model. There is also a hyper-parameter α that trades-off the relative costs of the misclassified instances and the model. Notice the familiarity between this equation and the generic equation for generalization error rate presented in Equation 3.11. A good model must have a total description length less than the number of bits required to encode the entire sequence of class labels. Furthermore, given two competing models, the model with lower total description length is preferred. An example showing how to compute the total description length of a decision tree is given in Exercise 11 on page 209.

3.5.3 Estimating Statistical Bounds

Instead of using Equation 3.11 to estimate the generalization error rate of a model, an alternative way is to apply a statistical correction to the training error rate of the model that is indicative of its model complexity. This can be done if the probability distribution of training error is available or can be assumed. For example, the number of errors committed by a leaf node in a decision tree can be assumed to follow a binomial distribution. We can thus compute an upper bound limit to the observed training error rate that can be used for model selection, as illustrated in the following example.

Example 3.10. [Statistical Bounds on Training Error] Consider the left-most branch of the binary decision trees shown in Figure 3.30. Observe that the left-most leaf node of T_R has been expanded into two child nodes in T_L. Before splitting, the training error rate of the node is $2/7 = 0.286$. By approximating a binomial distribution with a normal distribution, the following upper bound of the training error rate e can be derived:

$$e_{upper}(N, e, \alpha) = \frac{e + \frac{z^2_{\alpha/2}}{2N} + z_{\alpha/2}\sqrt{\frac{e(1-e)}{N} + \frac{z^2_{\alpha/2}}{4N^2}}}{1 + \frac{z^2_{\alpha/2}}{N}}, \tag{3.13}$$

where α is the confidence level, $z_{\alpha/2}$ is the standardized value from a standard normal distribution, and N is the total number of training instances used to compute e. By replacing $\alpha = 25\%$, $N = 7$, and $e = 2/7$, the upper bound for the error rate is $e_{upper}(7, 2/7, 0.25) = 0.503$, which corresponds to $7 \times 0.503 = 3.521$ errors. If we expand the node into its child nodes as shown in T_L, the training error rates for the child nodes are $1/4 = 0.250$ and $1/3 = 0.333$, respectively. Using Equation (3.13), the upper bounds of these error rates are $e_{upper}(4, 1/4, 0.25) = 0.537$ and $e_{upper}(3, 1/3, 0.25) = 0.650$, respectively. The overall training error of the child nodes is $4 \times 0.537 + 3 \times 0.650 = 4.098$, which is larger than the estimated error for the corresponding node in T_R, suggesting that it should not be split. ■

3.5.4 Model Selection for Decision Trees

Building on the generic approaches presented above, we present two commonly used model selection strategies for decision tree induction.

Prepruning (Early Stopping Rule) In this approach, the tree-growing algorithm is halted before generating a fully grown tree that perfectly fits

the entire training data. To do this, a more restrictive stopping condition must be used; e.g., stop expanding a leaf node when the observed gain in the generalization error estimate falls below a certain threshold. This estimate of the generalization error rate can be computed using any of the approaches presented in the preceding three subsections, e.g., by using pessimistic error estimates, by using validation error estimates, or by using statistical bounds. The advantage of prepruning is that it avoids the computations associated with generating overly complex subtrees that overfit the training data. However, one major drawback of this method is that, even if no significant gain is obtained using one of the existing splitting criterion, subsequent splitting may result in better subtrees. Such subtrees would not be reached if prepruning is used because of the greedy nature of decision tree induction.

Post-pruning In this approach, the decision tree is initially grown to its maximum size. This is followed by a tree-pruning step, which proceeds to trim the fully grown tree in a bottom-up fashion. Trimming can be done by replacing a subtree with (1) a new leaf node whose class label is determined from the majority class of instances affiliated with the subtree (approach known as **subtree replacement**), or (2) the most frequently used branch of the subtree (approach known as **subtree raising**). The tree-pruning step terminates when no further improvement in the generalization error estimate is observed beyond a certain threshold. Again, the estimates of generalization error rate can be computed using any of the approaches presented in the previous three subsections. Post-pruning tends to give better results than prepruning because it makes pruning decisions based on a fully grown tree, unlike prepruning, which can suffer from premature termination of the tree-growing process. However, for post-pruning, the additional computations needed to grow the full tree may be wasted when the subtree is pruned.

Figure 3.32 illustrates the simplified decision tree model for the web robot detection example given in Section 3.3.5. Notice that the subtree rooted at `depth` $= 1$ has been replaced by one of its branches corresponding to `breadth` $<= 7$, `width` > 3, and `MultiP` $= 1$, using subtree raising. On the other hand, the subtree corresponding to `depth` > 1 and `MultiAgent` $= 0$ has been replaced by a leaf node assigned to class 0, using subtree replacement. The subtree for `depth` > 1 and `MultiAgent` $= 1$ remains intact.

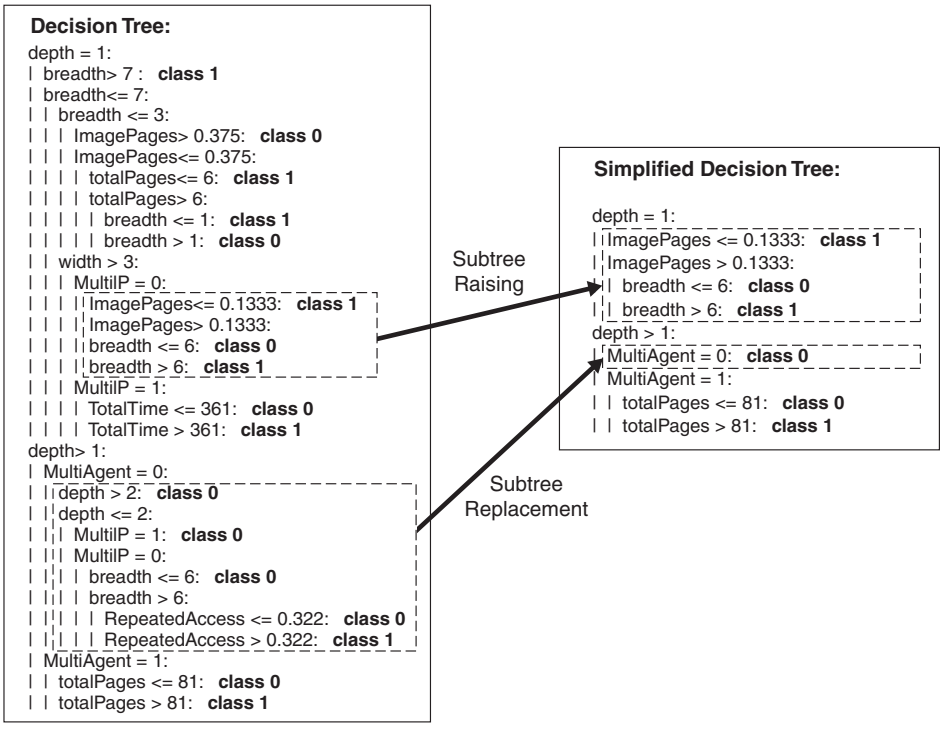


Figure 3.32. Post-pruning of the decision tree for web robot detection.

3.6 Model Evaluation

The previous section discussed several approaches for model selection that can be used to learn a classification model from a training set $D.train$. Here we discuss methods for estimating its generalization performance, i.e. its performance on unseen instances outside of $D.train$. This process is known as **model evaluation**.

Note that model selection approaches discussed in Section 3.5 also compute an estimate of the generalization performance using the training set $D.train$. However, these estimates are *biased* indicators of the performance on unseen instances, since they were used to guide the selection of classification model. For example, if we use the validation error rate for model selection (as described in Section 3.5.1), the resulting model would be deliberately chosen to minimize the errors on the validation set. The validation error rate may thus under-estimate the true generalization error rate, and hence cannot be reliably used for model evaluation.

