DATA MINING TECHNIQUES

(DMCA304)

(MCA)

ACHARYA NAGARJUNA UNIVERSITY

CENTRE FOR DISTANCE EDUCATION

NAGARJUNA NAGAR,

GUNTUR

ANDHRA PRADESH

CONTENTS

UNIT - I

1. Introduction

Structure

1.1 Introduction

1.2 The Nature of Data Sets

1.3 Types of Structure: Models and Patterns

1.4 Data Mining Tasks

1.5 Components of Data Mining Algorithms

- 1.5.1 Score Functions
- 1.5.2 Optimization and Search Methods
- 1.5.3 Data Management Strategies

1.6 Data Mining: Dredging, Snooping, and Fishing

Objective

After going through this lesson, you should be able to:

- Understand the Nature of Data Sets;
- Discuss Models and Patterns;
- Discuss Components of Data Mining Algorithms;

1.1 Introduction

Progress in digital data acquisition and storage technology has resulted in the growth of huge databases. This has occurred in all areas of human endeavor, from the mundane (such as supermarket transaction data, credit card usage records, telephone call details, and government statistics) to the more exotic (such as images of astronomical bodies, molecular databases, and medical records). Little wonder, then, that interest has grown in the possibility of tapping these data, of extracting from them information that might be of value to the owner of the database. The discipline concerned with this task has become known as data mining.

NOTES

Defining a scientific discipline is always a controversial task; researchers often disagree about the precise range and limits of their field of study. Bearing this in mind, and accepting that others might disagree about the details, we shall adopt as our working definition of data mining:

Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.

The relationships and summaries derived through a data mining exercise are often referred to as models or patterns. Examples include linear equations, rules, clusters, graphs, tree structures, and recurrent patterns in time series.

The definition above refers to "observational data," as opposed to "experimental data." Data mining typically deals with data that have already been collected for some purpose other than the data mining analysis (for example, they may have been collected in order to maintain an up -to-date record of all the transactions in a bank). This means that the objectives of the data mining exercise play no role in the data collection strategy. This is one way in which data mining differs from much of statistics, in which data are often collected by using efficient strategies to answer specific questions. For this reason, data mining is often referred to as "secondary" data analysis.

The process of seeking relationships within a data set of seeking accurate, convenient, and useful summary representations of some aspect of the data involves a number of steps:

- determining the nature and structure of the representation to be used;

- deciding how to quantify and compare how well different representations fit the data (that is, choosing a "score" function);

- choosing an algorithmic process to optimize the score function; and deciding what principles of data management are required to implement the algorithms efficiently.

Data mining is an interdisciplinary exercise. Statistics, database technology, machine learning, pattern recognition, artificial intelligence, and visualization, all play a role and just as it is difficult to define sharp boundaries between these disciplines, so it is difficult to define sharp boundaries between each of them and data mining. At the boundaries, one person's data mining is another's statistics, database, or machine learning problem.

1.2 The Nature of Data Sets

We begin by discussing at a high level the basic nature of data sets.

A data set is a set of measurements taken from some environment or process. In the simplest case, we have a collection of objects, and for each object we have a set of the same p measurements. In this case, we can think of the collection of the measurements on n objects as a form of n × p data matrix. The n rows represent the n objects on which measurements were taken (for example, medical patients, credit card customers, or individual objects observed in the night sky, such as stars and galaxies). Such rows may be referred to as individuals, entities, cases, objects, or records depending on the context.

The other dimension of our data matrix contains the set of p measurements made on each object. Typically we assume that the same p measurements are made on each individual although this need not be the case (for example, different medical tests could be performed on different patients). The p columns of the data matrix may be referred to as variables, features, attributes, or fields; again, the language depends on the research context. In all situations the idea is the same: these names refer to the measurement that is represented by each column.

1.3 Types of Structure: Models and Patterns

The different kinds of representations sought during a data mining exercise may be characterized in various ways. One such characterization is the distinction between a global model and a local pattern.

A model structure, as defined here, is a global summary of a data set; it makes statements about any point in the full measurement space. Geometrically, if we consider the rows of the data matrix as corresponding to p-dimensional vectors (i.e. points in p-dimensional space), the model can make a statement about any point in this space (and hence, any object). For example, it can assign a point to a cluster or predict the value of some other variable. Even when some of the measurements are missing (i.e., some of the components of the p-dimensional vector are unknown), a model can typically make some statement about the object represented by the (incomplete) vector.

A simple model might take the form Y = aX + c, where Y and X are variables a and c are parameters of the model (constants determined during the course of the data mining exercise). Here we would say that the functional form of the model is linear, since Y is a linear function of X. The conventional statistical use of the term is slightly different. In statistics, a model is linear if it is a linear function of the parameters. We will try to be clear in the text about which form of linearity we are assuming, but when we discuss the structure of a model (as we are doing here) it makes sense to consider linearity as a function of the variables of interest rather than the parameters. Thus, for example, the model structure $Y = aX^2 + bX + c$, is considered a linear model in classic statistical terminology, but the functional form of the model relating Y and X is nonlinear (it is a second-degree polynomial).

In contrast to the global nature of models, pattern structures make statements only about restricted regions of the space spanned by the variables. An example is a simple probabilistic statement of the form if $X > x_1$ then prob($Y > y_1$) = p1. This structure consists of constraints on the values of the variables X and Y, related in the form of a probabilistic rule. Alternatively we could describe the relationship as the conditional probability $p(Y > y_1|X > x_1) = p1$, which is semantically equivalent. Or we might notice that certain classes of transaction records do not show the peaks and troughs shown by the vast majority, and look more closely to see why. (This sort of exercise led one bank to discover that it had several open accounts that belonged to people who had died.)

Note that the model and pattern structures described above have parameters associated with them; a, b, c for the model and x_1, y_1 and p1 for the pattern. In general, once we have established the structural form we are interested in finding; the next step is to estimate its parameters from the available data. Once the parameters have been assigned values, we refer to a particular model, such as y = 3.2x + 2.8, as a "fitted model," or just "model" for short (and similarly for patterns). This distinction between model (or pattern) structures and the actual (fitted) model (or pattern) is quite important. The structures represent the general functional forms of the models (or patterns), with unspecified parameter values. A fitted model or pattern has specific values for its parameters.

1.4 Data Mining Tasks

It is convenient to categorize data mining into types of tasks, corresponding to different objectives for the person who is analyzing the data. The categorization below is not unique, and further division into finer tasks is possible, but it captures the types of data mining activities and previews the major types of data mining algorithms we will describe later in the text.

1. Exploratory Data Analysis (EDA):

As the name suggests, the goal here is simply to explore the data without any clear ideas of what we are looking for. Typically, EDA techniques are interactive and visual, and there are many effective graphical display methods for relatively small, low-dimensional data sets. As the dimensionality (number of variables, p) increases, it becomes much more difficult to visualize the cloud of points in p-space. For p higher than 3 or 4, projection techniques (such as principal components analysis) that produce informative low-dimensional projections of the data can be very useful. Large numbers of cases can be difficult to visualize effectively, however, and notions of scale and detail come into play: "lower resolution" data samples can be displayed or summarized at the cost of possibly missing important details.

2. Descriptive Modeling:

The goal of a descriptive model is describe all of the data (or the process generating the data). Examples of such descriptions include models for the overall probability distribution of the data (density estimation), partitioning of the p-dimensional space into groups (cluster analysis and segmentation), and models describing the relationship between variables (dependency modeling). In segmentation analysis, for example, the aim is to group together similar records, as in market segmentation of commercial databases. Here the goal is to split the records into homogeneous groups so that similar people (if the records refer to people) are put into the same group. This enables advertisers and marketers to efficiently direct their promotions to those most likely to respond. The number of groups here is chosen by the researcher; there is no "right" number. This contrasts with cluster analysis, in which the aim is to discover "natural" groups in data in scientific databases, for example.

3. Predictive Modeling: Classification and Regression:

The aim here is to build a model that will permit the value of one variable to be predicted from the known values of other variables. In classification, the variable being predicted is categorical, while in regression the variable is quantitative. The term "prediction" is used here in a general sense, and no notion of a time continuum is implied. So, for example, while we might want to predict the value of the stock market at some future date, or which horse will win a race, we might also want to determine the diagnosis of a patient, or the degree of brittleness of a weld. A large number of methods have been developed in statistics and machine learning to tackle predictive modeling problems, and work in this area has led to significant theoretical advances and improved understanding of deep issues of inference. The key distinction between prediction and description is that prediction has as its objective a unique variable (the market's value, the disease class, the brittleness, etc.); while in descriptive problems no single variable is central to the model.

4. Discovering Patterns and Rules:

The three types of tasks listed above are concerned with model building. Other data mining applications are concerned with pattern detection. One example is spotting fraudulent behavior by detecting regions of the space defining the different types of transactions where the data points significantly different from the rest. Another use is in astronomy, where detection of unusual stars or galaxies may lead to the discovery of previously unknown phenomena. Yet another is the task of finding combinations of items that occur frequently in transaction databases (e.g., grocery products that are often purchased together). This problem has been the focus of much attention in data mining and has been addressed using algorithmic techniques based on association rules.

5. Retrieval by Content:

Here the user has a pattern of interest and wishes to find similar patterns in the data set. This task is most commonly used for text and image data sets. For text, the pattern may be a set of keywords, and the user may wish to find relevant documents within a large set of possibly relevant documents (e.g., Web pages). For images, the user may have a sample image, a sketch of an image, or a description of an image, and wish to find similar images from a large set of images. In both cases the definition of similarity is critical, but so are the details of the search strategy.

1.5 Components of Data Mining Algorithms

In the preceding sections we have listed the basic categories of tasks that may be undertaken in data mining. We now turn to the question of how one actually accomplishes these tasks. We will take the view that data mining algorithms that address these tasks have four basic components:

1. Model or Pattern Structure: determining the underlying structure or functional forms that we seek from the data.

2. Score Function: judging the quality of a fitted model.

3. Optimization and Search Method: optimizing the score function and searching over different model and pattern structures.

4. Data management Strategy: handling data access efficiently during the search/optimization.

We have already discussed the distinction between model and pattern structures. In the remainder of this section we briefly discuss the other three components of a data mining algorithm.

1.5.1 Score Functions

Score functions quantify how well a model or parameter structure fits a given data set. In an ideal world the choice of score function would precisely reflect the utility (i.e., the true expected benefit) of a particular predictive model. In practice, however, it is often difficult to specify precisely the true utility of a model's predictions. Hence, simple, "generic" score functions, such as least squares and classification accuracy are commonly used.

Without some form of score function, we cannot tell whether one model is better than another or, indeed, how to choose a good set of values for the parameters of the model.

Several score functions are widely used for this purpose; these include likelihood, sum of squared errors, and misclassification rate (the latter is used in supervised classification problems). For example, the well-known squared error score function is defined as

$$\sum_{i=1}^{n} (y(i) - \overline{y}(i))^2$$

where we are predicting n "target" values y(i), 1 ≤ i ≤ n, and our predictions for each are denoted as $\bar{y}(i)$ (typically this is a function of some other "input" variable values for prediction and the parameters of the model).