A correct approach for model evaluation would be to assess the performance of a learned model on a labeled test set has not been used at any stage of model selection. This can be achieved by partitioning the entire set of labeled instances D, into two disjoint subsets, $D.train$, which is used for model selection and $D.test$, which is used for computing the test error rate, err_{test}. In the following, we present two different approaches for partitioning D into $D.train$ and $D.test$, and computing the test error rate, err_{test}.

3.6.1 Holdout Method

The most basic technique for partitioning a labeled data set is the holdout method, where the labeled set D is randomly partitioned into two disjoint sets, called the training set $D.train$ and the test set $D.test$. A classification model is then induced from $D.train$ using the model selection approaches presented in Section 3.5, and its error rate on $D.test$, err_{test}, is used as an estimate of the generalization error rate. The proportion of data reserved for training and for testing is typically at the discretion of the analysts, e.g., two-thirds for training and one-third for testing.

Similar to the trade-off faced while partitioning $D.train$ into $D.tr$ and $D.val$ in Section 3.5.1, choosing the right fraction of labeled data to be used for training and testing is non-trivial. If the size of $D.train$ is small, the learned classification model may be improperly learned using an insufficient number of training instances, resulting in a biased estimation of generalization performance. On the other hand, if the size of $D.test$ is small, err_{test} may be less reliable as it would be computed over a small number of test instances. Moreover, err_{test} can have a high variance as we change the random partitioning of D into $D.train$ and $D.test$.

The holdout method can be repeated several times to obtain a distribution of the test error rates, an approach known as **random subsampling** or repeated holdout method. This method produces a distribution of the error rates that can be used to understand the variance of err_{test}.

3.6.2 Cross-Validation

Cross-validation is a widely-used model evaluation method that aims to make effective use of all labeled instances in D for both training and testing. To illustrate this method, suppose that we are given a labeled set that we have randomly partitioned into three equal-sized subsets, S_1, S_2, and S_3, as shown in Figure 3.33. For the first run, we train a model using subsets S_2 and S_3 (shown as empty blocks) and test the model on subset S_1. The test error rate

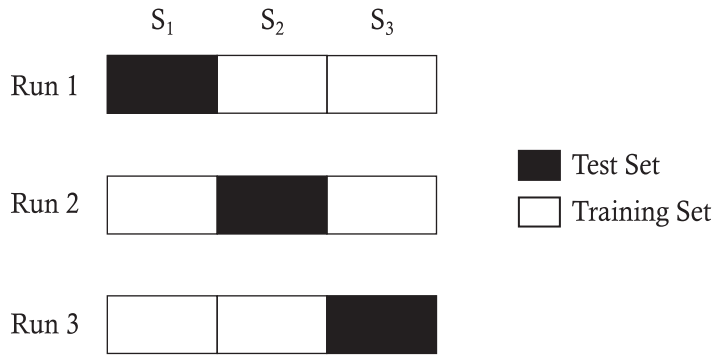


Figure 3.33. Example demonstrating the technique of 3-fold cross-validation.

on S_1, denoted as $err(S_1)$, is thus computed in the first run. Similarly, for the second run, we use S_1 and S_3 as the training set and S_2 as the test set, to compute the test error rate, $err(S_2)$, on S_2. Finally, we use S_1 and S_3 for training in the third run, while S_3 is used for testing, thus resulting in the test error rate $err(S_3)$ for S_3. The overall test error rate is obtained by summing up the number of errors committed in each test subset across all runs and dividing it by the total number of instances. This approach is called three-fold cross-validation.

The k-fold cross-validation method generalizes this approach by segmenting the labeled data D (of size N) into k equal-sized partitions (or folds). During the i^{th} run, one of the partitions of D is chosen as $D.test(i)$ for testing, while the rest of the partitions are used as $D.train(i)$ for training. A model $m(i)$ is learned using $D.train(i)$ and applied on $D.test(i)$ to obtain the sum of test errors, $err_{sum}(i)$. This procedure is repeated k times. The total test error rate, err_{test}, is then computed as

$$err_{test} = \frac{\sum_{i=1}^{k} err_{sum}(i)}{N}. \tag{3.14}$$

Every instance in the data is thus used for testing exactly once and for training exactly $(k-1)$ times. Also, every run uses $(k-1)/k$ fraction of the data for training and $1/k$ fraction for testing.

The right choice of k in k-fold cross-validation depends on a number of characteristics of the problem. A small value of k will result in a smaller training set at every run, which will result in a larger estimate of generalization error rate than what is expected of a model trained over the entire labeled set. On the other hand, a high value of k results in a larger training set at

every run, which reduces the bias in the estimate of generalization error rate. In the extreme case, when $k = N$, every run uses exactly one data instance for testing and the remainder of the data for testing. This special case of k-fold cross-validation is called the **leave-one-out** approach. This approach has the advantage of utilizing as much data as possible for training. However, leave-one-out can produce quite misleading results in some special scenarios, as illustrated in Exercise 12. Furthermore, leave-one-out can be computationally expensive for large data sets as the cross-validation procedure needs to be repeated N times. For most practical applications, the choice of k between 5 and 10 provides a reasonable approach for estimating the generalization error rate, because each fold is able to make use of 80% to 90% of the labeled data for training.

The k-fold cross-validation method, as described above, produces a single estimate of the generalization error rate, without providing any information about the variance of the estimate. To obtain this information, we can run k-fold cross-validation for every possible partitioning of the data into k partitions, and obtain a distribution of test error rates computed for every such partitioning. The average test error rate across all possible partitionings serves as a more robust estimate of generalization error rate. This approach of estimating the generalization error rate and its variance is known as the **complete cross-validation** approach. Even though such an estimate is quite robust, it is usually too expensive to consider all possible partitionings of a large data set into k partitions. A more practical solution is to repeat the cross-validation approach multiple times, using a different random partitioning of the data into k partitions at every time, and use the average test error rate as the estimate of generalization error rate. Note that since there is only one possible partitioning for the leave-one-out approach, it is not possible to estimate the variance of generalization error rate, which is another limitation of this method.

The k-fold cross-validation does not guarantee that the fraction of positive and negative instances in every partition of the data is equal to the fraction observed in the overall data. A simple solution to this problem is to perform a stratified sampling of the positive and negative instances into k partitions, an approach called **stratified cross-validation**.

In k-fold cross-validation, a different model is learned at every run and the performance of every one of the k models on their respective test folds is then aggregated to compute the overall test error rate, err_{test}. Hence, err_{test} does not reflect the generalization error rate of any of the k models. Instead, it reflects the *expected* generalization error rate of the *model selection approach*, when applied on a training set of the same size as one of the training

folds ($N(k-1)/k$). This is different than the err_{test} computed in the holdout method, which exactly corresponds to the specific model learned over $D.train$. Hence, although effectively utilizing every data instance in D for training and testing, the err_{test} computed in the cross-validation method does not represent the performance of a single model learned over a specific $D.train$.

Nonetheless, in practice, err_{test} is typically used as an estimate of the generalization error of a model built on D. One motivation for this is that when the size of the training folds is closer to the size of the overall data (when k is large), then err_{test} resembles the expected performance of a model learned over a data set of the same size as D. For example, when k is 10, every training fold is 90% of the overall data. The err_{test} then should approach the expected performance of a model learned over 90% of the overall data, which will be close to the expected performance of a model learned over D.

3.7 Presence of Hyper-parameters

Hyper-parameters are parameters of learning algorithms that need to be determined before learning the classification model. For instance, consider the hyper-parameter α that appeared in Equation 3.11, which is repeated here for convenience. This equation was used for estimating the generalization error for a model selection approach that used an explicit representations of model complexity. (See Section 3.5.2.)

$$gen.error(m) = train.error(m, D.train) + \alpha \times complexity(\mathcal{M})$$

For other examples of hyper-parameters, see Chapter 6.

Unlike regular model parameters, such as the test conditions in the internal nodes of a decision tree, hyper-parameters such as α do not appear in the final classification model that is used to classify unlabeled instances. However, the values of hyper-parameters need to be determined during model selection—a process known as **hyper-parameter selection**—and must be taken into account during model evaluation. Fortunately, both tasks can be effectively accomplished via slight modifications of the cross-validation approach described in the previous section.

3.7.1 Hyper-parameter Selection

In Section 3.5.2, a validation set was used to select α and this approach is generally applicable for hyper-parameter section. Let p be the hyper-parameter that needs to be selected from a finite range of values, $\mathcal{P} = \{p_1, p_2, \ldots p_n\}$.

Partition $D.train$ into $D.tr$ and $D.val$. For every choice of hyper-parameter value p_i, we can learn a model m_i on $D.tr$, and apply this model on $D.val$ to obtain the validation error rate $err_{val}(p_i)$. Let p^* be the hyper-parameter value that provides the lowest validation error rate. We can then use the model m^* corresponding to p^* as the final choice of classification model.

The above approach, although useful, uses only a subset of the data, $D.train$, for training and a subset, $D.val$, for validation. The framework of cross-validation, presented in Section 3.6.2, addresses both of those issues, albeit in the context of model evaluation. Here we indicate how to use a cross-validation approach for hyper-parameter selection. To illustrate this approach, let us partition $D.train$ into three folds as shown in Figure 3.34. At every run, one of the folds is used as $D.val$ for validation, and the remaining two folds are used as $D.tr$ for learning a model, for every choice of hyper-parameter value p_i. The overall validation error rate corresponding to each p_i is computed by summing the errors across all the three folds. We then select the hyper-parameter value p^* that provides the lowest validation error rate, and use it to learn a model m^* on the entire training set $D.train$.

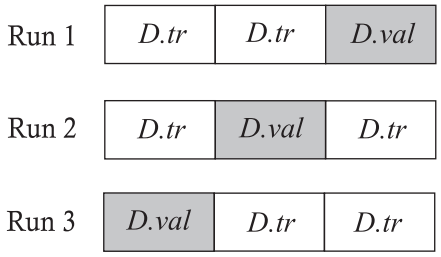


Figure 3.34. Example demonstrating the 3-fold cross-validation framework for hyper-parameter selection using $D.train$.

Algorithm 3.2 generalizes the above approach using a k-fold cross-validation framework for hyper-parameter selection. At the i^{th} run of cross-validation, the data in the i^{th} fold is used as $D.val(i)$ for validation (Step 4), while the remainder of the data in $D.train$ is used as $D.tr(i)$ for training (Step 5). Then for every choice of hyper-parameter value p_i, a model is learned on $D.tr(i)$ (Step 7), which is applied on $D.val(i)$ to compute its validation error (Step 8). This is used to compute the validation error rate corresponding to models learning using p_i over all the folds (Step 11). The hyper-parameter value p^* that provides the lowest validation error rate (Step 12) is now used to learn the final model m^* on the entire training set $D.train$ (Step 13). Hence, at the

Algorithm 3.2 Procedure `model-select`(k, $\mathcal{P}$, $D.train$)

1: $N_{train} = |D.train|$ {Size of $D.train$.}
2: Divide $D.train$ into k partitions, $D.train_1$ to $D.train_k$.
3: **for** each run $i = 1$ to k **do**
4: $D.val(i) = D.train_i$. {Partition used for validation.}
5: $D.tr(i) = D.train \setminus D.train_i$. {Partitions used for training.}
6: **for** each parameter $p \in \mathcal{P}$ **do**
7: $m =$ `model-train`(p, $D.tr(i)$). {Train model}
8: $err_{sum}(p, i) =$ `model-test`(m, $D.val(i)$). {Sum of validation errors.}
9: **end for**
10: **end for**
11: $err_{val}(p) = \sum_i^k err_{sum}(p, i)/N_{train}$. {Compute validation error rate.}
12: $p^* = \text{argmin}_p\ err_{val}(p)$. {Select best hyper-parameter value.}
13: $m^* =$ `model-train`(p^*, $D.train$). {Learn final model on $D.train$}
14: return (p^*, m^*).

end of this algorithm, we obtain the best choice of the hyper-parameter value as well as the final classification model (Step 14), both of which are obtained by making an effective use of every data instance in $D.train$.

3.7.2 Nested Cross-Validation

The approach of the previous section provides a way to effectively use all the instances in $D.train$ to learn a classification model when hyper-parameter selection is required. This approach can be applied over the entire data set D to learn the final classification model. However, applying Algorithm 3.2 on D would only return the final classification model m^* but not an estimate of its generalization performance, err_{test}. Recall that the validation error rates used in Algorithm 3.2 cannot be used as estimates of generalization performance, since they are used to guide the selection of the final model m^*. However, to compute err_{test}, we can again use a cross-validation framework for evaluating the performance on the entire data set D, as described originally in Section 3.6.2. In this approach, D is partitioned into $D.train$ (for training) and $D.test$ (for testing) at every run of cross-validation. When hyper-parameters are involved, we can use Algorithm 3.2 to train a model using $D.train$ at every run, thus "internally" using cross-validation for model selection. This approach is called **nested cross-validation** or double cross-validation. Algorithm 3.3 describes the complete approach for estimating err_{test} using nested cross-validation in the presence of hyper-parameters.

As an illustration of this approach, see Figure 3.35 where the labeled set D is partitioned into $D.train$ and $D.test$, using a 3-fold cross-validation method.

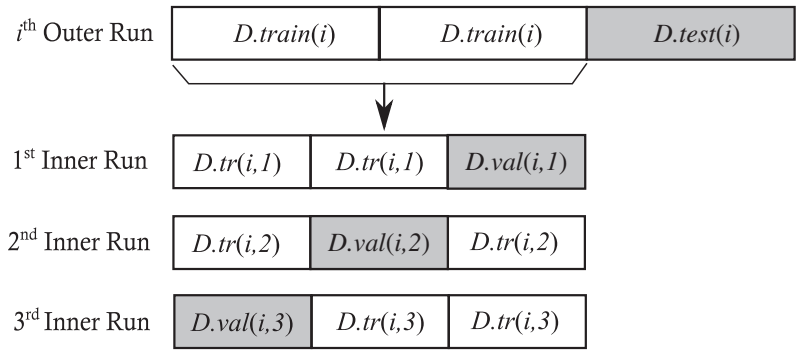


Figure 3.35. Example demonstrating 3-fold nested cross-validation for computing err_{test}.

At the i^{th} run of this method, one of the folds is used as the test set, $D.test(i)$, while the remaining two folds are used as the training set, $D.train(i)$. This is represented in Figure 3.35 as the i^{th} "outer" run. In order to select a model using $D.train(i)$, we again use an "inner" 3-fold cross-validation framework that partitions $D.train(i)$ into $D.tr$ and $D.val$ at every one of the three inner runs (iterations). As described in Section 3.7, we can use the inner cross-validation framework to select the best hyper-parameter value $p^*(i)$ as well as its resulting classification model $m^*(i)$ learned over $D.train(i)$. We can then apply $m^*(i)$ on $D.test(i)$ to obtain the test error at the i^{th} outer run. By repeating this process for every outer run, we can compute the average test error rate, err_{test}, over the entire labeled set D. Note that in the above approach, the inner cross-validation framework is being used for model selection while the outer cross-validation framework is being used for model evaluation.

Algorithm 3.3 The nested cross-validation approach for computing err_{test}.

1: Divide D into k partitions, D_1 to D_k.
2: **for** each outer run $i = 1$ to k **do**
3: $D.test(i) = D_i$. {Partition used for testing.}
4: $D.train(i) = D \setminus D_i$. {Partitions used for model selection.}
5: $(p^*(i), m^*(i)) =$ `model-select`$(k, \mathcal{P}, D.train(i))$. {Inner cross-validation.}
6: $err_{sum}(i) =$ `model-test`$(m^*(i), D.test(i))$. {Sum of test errors.}
7: **end for**
8: $err_{test} = \sum_i^k \ err_{sum}(i)/N$. {Compute test error rate.}

3.8 Pitfalls of Model Selection and Evaluation

Model selection and evaluation, when used effectively, serve as excellent tools for learning classification models and assessing their generalization performance. However, when using them effectively in practical settings, there are several pitfalls that can result in improper and often misleading conclusions. Some of these pitfalls are simple to understand and easy to avoid, while others are quite subtle in nature and difficult to catch. In the following, we present two of these pitfalls and discuss best practices to avoid them.