1.5.2 Optimization and Search Methods

The score function is a measure of how well aspects of the data match proposed models or patterns. Usually, these models or patterns are described in terms of a structure, sometimes with unknown parameter values. The goal of optimization and search is to determine the structure and the parameter values that achieve a minimum (or maximum, depending on the context) value of the score function. The task of finding the "best" values of parameters in models is typically cast as an optimization (or estimation) problem. The task of finding interesting patterns (such as rules) from a large family of potential patterns is typically cast as a combinatorial search problem, and is often accomplished using heuristic search techniques. In linear regression, a prediction rule is usually found by minimizing a least squares score function (the sum of squared errors between the prediction from a model and the observed values of the predicted variable). Such a score function is amenable to mathematical manipulation, and the model that minimizes it can be found algebraically. In contrast, a score function such as misclassification rate in supervised classification is difficult to minimize analytically. For example, since it is intrinsically discontinuous the powerful tool of differential calculus cannot be brought to bear.

1.5.3 Data Management Strategies

The final component in any data mining algorithm is the data management strategy: the ways in which the data are stored, indexed, and accessed. Most well-known data analysis algorithms in statistics and machine learning have been developed under the assumption that all individual data points can be accessed quickly and efficiently in random -access memory (RAM). While main memory technology has improved rapidly, there have been equally rapid improvements in secondary (disk) and tertiary (tape) storage technologies, to the extent that many massive data sets still reside largely on disk or tape and will not fit in available RAM. Thus, there will probably be a price to pay for accessing massive data sets, since not all data points can be simultaneously close to the main processor.

1.6 Data Mining: Dredging, Snooping, and Fishing

An introductory chapter on data mining would not be complete without reference to the historical use of terms such as "data mining," "dredging," "snooping," and "fishing." In the1960s, as computers were increasingly applied to data analysis problems, it was noted that if you searched long enough, you could always find some model to fit a data set arbitrarily well. There are two factors contributing to this situation: the complexity of the model and the size of the set of possible models.

Clearly, if the class of models we adopt is very flexible (relative to the size of the available data set), then we will probably be able to fit the available data arbitrarily well. However, as we remarked above, the aim may be to generalize beyond the available data; a model that fits well may not be ideal for this purpose. Moreover, even if the aim is to fit the data (for example, when we wish to produce the most accurate summary of data describing a complete population) it is generally preferable to do this with a simple model. To take an extreme, a model of complexity equivalent to that of the raw data would certainly fit it perfectly, but would hardly be of interest or value.

Even with a relatively simple model structure, if we consider enough different models with this basic structure, we can eventually expect to find a good fit. For example, consider predicting a response variable, Y from a predictor variable X which is chosen from a very large set of possible variables, X_1, ..., X_p, none of which are related to Y. By virtue of random variation in the data generating process, although there are no underlying relationships between Y and any of the X variables, there will appear to be relationships in the data at hand. The search process will then find the X variable that appears to have the strongest relationship to Y. By this means, as a consequence of the large search space, an apparent pattern is found where none really exists. The situation is particularly bad when working with a small sample size n and a large number p of potential X variables. Familiar examples of this sort of problem include the spurious correlations which are popularized in the media, such as the "discovery" that over the past 30 years when the winner of the Super Bowl championship in American football is from a particular league, a leading stock market index historically goes up in the following months. Similar examples are plentiful in areas such as economics and the social sciences, fields in which data are often relatively sparse but models and theories to fit to the data are

NOTES

relatively plentiful. For instance, in economic time-series prediction, there may be a relatively short time-span of historical data available in conjunction with a large number of economic indicators (potential predictor variables). One particularly humorous example of this type of prediction was provided by Leinweber (personal communication) who achieved almost perfect prediction of annual values of the well-known Standard and Poor 500 financial index as a function of annual values from previous years for butter production, cheese production, and sheep populations in Bangladesh and the United States.

NOTES

2. Measurement and Data

Structure

Objective

After going through this lesson, you should be able to:

- Discuss different types of measurements;
- Discuss about Quality of Measurements ;
- Discuss about Quality of collections of data;

2.1Introduction

Our aim is to discover relationships that exist in the "real world," where this may be the physical world, the business world, the scientific world, or some other conceptual domain. However, in seeking such relationships, we do not go out and look at that domain firsthand. Rather, we study data describing it. So first we need to be clear about what we mean by data.

Data are collected by mapping entities in the domain of interest to symbolic representation by means of some measurement procedure, which associates the value of a variable with a given property of an entity. The relationships between objects are represented by numerical relationships between variables. These numerical representations, the data items, are stored in the data set; it is these items that are the subjects of our data mining activities.

NOTES

2.2 Types of Measurement

Measurements may be categorized in many ways. Some of the distinctions arise from the nature of the properties the measurements represent, while others arise from the use to which the measurements are put.

To illustrate, we will begin by considering how we might measure the property WEIGHT. In this discussion we will denote a property by using uppercase letters, and the variable corresponding to it (the result of the mapping to numbers induced by the measurement operation) by lowercase letters. Thus a measurement of WEIGHT yields a value of weight. For concreteness, let us imagine we have a collection of rocks.

The first thing we observe is that we can rank the rocks according to the WEIGHT property. We could do this, for example, by placing a rock on each pan of a weighing scale and seeing which way the scale tipped. On this basis, we could assign a number to each rock so that larger numbers corresponded to heavier rocks. Note that here only the ordinal properties of these numbers are relevant. The fact that one rock was assigned the number 4 and another was assigned the number 2 would not imply that the first was in any sense twice as heavy as the second. We could equally have chosen some other number, provided it was greater than 2, to represent the WEIGHT of the first rock. In general, any monotonic (order preserving) transformation of the set of numbers we assigned would provide an equally legitimate assignment. We are only concerned with the order of the rocks in terms of their WEIGHT property.

We can take the rocks example further. Suppose we find that, when we place a large rock on one pan of the weighing scale and two small rocks on the other pan, the pans balance. In some sense the WEIGHT property of the two small rocks has combined to be equal to the WEIGHT property of the large rock. It turns out (this will come as no surprise!) that we can assign numbers to the rocks in such a way that not only does the order of the numbers correspond to the order observed from the weighing scales, but the sum of the numbers assigned to the two smaller rocks equals the number assigned to the larger rock. That is, the total weight of the two smaller rocks equals the weight of the larger rock. Note that even now the assignment of numbers is not unique. Suppose we had assigned the numbers 2 and 3 to the smaller rocks, and the number 5 to the larger rock. This assignment satisfies the ordinal and additive property requirements, but so too would the

NOTES

assignment of 4, 6, and 10 respectively. There is still some freedom in how we define the variable weight corresponding to the WEIGHT property.

The point of this example is that our numerical representation reflects the empirical properties of the system we are studying. Relationships between rocks in terms of their WEIGHT property correspond to relationships between values of the measured variable weight. This representation is useful because it allows us to make inferences about the physical system by studying the numerical system. Without juggling sacks of rocks, we can see which sack contains the largest rock, which sack has the heaviest rocks on average, and so on.

The rocks example involves two empirical relationships: the order of the rocks, in terms of how they tip the scales, and their concatenation property the way two rocks together balance a third. Other empirical systems might involve less than or more than two relationships. The order relationship is very common; typically, if an empirical system has only one relationship, it is an order relationship. Examples of the order relationship are provided by the SEVERITY property in medicine and the PREFERENCE property in psychology.

Of course, not even an order relationship holds with some properties, for example, the properties HAIR COLOR, RELIGION, and RESIDENCE OF PROGRAMMER; do not have a natural order. Numbers can still be used to represent "values" of the properties, (blond = 1, black = 2, brown = 3, and so on), but the only empirical relationship being represented is that the colors are different (and so are represented by different numbers). It is perhaps even more obvious here that the particular set of numbers assigned is not unique. Any set in which different numbers correspond to different values of the property will do.

Given that the assignment of numbers is not unique, we must find some way to restrict this freedom or else problems might arise if different researchers use different assignments. The solution is to adopt some convention. For the rocks example, we would adopt a basic "value" of the property WEIGHT, corresponding to a basic value of the variable weight, and defined measured values in terms of how many copies of the basic value are required to balance them. Examples of such basic values for the WEIGHT/weight system are the gram and pound.

Types of measurement may be categorized in terms of the empirical relationships they seek to preserve. However, an important alternative is to categorize them in terms of the transformations that lead to other equally legitimate numerical representations. Thus, a numerical severity scale, in which only order matters, may be represented equally well by any numbers that preserve the order numbers derived through a monotonic or ordinal transformation of the original ones. For this reason, such scales are termed ordinal scales.

2.3 Distance Measures

Many data mining techniques (for example, nearest neighbor classification methods, cluster analysis, and multidimensional scaling methods) are based on similarity measures between objects. There are essentially two ways to obtain measures of similarity. First, they can be obtained directly from the objects. For example, a marketing survey may ask respondents to rate pairs of objects according to their similarity, or subjects in a food tasting experiment may be asked to state similarities between flavors of ice-cream. Alternatively, measures of similarity may be obtained indirectly from vectors of measurements or characteristics describing each object. In the second case it is necessary to define precisely what we mean by "similar," so that we can calculate formal similarity measures. Instead of talking about how similar two objects are, we could talk about how dissimilar they are. Once we have a formal definition of either "similar" or "dissimilar," we can easily define the other by applying a suitable monotonically decreasing transformation. For example, if s(i, j) denotes the similarity and d(i, j) denotes the dissimilarity between objects i and j, possible transformations include d(i, j) = 1 - s(i, j) and $\mathbf{d(i,j)} = \sqrt{(2(1 - s(i,j))}$. The term proximity is often used as a general term to denote either a measure of similarity or dissimilarity.

Two additional terms distance and metric are often used in this context. The term distance is often used informally to refer to a dissimilarity measure derived from the characteristics describing the objects as in Euclidean distance, defined below. A metric, on the other hand, is a dissimilarity measure that satisfies three conditions:

1. $d(i, j) \geq 0$ for all i and j, and $d(i, j) = 0$ if and only if $i = j$;

NOTES

2. d(i, j) = d(j, i) for all i and j; and

3. d(i, j) ≤ d(i, k) + d(k, j) for all i, j, and k. The third condition is called the triangle inequality.

Suppose we have n data objects with p real-valued measurements on each object. We denote the vector of observations for the ith object by **x(i)** = (x_1(i), x_2(i), . . . , x_p(i)), 1 ≤ i ≤ n, where the value of the k th variable for the ith object is x_k(i). The Euclidean distance between the ith and jth objects is defined as

$$\mathbf{d_E(i,j)} = \left(\sum_{k=1}^{p}(\mathbf{x_k(i)} - \mathbf{x_k(j)})^2\right)^{\frac{1}{2}}$$

This measure assumes some degree of commensurability between the different variables. Thus, it would be effective if each variable was a measure of length (with the number p of dimensions being 2 or 3, it would yield our standard physical measure of distance) or a measure of weight, with each variable measured using the same units. It makes less sense if the variables are noncommensurate. For example, if one variable were length and another were weight, there would be no obvious choice of units; by altering the choice of units we would change which variables were most important as far as the distance was concerned.

Since we often have to deal with data sets in which the variables are not commensurate, we must find some way to overcome the arbitrariness of the choice of units. A common strategy is to standardize the data by dividing each of the variables by its sample standard deviation, so that they are all regarded as equally important. (But note that this does not resolve the issue—treating the variables as equally important in this sense is still making an arbitrary assumption.) The standard deviation for the k th variable X_k can be estimated as

$$\bar{\sigma}_k = \left(\frac{1}{n}\sum_{i=1}(x'_k(i) - \mu_k)^2\right)^{\frac{1}{2}}$$

where μ_k is the mean for variable x'_k, which (if unknown) can be estimated using the sample mean $\bar{x}_k = \frac{1}{n}\sum_{i=1}^{n} x_k(i)$. Thus, $x'_k = \frac{x_k}{\bar{\sigma}_k}$ removes the effect of as captured by $\bar{\sigma}_k$.