3.8.1 Overlap between Training and Test Sets

One of the basic requirements of a *clean* model selection and evaluation setup is that the data used for model selection ($D.train$) must be kept separate from the data used for model evaluation ($D.test$). If there is any overlap between the two, the test error rate err_{test} computed over $D.test$ cannot be considered representative of the performance on *unseen* instances. Comparing the effectiveness of classification models using err_{test} can then be quite misleading, as an overly complex model can show an inaccurately low value of err_{test} due to model overfitting (see Exercise 13 at the end of this chapter).

To illustrate the importance of ensuring no overlap between $D.train$ and $D.test$, consider a labeled data set where all the attributes are irrelevant, i.e. they have no relationship with the class labels. Using such attributes, we should expect no classification model to perform better than random guessing. However, if the test set involves even a small number of data instances that were used for training, there is a possibility for an overly complex model to show better performance than random, even though the attributes are completely irrelevant. As we will see later in Chapter 10, this scenario can actually be used as a criterion to detect overfitting due to improper setup of experiment. If a model shows better performance than a random classifier even when the attributes are irrelevant, it is an indication of a potential feedback between the training and test sets.

3.8.2 Use of Validation Error as Generalization Error

The validation error rate err_{val} serves an important role during model selection, as it provides "out-of-sample" error estimates of models on $D.val$, which is not used for training the models. Hence, err_{val} serves as a better metric than the training error rate for selecting models and hyper-parameter values, as described in Sections 3.5.1 and 3.7, respectively. However, once the

validation set has been used for selecting a classification model m^*, err_{val} no longer reflects the performance of m^* on *unseen* instances.

To realize the pitfall in using validation error rate as an estimate of generalization performance, consider the problem of selecting a hyper-parameter value p from a range of values $\mathcal{P}$, using a validation set $D.val$. If the number of possible values in $\mathcal{P}$ is quite large and the size of $D.val$ is small, it is possible to select a hyper-parameter value p^* that shows favorable performance on $D.val$ just by random chance. Notice the similarity of this problem with the multiple comparisons problem discussed in Section 3.4.1. Even though the classification model m^* learned using p^* would show a low validation error rate, it would lack generalizability on unseen test instances.

The correct approach for estimating the generalization error rate of a model m^* is to use an independently chosen test set $D.test$ that hasn't been used in any way to influence the selection of m^*. As a rule of thumb, the test set should never be examined during model selection, to ensure the absence of any form of overfitting. If the insights gained from any portion of a labeled data set help in improving the classification model even in an indirect way, then that portion of data must be discarded during testing.

3.9 Model Comparison*

One difficulty when comparing the performance of different classification models is whether the observed difference in their performance is statistically significant. For example, consider a pair of classification models, M_A and M_B. Suppose M_A achieves 85% accuracy when evaluated on a test set containing 30 instances, while M_B achieves 75% accuracy on a different test set containing 5000 instances. Based on this information, is M_A a better model than M_B? This example raises two key questions regarding the statistical significance of a performance metric:

1. Although M_A has a higher accuracy than M_B, it was tested on a smaller test set. How much confidence do we have that the accuracy for M_A is actually 85%?

2. Is it possible to explain the difference in accuracies between M_A and M_B as a result of variations in the composition of their test sets?

The first question relates to the issue of estimating the confidence interval of model accuracy. The second question relates to the issue of testing the statistical significance of the observed deviation. These issues are investigated in the remainder of this section.

3.9.1 Estimating the Confidence Interval for Accuracy

To determine its confidence interval, we need to establish the probability distribution for sample accuracy. This section describes an approach for deriving the confidence interval by modeling the classification task as a binomial random experiment. The following describes characteristics of such an experiment:

1. The random experiment consists of N independent trials, where each trial has two possible outcomes: success or failure.

2. The probability of success, p, in each trial is constant.

An example of a binomial experiment is counting the number of heads that turn up when a coin is flipped N times. If X is the number of successes observed in N trials, then the probability that X takes a particular value is given by a binomial distribution with mean Np and variance $Np(1-p)$:

$$P(X = v) = \binom{N}{v} p^v (1-p)^{N-v}.$$

For example, if the coin is fair ($p = 0.5$) and is flipped fifty times, then the probability that the head shows up 20 times is

$$P(X = 20) = \binom{50}{20} 0.5^{20}(1-0.5)^{30} = 0.0419.$$

If the experiment is repeated many times, then the average number of heads expected to show up is $50 \times 0.5 = 25$, while its variance is $50 \times 0.5 \times 0.5 = 12.5$.

The task of predicting the class labels of test instances can also be considered as a binomial experiment. Given a test set that contains N instances, let X be the number of instances correctly predicted by a model and p be the true accuracy of the model. If the prediction task is modeled as a binomial experiment, then X has a binomial distribution with mean Np and variance $Np(1-p)$. It can be shown that the empirical accuracy, $acc = X/N$, also has a binomial distribution with mean p and variance $p(1-p)/N$ (see Exercise 15). The binomial distribution can be approximated by a normal distribution when N is sufficiently large. Based on the normal distribution, the confidence interval for acc can be derived as follows:

$$P\left(-Z_{\alpha/2} \leq \frac{acc - p}{\sqrt{p(1-p)/N}} \leq Z_{1-\alpha/2} \right) = 1 - \alpha, \tag{3.15}$$

where $Z_{\alpha/2}$ and $Z_{1-\alpha/2}$ are the upper and lower bounds obtained from a standard normal distribution at confidence level $(1-\alpha)$. Since a standard normal distribution is symmetric around $Z = 0$, it follows that $Z_{\alpha/2} = Z_{1-\alpha/2}$. Rearranging this inequality leads to the following confidence interval for p:

$$\frac{2 \times N \times acc + Z_{\alpha/2}^2 \pm Z_{\alpha/2}\sqrt{Z_{\alpha/2}^2 + 4Nacc - 4Nacc^2}}{2(N + Z_{\alpha/2}^2)}. \tag{3.16}$$

The following table shows the values of $Z_{\alpha/2}$ at different confidence levels:

$1-\alpha$	0.99	0.98	0.95	0.9	0.8	0.7	0.5
$Z_{\alpha/2}$	2.58	2.33	1.96	1.65	1.28	1.04	0.67

Example 3.11. [Confidence Interval for Accuracy] Consider a model that has an accuracy of 80% when evaluated on 100 test instances. What is the confidence interval for its true accuracy at a 95% confidence level? The confidence level of 95% corresponds to $Z_{\alpha/2} = 1.96$ according to the table given above. Inserting this term into Equation 3.16 yields a confidence interval between 71.1% and 86.7%. The following table shows the confidence interval when the number of instances, N, increases:

N	20	50	100	500	1000	5000
Confidence Interval	0.584 − 0.919	0.670 − 0.888	0.711 − 0.867	0.763 − 0.833	0.774 − 0.824	0.789 − 0.811

Note that the confidence interval becomes tighter when N increases. ■

3.9.2 Comparing the Performance of Two Models

Consider a pair of models, M_1 and M_2, which are evaluated on two independent test sets, D_1 and D_2. Let n_1 denote the number of instances in D_1 and n_2 denote the number of instances in D_2. In addition, suppose the error rate for M_1 on D_1 is e_1 and the error rate for M_2 on D_2 is e_2. Our goal is to test whether the observed difference between e_1 and e_2 is statistically significant.

Assuming that n_1 and n_2 are sufficiently large, the error rates e_1 and e_2 can be approximated using normal distributions. If the observed difference in the error rate is denoted as $d = e_1 - e_2$, then d is also normally distributed with mean d_t, its true difference, and variance, σ_d^2. The variance of d can be

computed as follows:

$$\sigma_d^2 \simeq \widehat{\sigma}_d^2 = \frac{e_1(1-e_1)}{n_1} + \frac{e_2(1-e_2)}{n_2}, \tag{3.17}$$

where $e_1(1-e_1)/n_1$ and $e_2(1-e_2)/n_2$ are the variances of the error rates. Finally, at the $(1-\alpha)\%$ confidence level, it can be shown that the confidence interval for the true difference d_t is given by the following equation:

$$d_t = d \pm z_{\alpha/2}\widehat{\sigma}_d. \tag{3.18}$$

Example 3.12. [Significance Testing] Consider the problem described at the beginning of this section. Model M_A has an error rate of $e_1 = 0.15$ when applied to $N_1 = 30$ test instances, while model M_B has an error rate of $e_2 = 0.25$ when applied to $N_2 = 5000$ test instances. The observed difference in their error rates is $d = |0.15 - 0.25| = 0.1$. In this example, we are performing a two-sided test to check whether $d_t = 0$ or $d_t \neq 0$. The estimated variance of the observed difference in error rates can be computed as follows:

$$\widehat{\sigma}_d^2 = \frac{0.15(1-0.15)}{30} + \frac{0.25(1-0.25)}{5000} = 0.0043$$

or $\widehat{\sigma}_d = 0.0655$. Inserting this value into Equation 3.18, we obtain the following confidence interval for d_t at 95% confidence level:

$$d_t = 0.1 \pm 1.96 \times 0.0655 = 0.1 \pm 0.128.$$

As the interval spans the value zero, we can conclude that the observed difference is not statistically significant at a 95% confidence level. ∎

At what confidence level can we reject the hypothesis that $d_t = 0$? To do this, we need to determine the value of $Z_{\alpha/2}$ such that the confidence interval for d_t does not span the value zero. We can reverse the preceding computation and look for the value $Z_{\alpha/2}$ such that $d > Z_{\alpha/2}\widehat{\sigma}_d$. Replacing the values of d and $\widehat{\sigma}_d$ gives $Z_{\alpha/2} < 1.527$. This value first occurs when $(1-\alpha) \lesssim 0.936$ (for a two-sided test). The result suggests that the null hypothesis can be rejected at confidence level of 93.6% or lower.

3.10 Bibliographic Notes

Early classification systems were developed to organize various collections of objects, from living organisms to inanimate ones. Examples abound, from Aristotle's cataloguing of species to the Dewey Decimal and Library of Congress

classification systems for books. Such a task typically requires considerable human efforts, both to identify properties of the objects to be classified and to organize them into well distinguished categories.

With the development of statistics and computing, automated classification has been a subject of intensive research. The study of classification in classical statistics is sometimes known as **discriminant analysis**, where the objective is to predict the group membership of an object based on its corresponding features. A well-known classical method is Fisher's linear discriminant analysis [142], which seeks to find a linear projection of the data that produces the best separation between objects from different classes.

Many pattern recognition problems also require the discrimination of objects from different classes. Examples include speech recognition, handwritten character identification, and image classification. Readers who are interested in the application of classification techniques for pattern recognition may refer to the survey articles by Jain et al. [150] and Kulkarni et al. [157] or classic pattern recognition books by Bishop [125], Duda et al. [137], and Fukunaga [143]. The subject of classification is also a major research topic in neural networks, statistical learning, and machine learning. An in-depth treatment on the topic of classification from the statistical and machine learning perspectives can be found in the books by Bishop [126], Cherkassky and Mulier [132], Hastie et al. [148], Michie et al. [162], Murphy [167], and Mitchell [165]. Recent years have also seen the release of many publicly available software packages for classification, which can be embedded in programming languages such as Java (Weka [147]) and Python (scikit-learn [174]).

An overview of decision tree induction algorithms can be found in the survey articles by Buntine [129], Moret [166], Murthy [168], and Safavian et al. [179]. Examples of some well-known decision tree algorithms include CART [127], ID3 [175], C4.5 [177], and CHAID [153]. Both ID3 and C4.5 employ the entropy measure as their splitting function. An in-depth discussion of the C4.5 decision tree algorithm is given by Quinlan [177]. The CART algorithm was developed by Breiman et al. [127] and uses the Gini index as its splitting function. CHAID [153] uses the statistical χ^2 test to determine the best split during the tree-growing process.

The decision tree algorithm presented in this chapter assumes that the splitting condition at each internal node contains only one attribute. An oblique decision tree can use multiple attributes to form the attribute test condition in a single node [149, 187]. Breiman et al. [127] provide an option for using linear combinations of attributes in their CART implementation. Other approaches for inducing oblique decision trees were proposed by Heath et al. [149], Murthy et al. [169], Cantú-Paz and Kamath [130], and Utgoff

and Brodley [187]. Although an oblique decision tree helps to improve the expressiveness of the model representation, the tree induction process becomes computationally challenging. Another way to improve the expressiveness of a decision tree without using oblique decision trees is to apply a method known as **constructive induction** [161]. This method simplifies the task of learning complex splitting functions by creating compound features from the original data.

Besides the top-down approach, other strategies for growing a decision tree include the bottom-up approach by Landeweerd et al. [159] and Pattipati and Alexandridis [173], as well as the bidirectional approach by Kim and Landgrebe [154]. Schuermann and Doster [181] and Wang and Suen [193] proposed using a **soft splitting criterion** to address the data fragmentation problem. In this approach, each instance is assigned to different branches of the decision tree with different probabilities.

Model overfitting is an important issue that must be addressed to ensure that a decision tree classifier performs equally well on previously unlabeled data instances. The model overfitting problem has been investigated by many authors including Breiman et al. [127], Schaffer [180], Mingers [164], and Jensen and Cohen [151]. While the presence of noise is often regarded as one of the primary reasons for overfitting [164, 170], Jensen and Cohen [151] viewed overfitting as an artifact of failure to compensate for the multiple comparisons problem.

Bishop [126] and Hastie et al. [148] provide an excellent discussion of model overfitting, relating it to a well-known framework of theoretical analysis, known as bias-variance decomposition [146]. In this framework, the prediction of a learning algorithm is considered to be a function of the training set, which varies as the training set is changed. The generalization error of a model is then described in terms of its *bias* (the error of the average prediction obtained using different training sets), its *variance* (how different are the predictions obtained using different training sets), and *noise* (the irreducible error inherent to the problem). An underfit model is considered to have high bias but low variance, while an overfit model is considered to have low bias but high variance. Although the bias-variance decomposition was originally proposed for regression problems (where the target attribute is a continuous variable), a unified analysis that is applicable for classification has been proposed by Domingos [136]. The bias variance decomposition will be discussed in more detail while introducing ensemble learning methods in Chapter 6.

Various learning principles, such as the Probably Approximately Correct (PAC) learning framework [188], have been developed to provide a theoretical framework for explaining the generalization performance of learning

algorithms. In the field of statistics, a number of performance estimation methods have been proposed that make a trade-off between the goodness of fit of a model and the model complexity. Most noteworthy among them are the Akaike's Information Criterion [120] and the Bayesian Information Criterion [182]. They both apply corrective terms to the training error rate of a model, so as to penalize more complex models. Another widely-used approach for measuring the complexity of any general model is the Vapnik-Chervonenkis (VC) Dimension [190]. The VC dimension of a class of functions C is defined as the maximum number of points that can be *shattered* (every point can be distinguished from the rest) by functions belonging to C, for any possible configuration of points. The VC dimension lays the foundation of the structural risk minimization principle [189], which is extensively used in many learning algorithms, e.g., support vector machines, which will be discussed in detail in Chapter 6.

The Occam's razor principle is often attributed to the philosopher William of Occam. Domingos [135] cautioned against the pitfall of misinterpreting Occam's razor as comparing models with similar training errors, instead of generalization errors. A survey on decision tree-pruning methods to avoid overfitting is given by Breslow and Aha [128] and Esposito et al. [141]. Some of the typical pruning methods include reduced error pruning [176], pessimistic error pruning [176], minimum error pruning [171], critical value pruning [163], cost-complexity pruning [127], and error-based pruning [177]. Quinlan and Rivest proposed using the minimum description length principle for decision tree pruning in [178].