In addition, if we have some idea of the relative importance that should be accorded to each variable, then we can weight them (after standardization), to yield the weighted Euclidean distance measure

$$d_{wE}(i,j) = \left(\sum_{k=1}^{p} w_k (x_k(i) - x_k(j))^2 \right)^{\frac{1}{2}}$$

The Euclidean and weighted Euclidean distances are both additive, in the sense that the variables contribute independently to the measure of distance. This property may not always be appropriate. To take an extreme case, suppose that we are measuring the height s and diameters of a number of cups. Using commensurate units, we could define similarities between the cups in terms of these two measurements. Now suppose that we measured the height of each cup 100 times, and the diameter only once (so that for any give n cup we have 101 variables, 100 of which have almost identical values). If we combined these measurements in a standard Euclidean distance calculation, the height would dominate the apparent similarity between the cups. However, 99 of the height measurements do not contribute anything to what we really want to measure; they are very highly correlated (indeed, perfectly, apart from measurement error) with the first height measurement. To eliminate such redundancy we need a data-driven method. One approach is to standardize the data, not just in the direction of each variable, as with weighted Euclidean distance, but also taking into account the covariance between the variables.

Then the sample covariance between X and Y is defined as

$$Cov(X,Y) = \frac{1}{n} \sum_{i=1}^{n} (x(i) - \bar{x})(y(i) - \bar{y}),$$

where it is the sample mean of the X values and is the sample mean of the Y values.

The covariance is a measure of how X and Y vary together: it will have a large positive value if large values of X tend to be associated with large values of Y and small values of X with small values of Y. If large values of X tend to be associated with small values of Y, it will take a negative value.

More generally, with p variables we can construct a p × p matrix of covariances, in which the element (k, l) is the covariance between the k th and lth variables. From the definition of covariance above, we can see that such a matrix (a co-variance matrix) must be symmetric.

The value of the covariance depends on the ranges of X and Y. This dependence can be removed by standardizing, dividing the values of X by their standard deviation and the values of Y by their standard deviation. The result is the sample correlation coefficient ρ(X, Y) between X and Y:

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}(x(i)-\bar{x})(y(i)-\bar{y})}{\left(\sum_{i=1}^{n}(x(i)-\bar{x})^2 \sum_{i=1}^{n}(y(i)-\bar{y})^2\right)^{\frac{1}{2}}}.$$

In the same way that a covariance matrix can be formed if there are p variables, a p × p correlation matrix can be formed in the same manner.

Note that covariance and correlation capture linear dependencies between variables (they are more accurately termed linear covariance and linear correlation). Consider data points that are uniformly distributed around a circle in two dimensions (X and Y), centered at the origin. The variables are clearly dependent, but in a nonlinear manner and they will have zero linear correlation. Thus, independence implies a lack of correlation, but the reverse is not generally true.

Recall again our coffee cup example with 100 measurements of height and one measurement of width. We can discount the effect of the 100 correlated variables by incorporating the covariance matrix in our definition of distance. This leads to the Mahalanobis distance between two p-dimensional measurements **x**(i) and **x**(j), defined as:

$$d_{MH}(i,j) = \left((\mathbf{x}(i)-\mathbf{x}(j))^T \Sigma^{-1} (\mathbf{x}(i)-\mathbf{x}(j))\right)^{\frac{1}{2}}$$

where T represents the transpose, ∑ is the p × p sample covariance matrix, and $\sum^{-1}$ standardizes the data relative to ∑. Note that although we have been thinking about our p-dimensional measurement vectors **x**(i) as rows in our data matrix, the convention in matrix algebra is to treat these as p × 1 column vectors (we can still visualize our data matrix as being an n × p matrix). Entry (k, l) of ∑ is defined between variable X_k and X_l. Thus, we have a p × 1 vector transposed (to give a 1 × p vector), multiplied by the p × p matrix $\sum^{-1}$, multiplied by a p × 1 vector, yielding a scalar distance. Of course, other matrices could be used in place of ∑. Indeed, the statistical frameworks of canonical variates analysis and discriminate analysis use the average of the covariance matrices of different groups of cases.

The Euclidean metric can also be generalized in other ways. For example, one obvious generalization is to the Minkowski or L_λ metric:

$$\left(\sum_{k=1}^{p}(x_k(i) - x_k(j))^\lambda)\right)^{\frac{1}{\lambda}},$$

where $\lambda \geq 1$. Using this, the Euclidean distance is the special case of $\lambda = 2$. The L_1 metric (also called the Manhattan or city-block metric) can be defined as

$$\sum_{k=1}^{p} \mid x_k(i) - x_k(j)) \mid .$$

The case $\lambda \rightarrow \infty$ yields the L_∞ metric

$$\max_k |x_k(i) - x_k(j)| .$$

There is a huge number of other metrics for quantitative measurements, so the problem is not so much defining one but rather deciding which is most appropriate for a particular situation.

For multivariate binary data we can count the number of variables on which two objects take the same or take different values. Consider, in which all p variables defined for objects i and j take values in {0, 1}; the entry $n_{1,1}$ in the box for i = 1 and j = 1 denotes that there are $n_{1,1}$ variables such that i and j both have value 1.

With binary data, rather than measuring the dissimilarities between objects, we often measure the similarities. Perhaps the most obvious measure of similarity is the simple matching coefficient, defined as

$$\frac{n_{1,1} + n_{0,0}}{n_{1,1} + n_{1,0} + n_{0,1} + n_{0,0}},$$

the proportion of the variables on which the objects have the same value, where $n_{1,1} + n_{1,0} + n_{0,1} + n_{0,0} = p$, the total number of variables. Sometimes, however, it is inappropriate to include the (0,0) cell (or the (1,1) cell, depending on the meaning of 0 and 1).

For example, if the variables are scores of the presence (1) or absence (0) of certain properties, we may not care about all the irrelevant properties had by neither object. (For instance, in vector representations of text documents it may be not be relevant that two documents do not contain thousands of specific terms). This consideration leads to a modification of the matching coefficient, the Jaccard coefficient, defined as

$$\frac{n_{1,1}}{n_{1,1} + n_{1,0} + n_{0,1}}.$$

The Dice coefficient extends this argument. If (0,0) matches are irrelevant, then (0,1) and (1,0) mismatches should lie between (1, 1) matches and (0,0) matches in terms of relevance. For this reason the number of (0,1) and (1,0) mismatches should be multiplied by a half. This yields $2n_{1,1}/(2n_{1,1} + n_{1,0} + n_{0,1})$. As with quantitative data, there are many different measures for multivariate binary data again the problem is not so much defining such measures but choosing one that possesses properties that are desirable for the problem at hand.

For categorical data in which the variables have more than two categories, we can score 1 for variables on which the two objects agree and 0 otherwise, expressing the sum of these as a fraction of the possible total p. If we know about the categories, we might be able to define a matrix giving values for the different kinds of disagreement.

Additive distance measures can be readily adapted to deal with mixed data types (e.g., some binary variables, some categorical, and some quantitative) since we can add the contributions from each variable. Of course, the question of relative standardization still arises.

2.5 The Form of Data

The data sets come in different forms; these forms are known as schemas. The simplest form of data (and the only form we have discussed in any detail) is a set of vector measurements on objects o(1), . . . , o(n). For each object we have measurements of p variables $X_1, \ldots, X_p$. Thus, the data can be viewed as a matrix with n rows and p columns. We refer to this standard form of data as a data matrix, or simply standard data. We can also refer to the data set as a table.

NOTES

Often there are several types of objects we wish to analyze. For example, in a payroll database, we might have data both about employees, with variables name, department - name, age, and salary, and about departments with variables department -name, budget and manager. These data matrices are connected to each other by the occurrence of the same (categorical) values in the department -name fields and in the fields name and manager. Data sets consisting of several such matrices or tables are called multirelational data.

In many cases multirelational data can be mapped to a single data matrix or table. For example, we could join the two data tables using the values of the variable department - name. This would give us a data matrix with the variables name, department -name, age, salary, budget (of the department), and manager (of the department). The possibility of such a transformation seems to suggest that there is no need to consider multirelational structures at all since in principle we could represent the data in one large table or matrix. However, this way of joining the data sets is not the only possibility: we could also create a table with as many rows as there are departments (this would be useful if we were interested in getting information about the departments, e.g., determining whether there was dependence between the budget of a department and the age of the manager). Generally no single table best captures all the information in a multirelational data set. More important, from the point of view of efficiency in storage and data access, "flattening" multirelational data to form a single large table may involve the needless replication of numerous values.

Some data sets do not fit well into the matrix or table form. A typical example is a time series, in which consecutive values correspond to measurements taken at consecutive times, (e.g., measurements of signal strength in a waveform, or of responses of a patient at a series of times after receiving medical treatment). We can represent a time series using two variables, one for time and one for the measurement value at that time. This is actually the most natural representation to use for storing the time series in a database. However, representing the data as a two-variable matrix does not take into account the ordered aspect of the data. In analyzing such data, it is important to recognize that a natural order does exist. It is common, for example, to find that neighboring observations are more closely related (more highly correlated) than distant observations. Failure to account for this factor could lead to a poor model.

A string is a sequence of symbols from some finite alphabet. A sequence of values from

a categorical variable is a string, and so is standard English text, in which the values are alphanumeric characters, spaces, and punctuation marks. Protein and DNA/RNA sequences are other examples. Here the letters are individual proteins (note that a string representation of a protein sequence is a 2-dimensional view of a 3-dimensional structure). A string is another data type that is ordered and for which the standard matrix form is not necessarily suitable.

A related ordered data type is the event-sequence. Given a finite alphabet of categorical event types, an event -sequence is a sequence of pairs of the form {event, occurrence time}. This is quite similar to a string, but here each item in the sequence is tagged with an occurrence time. An example of an event -sequence is a telecommunication alarm log, which includes a time of occurrence for each alarm. More complicated event -sequences include transaction data (such as records of retail or financial transactions), in which each transaction is time-stamped and the events themselves can be relatively complex (e.g., listing all purchases along with prices, department names, and so forth). Furthermore, there is no reason to restrict the concept of event sequences to categorical data; for example we could extend it to real -valued events occurring asynchronously, such as data from animal behavioral experiments or bursts of energy from objects in deep space.

2.6 Data Quality for Individual Measurements

The effectiveness of a data mining exercise depends critically on the quality of the data. In computing this idea is expressed in the familiar acronym GIGO Garbage In, Garbage Out. Since data mining involves secondary analysis of large data sets, the dangers are multiplied. It is quite possible that the most interesting patterns we discover during a data mining exercise will have resulted from measurement inaccuracies, distorted samples or some other unsuspected difference between the reality of the data and our perception of it.

It is convenient to characterize data quality in two ways: the quality of the individual records and fields, and the overall quality of the collection of data. We deal with each of these in turn.

No measurement procedure is without the risk of error. The sources of error are infinite, ranging from human carelessness, and instrumentation failure, to inadequate definition of what it is that we are measuring. Measuring instruments can lead to errors in two ways: they can be inaccurate or they can be imprecise. This distinction is important, since different strategies are required for dealing with the different kinds of errors.

A precise measurement procedure is one that has small variability (often measured by its variance). Using a precise process, repeated measurements on the same object under the same conditions will yield very similar values. Sometimes the word precision is taken to connote a large number of digits in a given recording. We do not adopt this interpretation, since such "precision" can all too easily be spurious, as anyone familiar with modern data analysis packages (which sometimes give results of calculations to eight or more decimal places) will know.