The discussions in this chapter on the significance of cross-validation error estimates is inspired from Chapter 7 in Hastie et al. [148]. It is also an excellent resource for understanding "the right and wrong ways to do cross-validation", which is similar to the discussion on pitfalls in Section 3.8 of this chapter. A comprehensive discussion of some of the common pitfalls in using cross-validation for model selection and evaluation is provided in Krstajic et al. [156].

The original cross-validation method was proposed independently by Allen [121], Stone [184], and Geisser [145] for model assessment (evaluation). Even though cross-validation can be used for model selection [194], its usage for model selection is quite different than when it is used for model evaluation, as emphasized by Stone [184]. Over the years, the distinction between the two usages has often been ignored, resulting in incorrect findings. One of the common mistakes while using cross-validation is to perform pre-processing operations (e.g., hyper-parameter tuning or feature selection) using the entire data set and not "within" the training fold of every cross-validation run.

Ambroise et al., using a number of gene expression studies as examples, [124] provide an extensive discussion of the *selection bias* that arises when feature selection is performed outside cross-validation. Useful guidelines for evaluating models on microarray data have also been provided by Allison et al. [122].

The use of the cross-validation protocol for hyper-parameter tuning has been described in detail by Dudoit and van der Laan [138]. This approach has been called "grid-search cross-validation." The correct approach in using cross-validation for both hyper-parameter selection and model evaluation, as discussed in Section 3.7 of this chapter, is extensively covered by Varma and Simon [191]. This combined approach has been referred to as "nested cross-validation" or "double cross-validation" in the existing literature. Recently, Tibshirani and Tibshirani [185] have proposed a new approach for hyper-parameter selection and model evaluation. Tsamardinos et al. [186] compared this approach to nested cross-validation. The experiments they performed found that, on average, both approaches provide conservative estimates of model performance with the Tibshirani and Tibshirani approach being more computationally efficient.

Kohavi [155] has performed an extensive empirical study to compare the performance metrics obtained using different estimation methods such as random subsampling and k-fold cross-validation. Their results suggest that the best estimation method is ten-fold, stratified cross-validation.

An alternative approach for model evaluation is the bootstrap method, which was presented by Efron in 1979 [139]. In this method, training instances are sampled *with replacement* from the labeled set, i.e., an instance previously selected to be part of the training set is equally likely to be drawn again. If the original data has N instances, it can be shown that, on average, a bootstrap sample of size N contains about 63.2% of the instances in the original data. Instances that are not included in the bootstrap sample become part of the test set. The bootstrap procedure for obtaining training and test sets is repeated b times, resulting in a different error rate on the test set, $err(i)$, at the i^{th} run. To obtain the overall error rate, err_{boot}, the **.632 bootstrap** approach combines $err(i)$ with the error rate obtained from a training set containing all the labeled examples, err_s, as follows:

$$err_{boot} = \frac{1}{b}\sum_{i=1}^{b}(0.632 \times err(i) + 0.368 \times err_s). \tag{3.19}$$

Efron and Tibshirani [140] provided a theoretical and empirical comparison between cross-validation and a bootstrap method known as the 632+ rule.

While the .632 bootstrap method presented above provides a robust estimate of the generalization performance with low variance in its estimate, it may produce misleading results for highly complex models in certain conditions, as demonstrated by Kohavi [155]. This is because the overall error rate is not truly an out-of-sample error estimate as it depends on the training error rate, err_s, which can be quite small if there is overfitting.

Current techniques such as C4.5 require that the entire training data set fit into main memory. There has been considerable effort to develop parallel and scalable versions of decision tree induction algorithms. Some of the proposed algorithms include SLIQ by Mehta et al. [160], SPRINT by Shafer et al. [183], CMP by Wang and Zaniolo [192], CLOUDS by Alsabti et al. [123], RainForest by Gehrke et al. [144], and ScalParC by Joshi et al. [152]. A survey of parallel algorithms for classification and other data mining tasks is given in [158]. More recently, there has been extensive research to implement large-scale classifiers on the compute unified device architecture (CUDA) [131, 134] and MapReduce [133, 172] platforms.

Bibliography

[120] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.

[121] D. M. Allen. The relationship between variable selection and data agumentation and a method for prediction. *Technometrics*, 16(1):125–127, 1974.

[122] D. B. Allison, X. Cui, G. P. Page, and M. Sabripour. Microarray data analysis: from disarray to consolidation and consensus. *Nature reviews genetics*, 7(1):55–65, 2006.

[123] K. Alsabti, S. Ranka, and V. Singh. CLOUDS: A Decision Tree Classifier for Large Datasets. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 2–8, New York, NY, August 1998.

[124] C. Ambroise and G. J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the national academy of sciences*, 99 (10):6562–6566, 2002.

[125] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K., 1995.

[126] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[127] L. Breiman, J. H. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[128] L. A. Breslow and D. W. Aha. Simplifying Decision Trees: A Survey. *Knowledge Engineering Review*, 12(1):1–40, 1997.

[129] W. Buntine. Learning classification trees. In *Artificial Intelligence Frontiers in Statistics*, pages 182–201. Chapman & Hall, London, 1993.

[130] E. Cantú-Paz and C. Kamath. Using evolutionary algorithms to induce oblique decision trees. In *Proc. of the Genetic and Evolutionary Computation Conf.*, pages 1053–1060, San Francisco, CA, 2000.

[131] B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th International Conference on Machine Learning*, pages 104–111, 2008.

[132] V. Cherkassky and F. M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley, 2nd edition, 2007.

[133] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.

[134] A. Cotter, N. Srebro, and J. Keshet. A GPU-tailored Approach for Training Kernelized SVMs. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 805–813, San Diego, California, USA, 2011.

[135] P. Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.

[136] P. Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*, pages 231–238, 2000.

[137] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.

[138] S. Dudoit and M. J. van der Laan. Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. *Statistical Methodology*, 2(2):131–154, 2005.

[139] B. Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in Statistics*, pages 569–593. Springer, 1992.

[140] B. Efron and R. Tibshirani. Cross-validation and the Bootstrap: Estimating the Error Rate of a Prediction Rule. Technical report, Stanford University, 1995.

[141] F. Esposito, D. Malerba, and G. Semeraro. A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19 (5):476–491, May 1997.

[142] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[143] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1990.

[144] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest—A Framework for Fast Decision Tree Construction of Large Datasets. *Data Mining and Knowledge Discovery*, 4(2/3):127–162, 2000.

[145] S. Geisser. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328, 1975.

[146] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[147] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.

[148] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[149] D. Heath, S. Kasif, and S. Salzberg. Induction of Oblique Decision Trees. In *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence*, pages 1002–1007, Chambery, France, August 1993.

[150] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical Pattern Recognition: A Review. *IEEE Tran. Patt. Anal. and Mach. Intellig.*, 22(1):4–37, 2000.

[151] D. Jensen and P. R. Cohen. Multiple Comparisons in Induction Algorithms. *Machine Learning*, 38(3):309–338, March 2000.

[152] M. V. Joshi, G. Karypis, and V. Kumar. ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets. In *Proc. of 12th Intl. Parallel Processing Symp. (IPPS/SPDP)*, pages 573–579, Orlando, FL, April 1998.

[153] G. V. Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 29:119–127, 1980.

[154] B. Kim and D. Landgrebe. Hierarchical decision classifiers in high-dimensional and large class data. *IEEE Trans. on Geoscience and Remote Sensing*, 29(4):518–528, 1991.

[155] R. Kohavi. A Study on Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pages 1137–1145, Montreal, Canada, August 1995.

[156] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of cheminformatics*, 6(1):1, 2014.

[157] S. R. Kulkarni, G. Lugosi, and S. S. Venkatesh. Learning Pattern Classification—A Survey. *IEEE Tran. Inf. Theory*, 44(6):2178–2206, 1998.

[158] V. Kumar, M. V. Joshi, E.-H. Han, P. N. Tan, and M. Steinbach. High Performance Data Mining. In *High Performance Computing for Computational Science (VECPAR 2002)*, pages 111–125. Springer, 2002.