An accurate measurement procedure, in contrast, not only possesses small variability, but also yields results close to what we think of as the true value. A measurement procedure may yield precise but inaccurate measurements. For example repeated measurements of someone's height may be precise, but if these were made while the subject was wearing shoes, the result would be inaccurate. In statistical terms, the difference between the mean of repeated measurements and the true value is the bias of a measurement procedure. Accurate procedures have small bias as well as small variance.

2.7 Data Quality for Collections of Data

In addition to the quality of individual observations, we need to consider the quality of collections of observations. Much of statistics and data mining is concerned with inference from a sample to a population, that is, how, on the basis of examining just a fraction of the objects in a collection, one can infer things about the entire population. Statisticians use the term parameter to refer to descriptive summaries of populations or distributions of objects (more generally, of course, a parameter is a value that indexes a family of mathematical functions). Values computed from a sample of objects are called statistics, and appropriately chosen statistics can be used as estimates of parameters. Thus, for example, we can use the average of a sample as an estimate of the mean (parameter) of an entire population or distribution.

NOTES

Such estimates are useful only if they are accurate. As we have just noted, inaccuracies can occur in two ways. Estimates from different samples might vary greatly, so that they are unreliable: using a different sample might have led to a very different estimate. Or the estimates might be biased, tending to be too large or too small. In general, the precision of an estimate (the extent to which it would vary from sample to sample) increases with increasing sample size; as resources permit, we can reduce this uncertainty to an acceptable value. Bias, on the other hand, is not so easily diminished.

Some estimates are intrinsically biased, but do not cause a problem because the bias decreases with increasing sample size. Of more significance in data mining are biases arising from an inappropriate sample. If we wanted to calculate the average weight of people living in New York, it would obviously be inadvisable to restrict our sample to women. If we did this, we would probably underestimate the average. Clearly, in this case, the population from which our sample is drawn (women in New York) is not the population to which we wish to generalize (everyone in New York). Our sampling frame, the list of people from which we will draw our sample, does not match the population about which we want to make an inference. This is a simple example we were able to clearly identify the population from which the sample was drawn (women in New York). Difficulties arise when it is less obvious what the effect of the incorrect sampling frame will be. Suppose, for example, that we drew our sample from people working in offices. Would this lead to biased estimates? Maybe the sexes are disproportionately represented in offices. Maybe office workers have a tendency to be heavier than average because of their sedentary occupation. There are many reasons why such a sample might not be representative of the population we aim to study.

Because we often have no control over the way the data are collected, quality issues are particularly important in data. Our data set may be a distorted sample of the population we wish to describe. If we know the nature of this distortion then we might be able to allow for it in our inferences, but in general this is not the case and inferences must be made with care. The terms opportunity sample and convenience sample are sometimes used to describe samples that are not properly drawn from the population of interest. The sample of office workers above would be a convenience sample it is much more convenient to sample from them than to sample from the whole population of New York. Distortions of a sample can occur for many reasons, but the risk is

NOTES

especially grave when humans are involved. The effects can be subtle and unexpected: for instance, in large samples, the distribution of stated ages tends to cluster around integers ending with 0 or 5 just the sort of pattern that data mining would detect as potentially interesting. Interesting it may be, but will probably be of no value in our analysis.

A different kind of distortion occurs when customers are selected through a chain of selection steps. With bank loans, for example, an initial population of potential customers is contacted (some reply and some do not), those who reply are assessed for creditworthiness (some receive high scores and some do not), those with high scores are offered a loan (some accept and some do not), those who take out a loan are followed up (some are good customers, paying the installments on time, and others are not), and so on. A sample drawn at any particular stage would give a distorted perspective on the population at an earlier stage.

In this example of candidates for bank loans, the selection criteria at each step are clearly and explicitly stated but, as noted above, this is not always the case. For example, in clinical trials samples of patients are selected from across the country, having been exposed to different diagnostic practices and perhaps different previous treatments in different primary care facilities. Here the notion of taking a "random sample from a well-defined population" makes no sense. This problem is compounded by the imposition of inclusion/exclusion criteria: perhaps the patients must be male, aged between 18 and 50, with a primary diagnosis of the disease in question made no longer than two years ago, and so on. (It is hardly surprising in this context, that the sizes of effects recorded in clinical trials are typically larger than those found when the treatments are applied more widely. On the other hand it is reassuring that the directions of the effects do normally generalize in this way.)

In addition to sample distortion arising from a mismatch between the sample population and the population of interest other kinds of distortion arise. The aim of many data mining exercises is to make some prediction of what will happen in the future. In such cases it is important to remember that populations are not static. For instance the nature of a customers shopping at a certain store will change over time, perhaps because of changes in the social culture of the surrounding neighborhood, or in response to a marketing initiative, or for many other reasons. Much work on predictive methods has failed to take account of such population drift. Typically, the future performance of such methods is

NOTES

assessed using data collected at the same time as the data used to build the model implicitly assuming that the distribution of objects used to construct the model is the same as that of future objects. Ideally, a more sophisticated model is required that can allow for evolution over time. In principle, population drift can be modeled, but in practice this may not be easy.

Distortion of samples can be viewed as a special case of incomplete data, one in which entire records are missing from what would otherwise be a representative sample. Data can also be missing in other ways. In particular, individual fields may be missing from records. In some ways this is not as serious as the situation described above. (At least here, one can see that the data are missing!) Still, significant problems may arise from incomplete data. The fundamental question is "Why are the data missing?" Was there information in the missing data that is not present in the data that have been recorded? If so, inferences based on the observed data are likely to be biased. In any incomplete data problem, it is crucial to be clear about the objectives of the analysis. In particular, if the aim is to make an inference only about the cases that have complete records, inferences based only on the complete cases is entirely valid.

3. Visualizing and Exploring Data

Structure

3.1 Introduction

3.2 Summarizing Data: Some Simple Examples

3.3 Tools for Displaying Single Variables

3.4 Tools for Displaying Relationships between Two Variables

3.5 Tools for Displaying More Than Two Variables

3.6 Principal Components Analysis

3.7 Multidimensional Scaling

Objective

After going through this lesson, you should be able to:

- Discuss different types of tools for displaying;
- Discuss about principal components analysis;
- Discuss about multidimensional scaling;

3.1 Introduction

This lesson explores visual methods for finding structures in data. Visual methods have a special place in data exploration because of the power of the human eye/brain to detect structures the product of aeons of evolution. Visual methods are used to display data in ways that capitalize upon the particular strengths of human pattern processing abilities. This approach lies at quite the opposite end of the spectrum from methods for formal model building and for testing to see whether observed data could have arisen from a hypothesized data generating structure. Visual methods are important in data mining because they are ideal for sifting through data to find unexpected relationships. On the other hand, they do have their limitations, particularly, as we illustrate below, with very large data sets.

Exploratory data analysis can be described as data-driven hypothesis generation. We examine the data, in search of structures that may indicate deeper relationships between cases or variables. This process stands in contrast to hypothesis testing which begins with a proposed model or hypothesis and undertakes statistical manipulations to determine the likelihood that the data arose from such a model. The phrase data based in the above description indicates that it is the patterns in the data that give rise to the hypotheses in contrast to situations in which hypotheses are generated from theoretical arguments about underlying mechanisms. This distinction has implications for the legitimacy of subsequent testing of the hypotheses. If we take 10 random samples of size 20 from the same population, and measure the values of a single variable, the random samples will have different means (just by virtue of random variability). We could compare the means using formal tests. Suppose, however, we took only the two samples giving rise to the smallest and largest means, ignoring the others. A test of the difference between these means might well show significance. If we took 100 samples, instead of 10, then we would be even more likely to find a significant difference between the largest and the smallest means. By ignoring the fact that these are the largest and smallest in a set of 100, we are biasing the analysis toward detecting a difference even though the samples were generated from the same population.

In general, when searching for patterns, we cannot test whether a discovered pattern is a real property of the underlying distribution (as opposed to a chance property of the sample) without taking into account the size of the search the number of possible patterns we have examined. The informal nature of exploratory data analysis makes this very difficult it is often impossible to say how many patterns have been examined. For this reason researchers often use a separate data set, obtained from the same source as the first, to conduct formal testing for the existence of any pattern.

3.2 Summarizing Data: Some Simple Examples

We mentioned in earlier chapters that the mean is a simple summary of the average of a collection of values. Suppose that x(1), ..., x(n) comprise a set of n data values. The sample mean is defined as

$$\hat{\mu} = \sum_i x(i)/n.$$

(Note that we use μ to refer to the true mean of the population, and $\hat{\mu}$ to refer a sample- based estimate of this mean). The sample mean has the property that it is the value that is "central" in the sense that it minimizes the sum of squared differences between it and the data values. Thus, if there are n data values, the mean is the value such that the sum of n copies of it equals the sum of the data values.

NOTES

The mean is a measure of location. Another important measure of location is the median, which is the value that has an equal number of data points above and below it. (Easy if n is an odd number. When there is an even number it is usually defined as halfway between the two middle values.)

The most common value of the data is the mode. Sometimes distributions have more than one mode (for example, there may be 10 objects which take the value 3 on some variable, and another 10 which take the value 7, with all other values taken less often than 10 times) and are therefore called multimodal.

Other measures of location focus on different parts of the distribution of data values. The first quartile is the value that is greater than a quarter of the data points. The third quartile is greater than three quarters. (We leave it to you to discover why we have not mentioned the second quartile.) Likewise, deciles and percentiles are sometimes used. Various measures of dispersion or variability are also common. These include the standard deviation and its square, the variance. The variance is defined as the average of the squared differences between the mean and the individual data values:

$$\hat{\sigma}^2 = \sum_i (x(i) - \mu)^2 / n.$$

Note that since the mean minimizes the sum of these squared differences, there is a close link between the mean and the variance. If µ is unknown, as is often the case in practice, we can replace µ above with, our data based estimate. When µ is replaced with $\hat{\mu}$ to get an unbiased estimate, the variance is estimated as

$$\sum_i (x(i) - \hat{\mu})^2 / (n - 1).$$

The standard deviation is the square root of the variance:

$$\hat{\sigma} = \sqrt{\sum_i (x(i) - \mu)^2 / n}.$$

The interquartile range, common in some applications, is the difference between the third and first quartile. The range is the difference between the largest and smallest data point.

Skewness measures whether or not a distribution has a single long tail and is commonly defined as

$$\frac{\sum(x(i)-\hat{\mu})^3}{\left(\sum(x(i)-\hat{\mu})^2\right)^{3/2}}.$$

For example, the distribution of peoples' incomes typically shows the vast majority of people earning small to moderate amounts, and just a few people earning large sums, tailing off to the very few who earn astronomically large sums the Bill Gateses of the world. A distribution is said to be right-skewed if the long tail extends in the direction of increasing values and left-skewed otherwise. Right-skewed distributions are more common. Symmetric distributions have zero skewness.

3.3 Tools for Displaying Single Variables

One of the most basic displays for univariate data is the histogram, showing the number of values of the variable that lie in consecutive intervals. With small data sets, histograms can be misleading: random fluctuations in the values or alternative choices for the ends of the intervals can give rise to very different diagrams. Apparent multimodality can arise, and then vanish for different choices of the intervals or for a different small sample. As the size of the data set increases, however, these effects diminish. With large data sets, even subtle features of the histogram can represent real aspects of the distribution.