[159] G. Landeweerd, T. Timmers, E. Gersema, M. Bins, and M. Halic. Binary tree versus single level tree classification of white blood cells. *Pattern Recognition*, 16:571–577, 1983.

[160] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A Fast Scalable Classifier for Data Mining. In *Proc. of the 5th Intl. Conf. on Extending Database Technology*, pages 18–32, Avignon, France, March 1996.

[161] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–116, 1983.

[162] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, 1994.

[163] J. Mingers. Expert Systems—Rule Induction with Statistical Data. *J Operational Research Society*, 38:39–47, 1987.

[164] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[165] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.

[166] B. M. E. Moret. Decision Trees and Diagrams. *Computing Surveys*, 14(4):593–623, 1982.

[167] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[168] S. K. Murthy. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[169] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *J of Artificial Intelligence Research*, 2:1–33, 1994.

[170] T. Niblett. Constructing decision trees in noisy domains. In *Proc. of the 2nd European Working Session on Learning*, pages 67–78, Bled, Yugoslavia, May 1987.

[171] T. Niblett and I. Bratko. Learning Decision Rules in Noisy Domains. In *Research and Development in Expert Systems III*, Cambridge, 1986. Cambridge University Press.

[172] I. Palit and C. K. Reddy. Scalable and parallel boosting with mapreduce. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1904–1916, 2012.

[173] K. R. Pattipati and M. G. Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(4):872–887, 1990.

[174] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[175] J. R. Quinlan. Discovering rules by induction from large collection of examples. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*. Edinburgh University Press, Edinburgh, UK, 1979.

[176] J. R. Quinlan. Simplifying Decision Trees. *Intl. J. Man-Machine Studies*, 27:221–234, 1987.

[177] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, San Mateo, CA, 1993.

[178] J. R. Quinlan and R. L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation*, 80(3):227–248, 1989.

[179] S. R. Safavian and D. Landgrebe. A Survey of Decision Tree Classifier Methodology. *IEEE Trans. Systems, Man and Cybernetics*, 22:660–674, May/June 1998.

[180] C. Schaffer. Overfitting avoidence as bias. *Machine Learning*, 10:153–178, 1993.

[181] J. Schuermann and W. Doster. A decision-theoretic approach in hierarchical classifier design. *Pattern Recognition*, 17:359–369, 1984.

[182] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.

[183] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proc. of the 22nd VLDB Conf.*, pages 544–555, Bombay, India, September 1996.

[184] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147, 1974.

[185] R. J. Tibshirani and R. Tibshirani. A bias correction for the minimum error rate in cross-validation. *The Annals of Applied Statistics*, pages 822–829, 2009.

[186] I. Tsamardinos, A. Rakhshani, and V. Lagani. Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. In *Hellenic Conference on Artificial Intelligence*, pages 1–14. Springer, 2014.

[187] P. E. Utgoff and C. E. Brodley. An incremental method for finding multivariate splits for decision trees. In *Proc. of the 7th Intl. Conf. on Machine Learning*, pages 58–65, Austin, TX, June 1990.

[188] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[189] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[190] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer, 2015.

[191] S. Varma and R. Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1):1, 2006.

[192] H. Wang and C. Zaniolo. CMP: A Fast Decision Tree Classifier Using Multivariate Predictions. In *Proc. of the 16th Intl. Conf. on Data Engineering*, pages 449–460, San Diego, CA, March 2000.

Table 3.5. Data set for Exercise 3.

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

[193] Q. R. Wang and C. Y. Suen. Large tree classifier with heuristic search and global training. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9(1):91–102, 1987.

[194] Y. Zhang and Y. Yang. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112, 2015.

3.11 Exercises

1. What is the stopping criterion for Hunt's decision tree?

2. Draw a full decision tree for the odd parity function, where only when the count of True is odd is the class label True, of four Boolean attributes A, B, C, and D. Is it possible to simplify the tree?

3. Consider the training examples shown in Table 3.5 for a binary classification problem.

   (a) Compute the Gini index for the overall collection of training examples.

   (b) Compute the Gini index for the `Customer ID` attribute.

   (c) Compute the Gini index for the `Gender` attribute.

Table 3.6. Data set for Exercise 4.

Instance	a_1	a_2	a_3	Target Class
1	T	T	1.0	+
2	T	T	6.0	+
3	T	F	5.0	−
4	F	F	4.0	+
5	F	T	7.0	−
6	F	T	3.0	−
7	F	F	8.0	−
8	T	F	7.0	+
9	F	T	5.0	−

(d) Compute the Gini index for the `Car Type` attribute using multiway split.

(e) Compute the Gini index for the `Shirt Size` attribute using multiway split.

(f) Which attribute is better, `Gender`, `Car Type`, or `Shirt Size`?

(g) Explain why `Customer ID` should not be used as the attribute test condition even though it has the lowest Gini.

4. Consider the training examples shown in Table 3.6 for a binary classification problem.

   (a) What is the entropy of this collection of training examples with respect to the class attribute?

   (b) What are the information gains of a_1 and a_2 relative to these training examples?

   (c) For a_3, which is a continuous attribute, compute the information gain for every possible split.

   (d) What is the best split (among a_1, a_2, and a_3) according to the information gain?

   (e) What is the best split (between a_1 and a_2) according to the misclassification error rate?

   (f) What is the best split (between a_1 and a_2) according to the Gini index?

5. Show that the entropy of a node never increases after splitting it into smaller successor nodes.

6. Consider the following data set for a binary class problem.

A	B	Class Label
T	F	+
T	T	+
T	T	+
T	F	−
T	T	+
F	F	−
F	F	−
F	F	−
T	T	−
T	F	−

(a) Calculate the information gain when splitting on A and B. Which attribute would the decision tree induction algorithm choose?

(b) Calculate the gain in the Gini index when splitting on A and B. Which attribute would the decision tree induction algorithm choose?

(c) Figure 3.11 shows that entropy and the Gini index are both monotonically increasing on the range [0, 0.5] and they are both monotonically decreasing on the range [0.5, 1]. Is it possible that information gain and the gain in the Gini index favor different attributes? Explain.

7. Consider splitting a parent node P into two child nodes, C_1 and C_2, using some attribute test condition. The composition of labeled training instances at every node is summarized in the Table below.

	P	C_1	C_2
Class 0	7	3	4
Class 1	3	0	3

(a) Calculate the Gini index and misclassification error rate of the parent node P.

(b) Calculate the weighted Gini index of the child nodes. Would you consider this attribute test condition if Gini is used as the impurity measure?

(c) Calculate the weighted misclassification rate of the child nodes. Would you consider this attribute test condition if misclassification rate is used as the impurity measure?

8. Consider the following set of training examples.

X	Y	Z	No. of Class C1 Examples	No. of Class C2 Examples
0	0	0	5	40
0	0	1	0	15
0	1	0	10	5
0	1	1	45	0
1	0	0	10	5
1	0	1	25	0
1	1	0	5	20
1	1	1	0	15

(a) Compute a two-level decision tree using the greedy approach described in this chapter. Use the classification error rate as the criterion for splitting. What is the overall error rate of the induced tree?

(b) Repeat part (a) using X as the first splitting attribute and then choose the best remaining attribute for splitting at each of the two successor nodes. What is the error rate of the induced tree?

(c) Compare the results of parts (a) and (b). Comment on the suitability of the greedy heuristic used for splitting attribute selection.

9. The following table summarizes a data set with three attributes A, B, C and two class labels $+$, $-$. Build a two-level decision tree.

A	B	C	Number of Instances	
			$+$	$-$
T	T	T	5	0
F	T	T	0	20
T	F	T	20	0
F	F	T	0	5
T	T	F	0	0
F	T	F	25	0
T	F	F	0	0
F	F	F	0	25

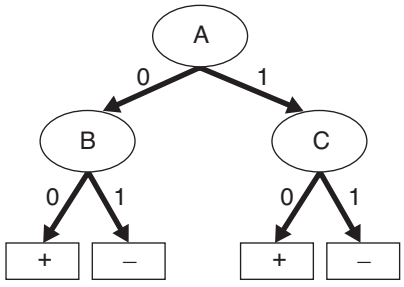
(a) According to the classification error rate, which attribute would be chosen as the first splitting attribute? For each attribute, show the contingency table and the gains in classification error rate.