Figure 3.1 shows a histogram of the number of weeks during 1996 in which owners of a particular credit card used that card to make supermarket purchases (the label on the vertical axis has been removed to conceal commercially sensitive details). There is a large mode to the left of the diagram: most people did not use their card in a supermarket, or used it very rarely. The number of people who used the card a given number of times decreases rapidly with increases in the number of times. However, the relatively large number of people represented in this diagram allows us to detect another, much smaller mode toward the right hand end of the diagram. Apparently there is a tendency for people to make regular weekly trips to a supermarket, though this is reduced from 52 annual transactions, probably by interruptions such as holidays.

NOTES

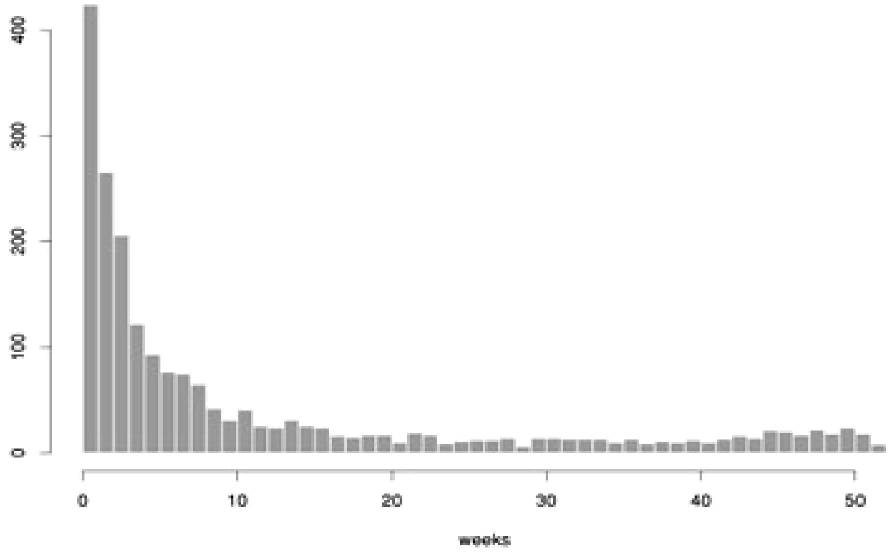


Figure 3.1: Histogram of the number of weeks of the year a particular brand of credit card was used.

The disadvantages of histograms have also been tackled by smoothing estimates. One of the most widely used types is the kernel estimate.

Kernel estimates smooth out the contribution of each observed data point over a local neighborhood of that point. Consider a single variable X for which we have measured values {x(1), ..., x(n)}. The contribution of data point x(i) to the estimate at some point x* depends on how far apart x(i) and x* are. The extent of this contribution is dependent upon on the shape of the kernel function adopted and the width accorded to it. Denoting the kernel function by K and its width (or bandwidth) by h, the estimated density at any point x is

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K\left(\frac{x - x(i)}{h}\right),$$

where $\int \mathbf{K(t)dt} = \mathbf{1}$ to ensure that the estimate f(x) itself integrates to 1 (i.e., is a proper density) and where the kernel function K is usually chosen to be a smooth unimodal function with a peak at 0. The quality of a kernel estimate depends less on the shape of K than on the value of h.A common form for K is the Normal (Gaussian) curve, with h as its spread parameter (standard deviation), i.e.,

$$K(t, h) = Ce^{-\frac{1}{2}(\frac{t}{h})^2}$$

where C is a normalization constant and t = x - x(i) is the distance of the query point x to data point x(i). The bandwidth h is equivalent to s, the standard deviation (or width) of the Gaussian kernel function.

There are formal methods for optimizing the fit of these estimates to the unknown distribution that generated the data, but here our interest is in graphical procedures. For our purposes the attraction of such estimates is that by varying h, we can search for peculiarities in the shape of the sample distribution. Small values of h lead to very spiky estimates (not much smoothing at all), while large values lead to over smoothing. The limits at each extreme of h are the empirical distribution of the data points (i.e., "delta functions" on each data point x(i)) as h→0, and a uniform flat distribution as h→8. These limits correspond to the extremes of total commitment to the data (with no mass anywhere except at the observed data points), versus completely ignoring the observed data.

Figure 3.2 shows a kernel estimate of the density of the weights of 856 elderly women who took part in a study of osteoporosis. The distribution is clearly right skewed and there is a hint of multimodality. Certainly the assumption often made in classical statistical work that distributions are normal does not apply in this case. (This is not to say that statistical techniques nominally based on that assumption might not still be valid. Often the arguments are asymptotic based on normality arising from the central limit theorem. In this case, the assumption that the sample mean of 856 subjects would vary from sample to sample according to a normal distribution would be reasonable for practical purposes.)

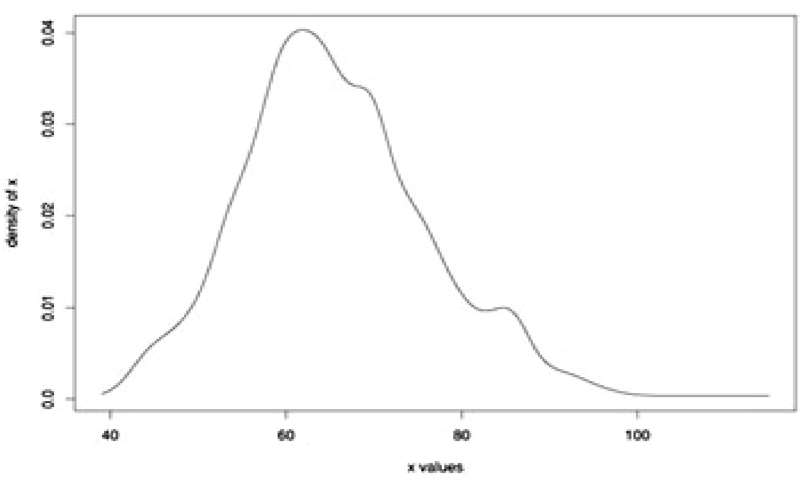


Figure 3.2: Kernel estimate of the weights (in kg) of 856 elderly women.

Figure 3.3 shows what happens when a larger value is used for the smoothing parameter h. Which of the two kernel estimates is "better" is a difficult question to answer. Figure 3.3 is more conservative in that less credence is given to local (potentially random) fluctuations in the observed data values.

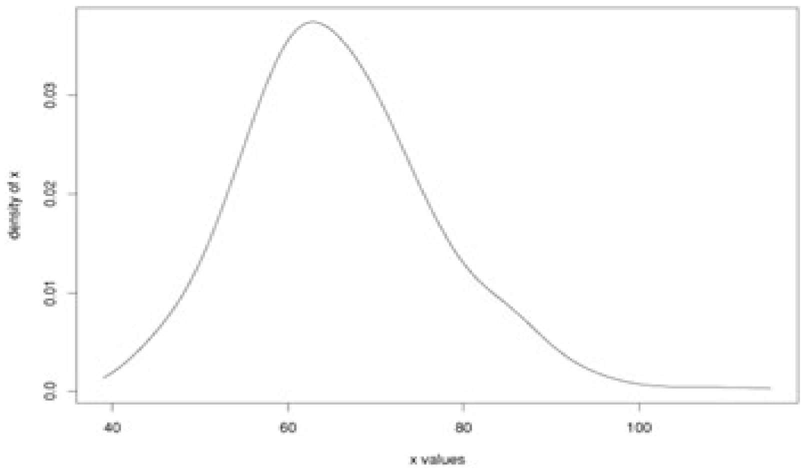


Figure 3.3: As figure 3.2, but with more smoothing.

Although this section focuses on displaying single variables, it is often desirable to display different groups of scores on a single variable separately, so that the groups may be compared. (Of course, we can think of this as a two-variable situation, in which one of the variables is the grouping factor.) Histograms, kernel plots, and other unidimensional displays can be used separately for each group. However, this can become unwieldy if there are more than two or three groups. In such cases a useful alternative display is the box and whisker plot.

Although various versions of box and whisker plots exist, the essential ideas are the same. A box containing which the bulk of the data is defined for example, the interval between the first and third quartiles. A line across this box indicates some measure of location often the median of the data. Whiskers project from the ends of the box to indicate the spread of the tails of the empirical distribution.

3.4 Tools for Displaying Relationships between Two Variables

The scatterplot is a standard tool for displaying two variables at a time. Figure 3.4 shows the relationship between two variables describing credit card repayment patterns (the details are confidential). It is clear from this diagram that the variables are strongly correlated when one value has a high (low) value; the other variable is likely to have a high (low) value. However, a significant number of people depart from this pattern; showing high values on one of the variables and low values on the other. It might be worth investigating these individuals to find out why they are unusual.

Unfortunately, in data mining, scatterplots are not always so useful. If there are too many data points we will find ourselves looking at a purely black rectangle. Figure 3.5 illustrates this sort of problem. This shows a scatterplot of 96,000 points from a study of bank loans. Little obvious structure is discernible, although it might appear that later applicants in general are older. On the other hand, the apparent greater vertical dispersion toward the right end of the diagram could equally be caused by a greater number of samples on the right side. In fact, the linear regression fit to these data has a very small but highly significant downward slope.

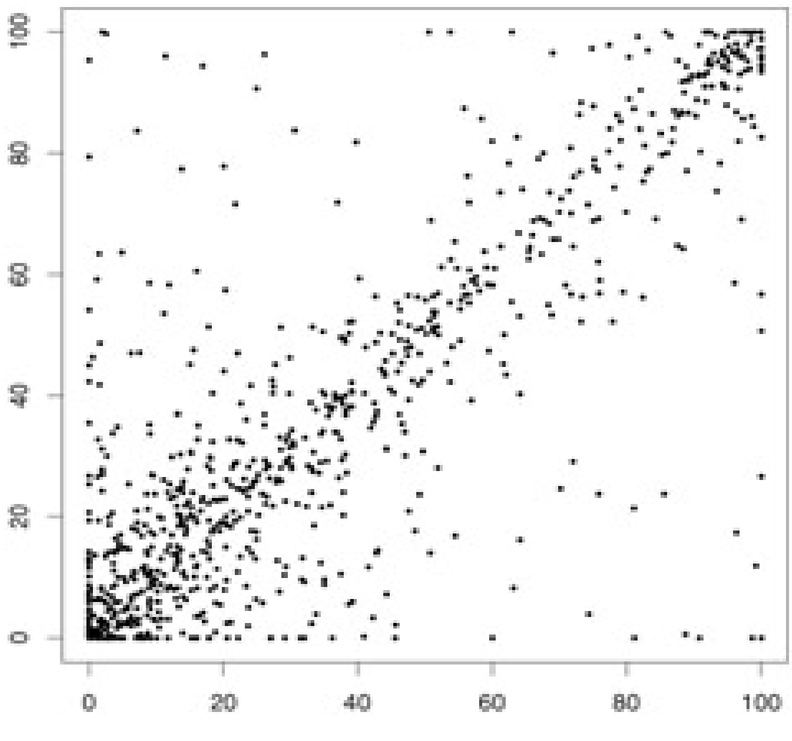


Figure 3.4: A standard scatterplot for two banking variables.