(b) Repeat for the two children of the root node.

(c) How many instances are misclassified by the resulting decision tree?

(d) Repeat parts (a), (b), and (c) using C as the splitting attribute.

(e) Use the results in parts (c) and (d) to conclude about the greedy nature of the decision tree induction algorithm.



Training:

Instance	A	B	C	Class
1	0	0	0	+
2	0	0	1	+
3	0	1	0	+
4	0	1	1	–
5	1	0	0	+
6	1	0	0	+
7	1	1	0	–
8	1	0	1	+
9	1	1	0	–
10	1	1	0	–

Validation:

Instance	A	B	C	Class
11	0	0	0	+
12	0	1	1	+
13	1	1	0	+
14	1	0	1	–
15	1	0	0	+

Figure 3.36. Decision tree and data sets for Exercise 10.

10. Consider the decision tree shown in Figure 3.36.

    (a) Compute the generalization error rate of the tree using the optimistic approach.

    (b) Compute the generalization error rate of the tree using the pessimistic approach. (For simplicity, use the strategy of adding a factor of 0.5 to each leaf node.)

    (c) Compute the generalization error rate of the tree using the validation set shown above. This approach is known as **reduced error pruning**.

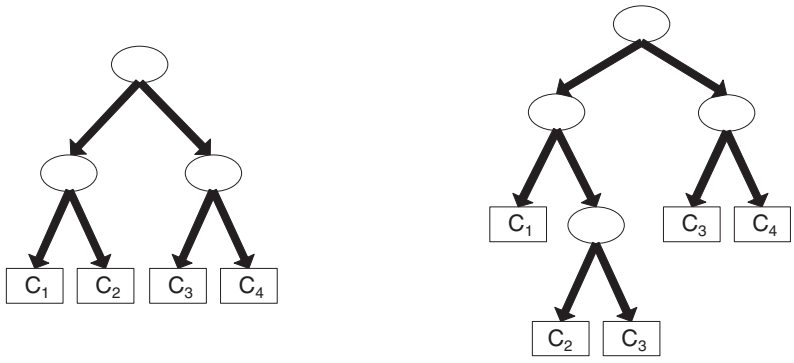
11. Consider the decision tree shown in Figure 3.37. Assume they are generated from a data set that contains 32 binary attributes and 4 classes, C_1, C_2, C_3 and C_4.

    Compute the total description length of each decision tree according to the following formulation of the minimum description length principle.

    - The total description length of a tree is given by

      $$Cost(tree, data) = Cost(tree) + Cost(data|tree).$$

    - Each internal node of the tree is encoded by the ID of the splitting attribute. If there are m attributes, the cost of encoding each attribute is $\log_2 m$ bits.



(a) Decision tree with 5 errors

(b) Decision tree with 4 errors

Figure 3.37. Decision trees for Exercise 11.

- Each leaf is encoded using the ID of the class it is associated with. If there are k classes, the cost of encoding a class is $\log_2 k$ bits.
- $Cost(tree)$ is the cost of encoding all the nodes in the tree. To simplify the computation, you can assume that the total cost of the tree is obtained by adding up the costs of encoding each internal node and each leaf node.
- $Cost(data|tree)$ is encoded using the classification errors the tree commits on the training set. Each error is encoded by $\log_2 n$ bits, where n is the total number of training instances.

Which decision tree is better, according to the MDL principle?

12. This exercise, inspired by the discussions in [155], highlights one of the known limitations of the leave-one-out model evaluation procedure. Let us consider a data set containing 50 positive and 50 negative instances, where the attributes are purely random and contain no information about the class labels. Hence, the generalization error rate of any classification model learned over this data is expected to be 0.5. Let us consider a classifier that assigns the majority class label of training instances (ties resolved by using the positive label as the default class) to any test instance, irrespective of its attribute values. We can call this approach as the *majority inducer* classifier. Determine the error rate of this classifier using the following methods.

    (a) Leave-one-out.

    (b) 2-fold stratified cross-validation, where the proportion of class labels at every fold is kept same as that of the overall data.

    (c) From the results above, which method provides a more reliable evaluation of the classifier's generalization error rate?

Table 3.7. Comparing the test accuracy of decision trees T_{10} and T_{100}.

	Accuracy	
Data Set	T_{10}	T_{100}
A	0.86	0.97
B	0.84	0.77

13. Consider a labeled data set containing 100 data instances, which is randomly partitioned into two sets A and B, each containing 50 instances. We use A as the training set to learn two decision trees, T_{10} with 10 leaf nodes and T_{100} with 100 leaf nodes. The accuracies of the two decision trees on data sets A and B are shown in Table 3.7.

    (a) Based on the accuracies shown in Table 3.7, which classification model would you expect to have better performance on unseen instances?

    (b) Now, you tested T_{10} and T_{100} on the entire data set $(A + B)$ and found that the classification accuracy of T_{10} on data set $(A+B)$ is 0.85, whereas the classification accuracy of T_{100} on the data set $(A + B)$ is 0.87. Based on this new information and your observations from Table 3.7, which classification model would you finally choose for classification?

14. Consider the following approach for testing whether a classifier A beats another classifier B. Let N be the size of a given data set, p_A be the accuracy of classifier A, p_B be the accuracy of classifier B, and $p = (p_A + p_B)/2$ be the average accuracy for both classifiers. To test whether classifier A is significantly better than B, the following Z-statistic is used:

$$Z = \frac{p_A - p_B}{\sqrt{\frac{2p(1-p)}{N}}}.$$

    Classifier A is assumed to be better than classifier B if Z > 1.96.

    Table 3.8 compares the accuracies of three different classifiers, decision tree classifiers, naïve Bayes classifiers, and support vector machines, on various data sets. (The latter two classifiers are described in Chapter 6.)

    Summarize the performance of the classifiers given in Table 3.8 using the following 3×3 table:

win-loss-draw	Decision tree	Naïve Bayes	Support vector machine
Decision tree	0 - 0 - 23		
Naïve Bayes		0 - 0 - 23	
Support vector machine			0 - 0 - 23

Table 3.8. Comparing the accuracy of various classification methods.

Data Set	Size (N)	Decision Tree (%)	naïve Bayes (%)	Support vector machine (%)
Anneal	898	92.09	79.62	87.19
Australia	690	85.51	76.81	84.78
Auto	205	81.95	58.05	70.73
Breast	699	95.14	95.99	96.42
Cleve	303	76.24	83.50	84.49
Credit	690	85.80	77.54	85.07
Diabetes	768	72.40	75.91	76.82
German	1000	70.90	74.70	74.40
Glass	214	67.29	48.59	59.81
Heart	270	80.00	84.07	83.70
Hepatitis	155	81.94	83.23	87.10
Horse	368	85.33	78.80	82.61
Ionosphere	351	89.17	82.34	88.89
Iris	150	94.67	95.33	96.00
Labor	57	78.95	94.74	92.98
Led7	3200	73.34	73.16	73.56
Lymphography	148	77.03	83.11	86.49
Pima	768	74.35	76.04	76.95
Sonar	208	78.85	69.71	76.92
Tic-tac-toe	958	83.72	70.04	98.33
Vehicle	846	71.04	45.04	74.94
Wine	178	94.38	96.63	98.88
Zoo	101	93.07	93.07	96.04

Each cell in the table contains the number of wins, losses, and draws when comparing the classifier in a given row to the classifier in a given column.

15. Let X be a binomial random variable with mean Np and variance $Np(1-p)$. Show that the ratio X/N also has a binomial distribution with mean p and variance $p(1-p)/N$.

16. What is *underfitting* in the decision tree?

17. How can continuous attributes be handled? How can splitting points be chosen for binary splits and for multiway splits?