NOTES

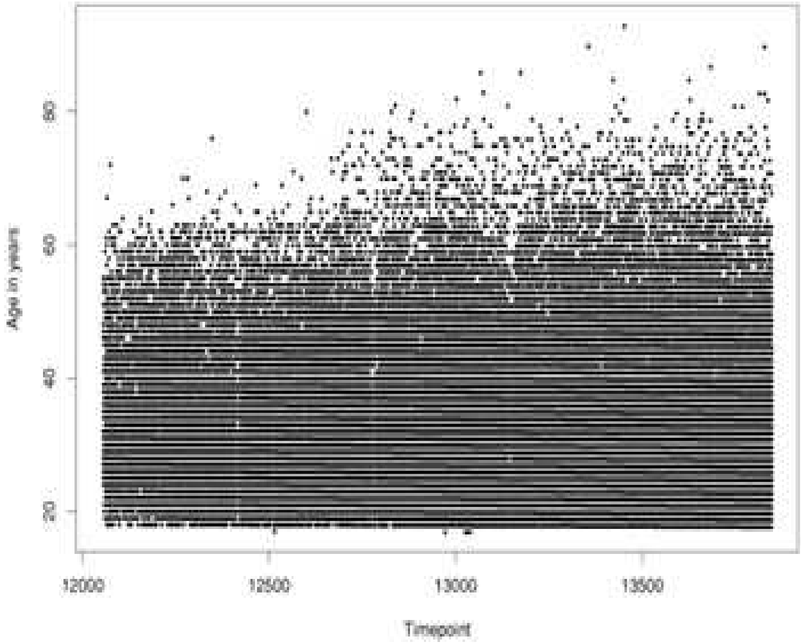


Figure 3.5: A scatterplot of 96,000 cases, with much overprinting. each data point represents an individual applicant for a loan. the vertical axis shows the age of the applicant, and the horizontal axis indicates the day on which the application was made.

Even when the situation is not quite so extreme, scatterplots with large numbers of points can conceal more than they reveal. Figure 3.6 plots the number of weeks a particular credit card was used to buy petrol (gasoline) in a given year against the number of weeks the card was used in a supermarket (each data point represents an individual credit card). There is clearly some correlation, but the actual correlation 0.482 is much higher than it appears here. The diagram is deceptive because it conceals a great deal of overprinting in the bottom left corner there are 10,000 customers represented here altogether. The bimodality shown in figure 3.1 can also be discerned in this figure, though not as easily as in figure 3.1.

Another curious phenomenon is also apparent in figure 3.6. The distribution of the number of weeks the card was used in a petrol station is skewed for low values of the supermarket variable, but fairly uniform for high values. What could explain this? (Of course, bearing in mind the point above, this apparent phenomenon needs to be checked for overprinting.)

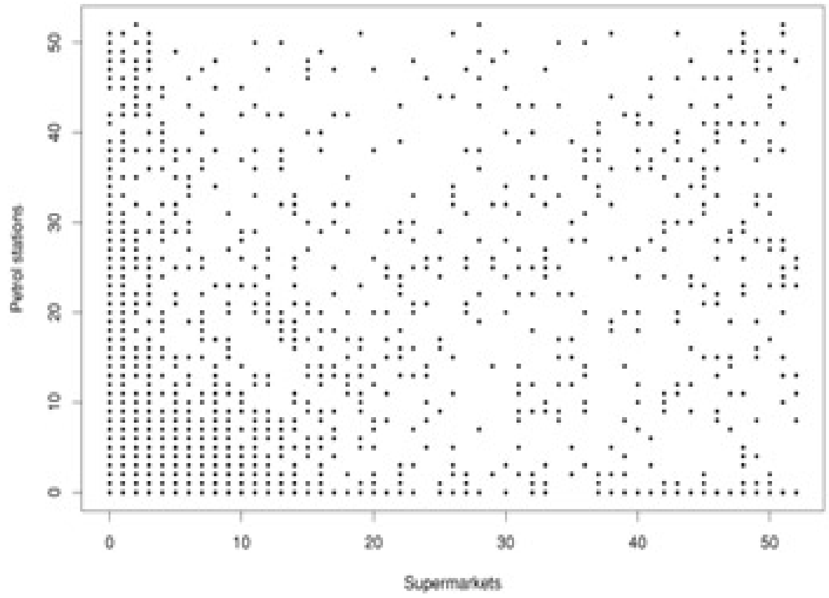


Figure 3.6: Overprinting conceals the actual strength of the correlation.

Contour plots can help overcome some of these problems. Note that creating a contour plot in two dimensions effectively requires us to construct a two -dimensional density estimate, using something like a two -dimensional generalization of the kernel method of equation, again raising the issue of bandwidth selection but now in a two-dimensional context. A contour plot of the 96,000 points shown in figure 3.5 is given in figure 3.7. Certain trends are clear from this display that cannot be discerned in figure 3.5. For instance the density of points increases toward the right side of the diagram; the apparent increasing dispersion of the vertical axis is due to there being a greater concentration of points in that area. The vertical skewness of the data is also very evident in this diagram. The unimodality of the data and the position of the single mode cannot be seen at all in figure 3.5 but is quite clear in figure 3.7. Note that since the horizontal axis in these plots is time, an alternative way to display the data is to plot contours of constant conditional probability density, as time progresses.

NOTES

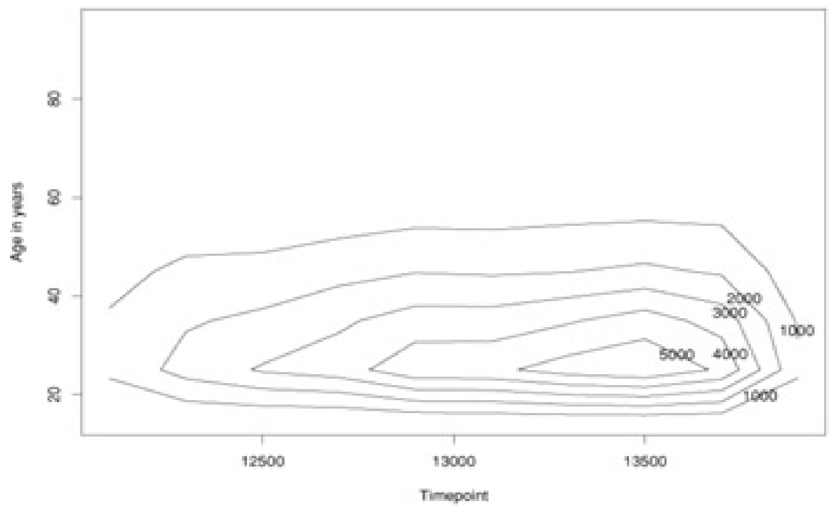


Figure 3.7: A contour plot of the data from figure 3.5.

Other standard forms of display can be used when one of the two variables is time, to show the value of the other variable as time progresses. This can be a very effective way of detecting trends and departures from expected or standard behaviour. Figure 3.8 shows a plot of the number of credit cards issued in the United Kingdom from 1985 to1993 inclusive. A smooth curve has been fitted to the data to place emphasis on the main features of the relationship. It is clear that around 1990 something caused a break in a growth pattern that had been linear up to that point. In fact, what happened was that in 1990 and 1991 annual fees were introduced for credit cards, and many users reduced their holding to a single card.

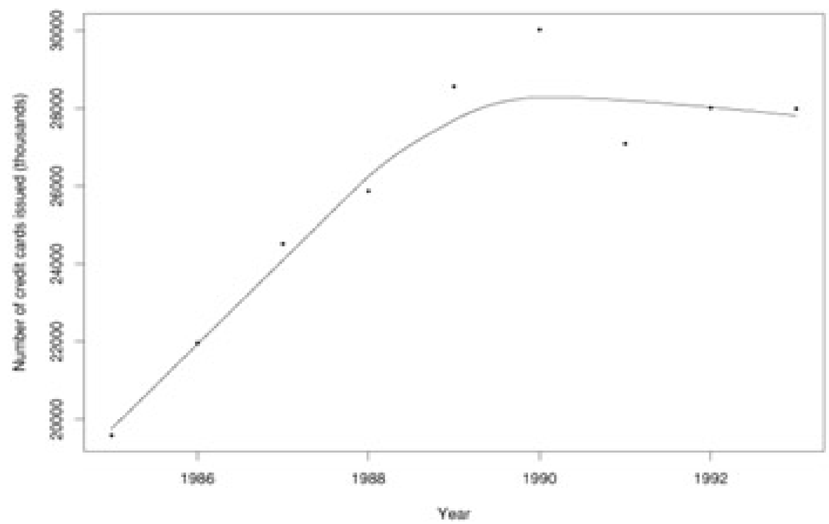


Figure 3.8: A plot of the number of credit cards in circulation in the united kingdom, by year.

Figure 3.9 shows a plot of the number of miles flown by UK airlines, during each month from January 1963 to December 1970. There are several patterns immediately apparent from this display that conform with what one might expect to observe, such as the gradually increasing trend and the periodicity (with large peaks in the summer and small peaks around the new year). The plot also reveals an interesting bifurcation of the summer peak, suggesting a tendency for travelers to favor the early and late summer over the middle period.

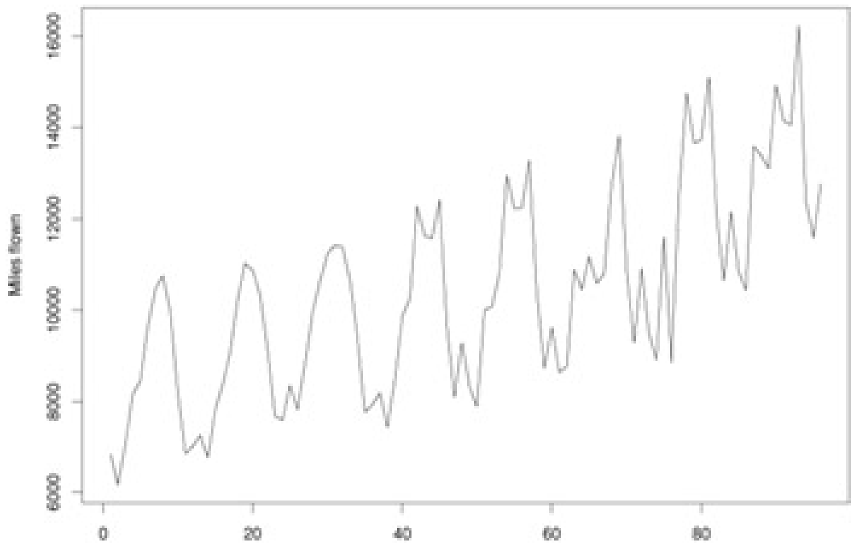


Figure 3.9: Patterns of change over time in the number of miles flown by uk airlines in the 1960s.

Figure 3.10 provides a third example of the power of plots in which time is one of the two variables. From February to June 1930, an experiment was carried out in Lanarkshire, Scotland to investigate whether adding milk to children's diets had an effect on "physique, general health and increasing mental alertness" (Leighton and McKinlay,1930). In this study 20,000 children were allocated to one of three groups; 5000 of the children received three-quarters of a pint of raw milk per day, 5000 received three - quarters of a pint of pasteurized milk per day, and 10,000 formed a control group receiving no dietary milk supplement. The children were weighed at the start of the experiment and again four months later. Interest lay in whether there was differential growth between the three groups.

Figure 3.10 plots the mean weight of the control group of girls against the mean age of the group they are in. The first point corresponds to the youngest age group (mean age 5.5 years) at the start of the experiment, and the second point corresponds to

NOTES

this group four months later. The third and fourth points correspond to the second age group, and so on. The points are connected by lines to make the shape easier to discern. Similar shapes are apparent for all groups in the experiment.

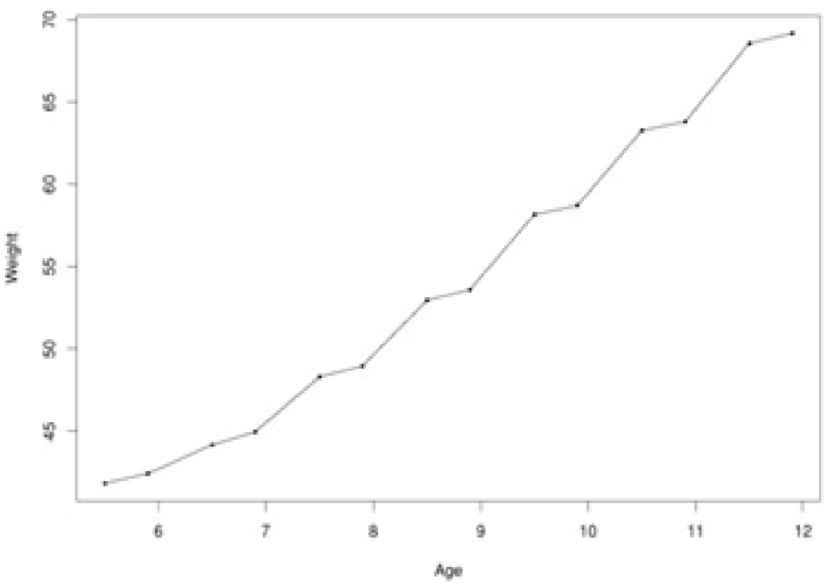


Figure 3.10: Weight changes over time in a group of 10,000 school children in the 1930s. the steplike pattern in the data highlights a problem with the measurement process.

The plot immediately reveals an unexpected pattern that cannot be seen from a table of the data. We would expect a smooth plot, but there are clear steps evident here. It seems that each age group does not gain as much weight as expected. There are various possible explanations for this shape. Perhaps children grow less during the early months of the year than during the later ones. However, similar plots of heights show no such intermittent growth, so we need a more elaborate explanation in which height increases uniformly but weight increases in spurts. Another possible explanation arises from the fact that the children were weighed in their clothes. The report does say, "All of the children were weighed without their boots or shoes and wearing only their ordinary outdoor clothing. The boys were made to turn out the miscellaneous collection of articles that is normally found in their pockets, and overcoats, mufflers, etc., were also discarded. Where a child was found to be wearing three or four jerseys a not uncommon experience all in excess of one were removed." It still seems likely, however, that the summer garb was lighter than the winter garb. This example illustrates that the patterns discovered

NOTES

by data mining may not shed much light on the phenomena under investigation, but finding data anomalies and shortcomings may be just as valuable.

3.5 Tools for Displaying More Than Two Variables

Since sheets of paper and computer screens are flat, they are readily suited for displaying two-dimensional data, but are not effective for displaying higher dimensional data. We need some kind of projection, from the higher dimensional data to a two dimensional plane, with modifications to show (aspects of) the other dimensions.

Figure 3.11 illustrates a scatterplot matrix for characteristics, performance measures, and relative performance measures of 209 computer CPUs dating from over 10 years ago. The variables are cycle time, minimum memory (kb), maximum memory (kb), cache size (kb), minimum channels, maximum channels, relative performance, and estimated relative performance (relative to an IBM 370/158 -3). While some pairs of variables appear to be unrelated, others are strongly related. Brushing allows us to highlight points in a scatterplot matrix in such a way that the points corresponding to the same objects in each scatterplot are highlighted. This is particularly useful in interactive exploration of data.

NOTES

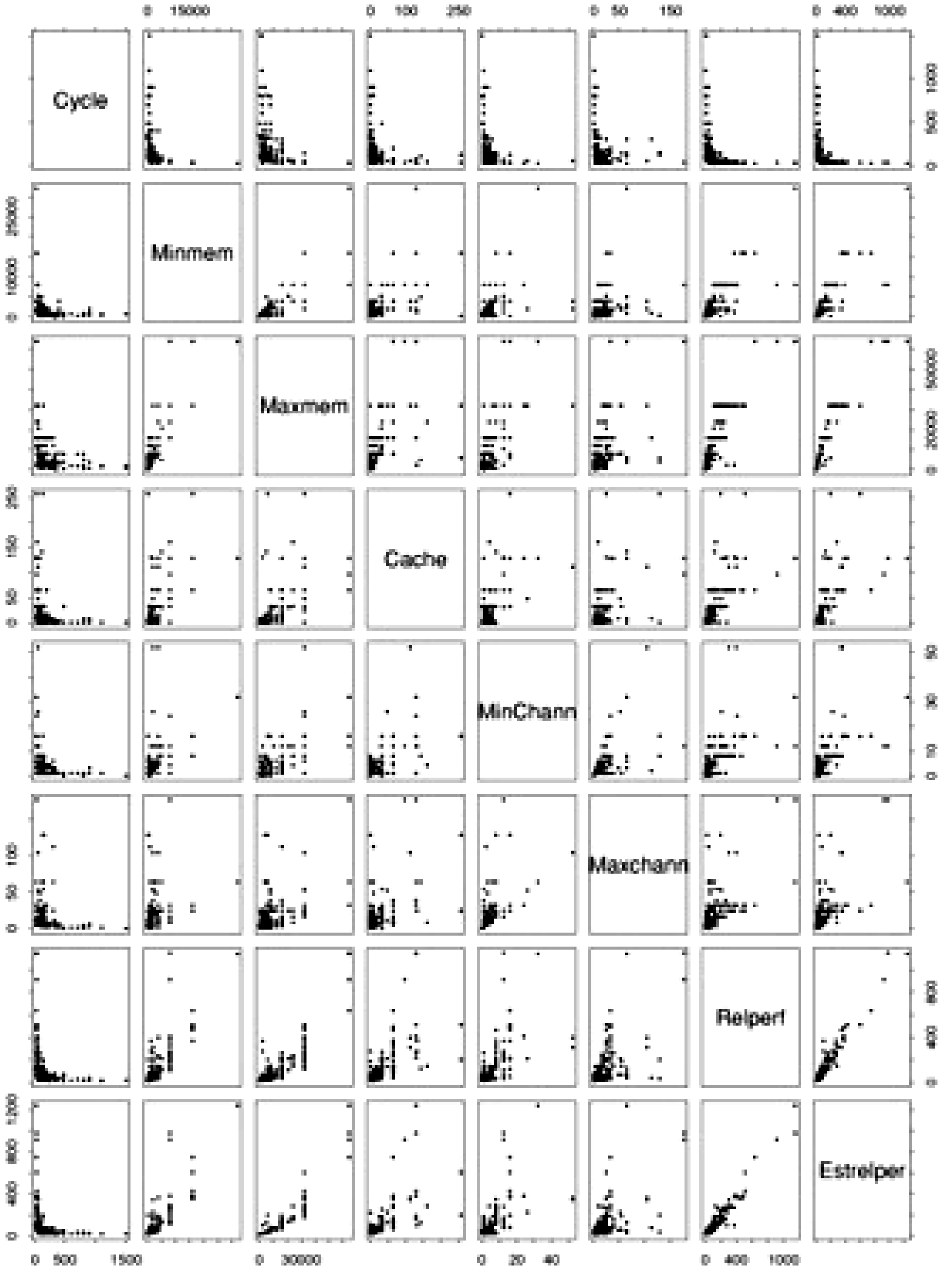


Figure 3.11: A scatterplot matrix for the computer CPU data.

Of course, scatterplot matrices are not really multivariate solutions: they are multiple bivariate solutions, in which the multivariate data are projected into multiple two - dimensional plots (and in each two-dimensional plot all other variables are ignored). Such projections necessarily sacrifice information. Picture a cube formed from eight smaller cubes. If data points are uniformly distributed in alternate subcubes, with the others being empty, all three one - dimensional and all three two-dimensional projections show uniform distributions. Interactive graphics come into their own

NOTES

when more than two variables are involved, since then we can rotate ("spin") the direction of projection in a search for structure. Some systems even let the software follow random rotations, while we watch and wait for interesting structures to become apparent. While this is a good idea in principle, the excitement of watching a cloud of points shift relative position as the direction of viewing changes can quickly pall, and more structured methods are desirable.

Trellis plotting also utilizes multiple bivariate plots. Here, however, rather than displaying a scatterplot for each pair of variables, they fix a particular pair of variables that is to be displayed and produce a series of scatterplots conditioned on levels of one or more other variables .

Figure 3.12 shows a trellis plot for data on epileptic seizures. The horizontal axis of each plot gives the number of seizures that 58 patients experienced over a certain two week period, and the vertical axis gives the number of seizures experienced over a later two week period. The two left hand graphs show the figures for males, and the two right hand graphs the figures for females. The two upper graphs show ages 29 to 42 while the two lower graphs show ages 18 to 28. (The original data set included the record of another subject who had much higher counts. We have removed this subject here so that we can more clearly see the relationships between the scores of the other subjects.) From these plots, we can see that the younger group show lower average counts than the older group. The figures also hint at some possible differences between the slopes of the estimated best fitting lines relating the y and x axes, though we would need to carry out formal tests to be confident that these differences were real.

Trellis plots can be produced with any kind of component graph. Instead of scatterplots in each cell, we could have histograms, time series plots, contour plots, or any other types of plots.

NOTES

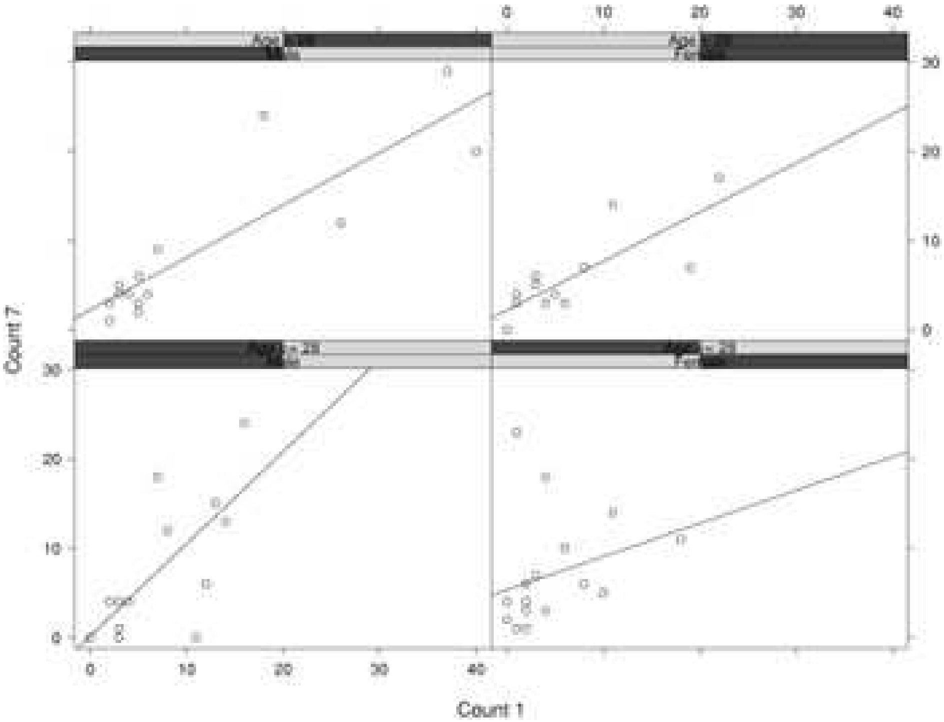


Figure 3.12: A trellis plot for the epileptic seizures data.

An entirely different way to display multivariate data is through the use of icons, small diagrams in which the sizes of different features are determined by the values of particular variables. Star icons are among the most popular. In these, different directions from the origin correspond to different variables, and the lengths of radii projecting in these directions correspond to the magnitudes of the variables. Figure 3.13 shows an example. The data displayed here come from 12 chemical properties that were measured on 53 mineral samples equally spaced along a long drill into the Earth's surface.

Another type of icon plot, Chernoff's faces, is discussed frequently in introductory texts on the subject. In these plots, the sizes of features in cartoon faces (length of nose, degree of smile, shape of eyes, etc.) represent the values of the variables. The method is based on the principle that the human eye is particularly adept at recognizing and distinguishing between faces. Although they are entertaining, plots of this type are seldom used in serious data analysis since the idea does not work very well in practice with more than a handful of cartoon faces. In general, iconic representations are effective only for relatively small numbers of cases since they require the eye to scan each case separately.Parallel coordinates plots show variables as parallel axes, representing each case as a piecewise linear plot connecting the measured values for that case. Figure 3.14 shows such a plot for four repeated measurements of the number of epileptic seizures experienced by 58 patients during successive two week periods. The data are clearly skewed and might be modeled by a Poisson distribution. Since the data set is not too large, we can follow the trajectories of individual patients.

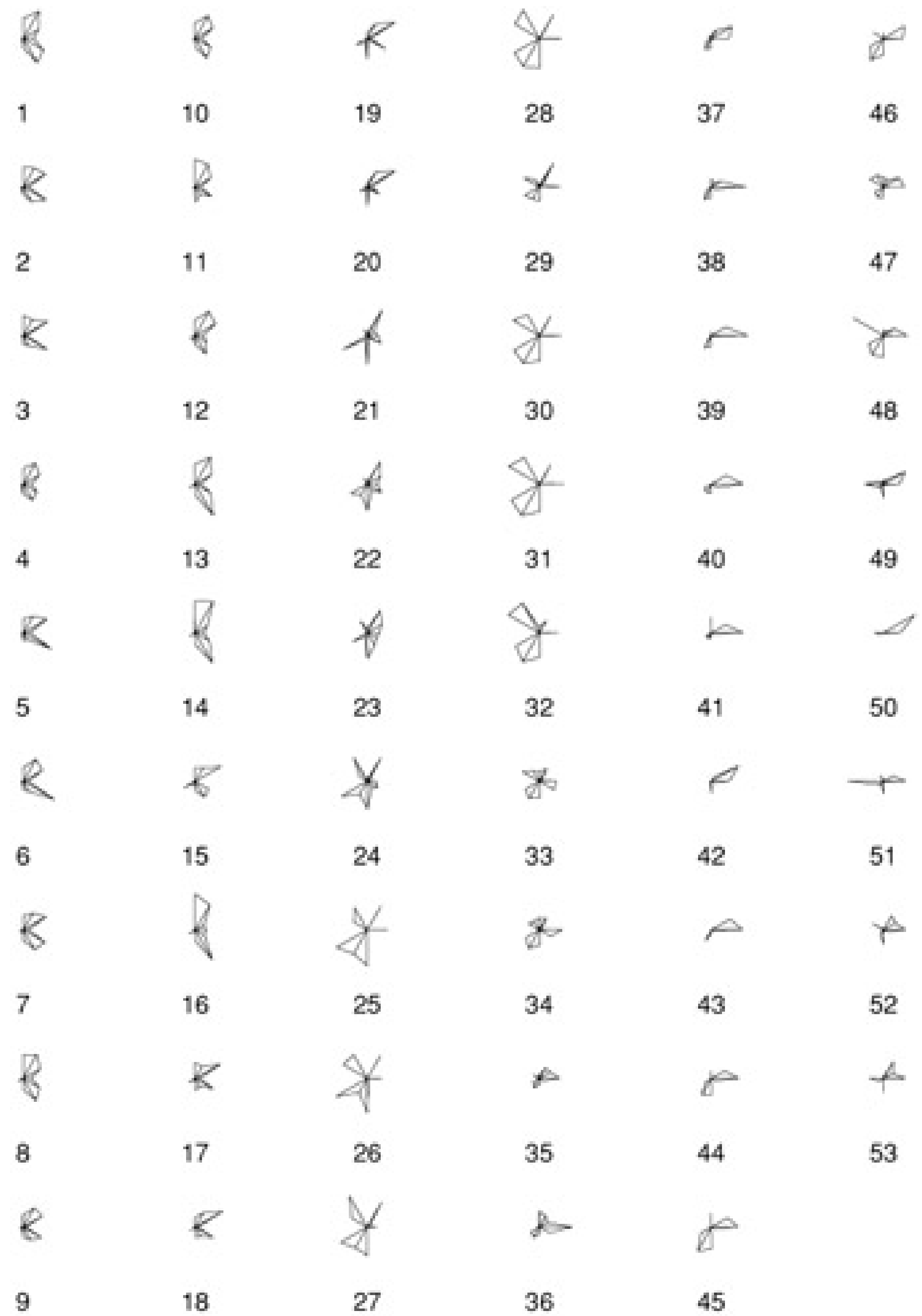


Figure 3.13: An example of a star plot.

NOTES

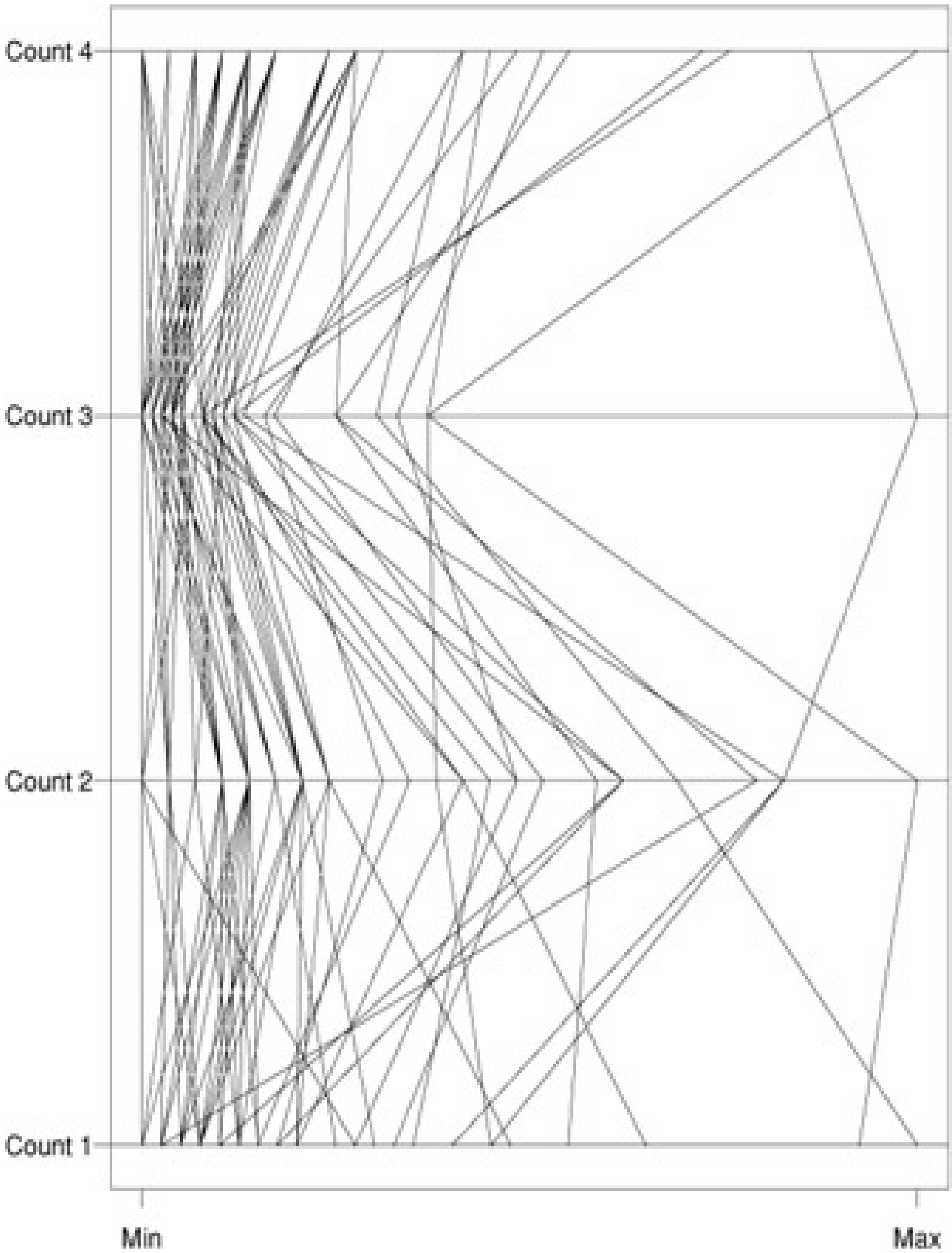


Figure 3.14: A parallel coordinates plot for the epileptic seizure data.

Another way of representing dimensions is through the use of color. Line styles, as in the parallel coordinates plot above, can serve the same purpose.

No single method of representing multivariate data is a universal solution. Which method is most useful in a given situation will depend on the data and on the structures being sought.

3.6 Principal Components Analysis

Scatterplots project multivariate data into a two-dimensional space defined by just two of the variables. This allows us to examine pair wise relationships between variables, but such simple projections might conceal more complicated relationships. To detect these relationships we can use projections along different directions, defined by any weighted linear combination of variables (e.g., along the direction defined by $2x_1 + 3x_2 + x_3$).

With only a few variables, it might be feasible to search for such interesting spaces manually, rotating the distribution of the data. With more than a few variables, however, it is best to let the computer loose to search by itself. To do this, we need to define what an "interesting" projection might look like, so that the computer knows when it has found one. Projection pursuit methods are based on this general principle of allowing the computer to search for interesting directions.

However, in one special case for one specific definition of what constitutes an "interesting" direction a computationally efficient explicit solution can be found. This is when we seek the projection onto the two-dimensional plane for which the sum of squared differences between the data points and their projections onto this plane is smaller than when any other plane is used. (We use two dimensional projections here for convenience, but in general we can use any k -dimensional projection, $1 \le k \le p - 1$). This two-dimensional plane can be shown to be spanned by (1) the linear combination of the variables that has maximum sample variance and (2) the linear combination that has maximum variance subject to being uncorrelated with the first linear combination. Thus "interesting" here is defined in terms of the maximum variability in the data.

Of course, we can take this process further, seeking additional linear combinations that maximize the variance subject to being uncorrelated with all those already selected. In general, if we are lucky, we find a set of just a few such linear combinations ("components") that describes the data fairly accurately. The mathematics of this process is described below. Our aim here is to capture the intrinsic variability in the data. This is a useful way of reducing the dimensionality of a data set, either to ease interpretation or as a way to avoid over fitting and to prepare for subsequent analysis.

Suppose that **X** is an n × p data matrix in which the rows represent the cases (each row is a data vector **x**(i)) and the columns represent the variables. Strictly speaking, the ith row of this matrix is actually the transpose $\mathbf{x}^T$ of the ith data vector **x**(i), since the convention is to consider data vectors as being p × 1 column vectors rather than 1 × p row vectors. In addition, assume that **X** is mean-centered so that the value of each variable is relative to the sample mean for that variable (i.e., the estimated mean has been subtracted from each column).

Let **a** be the p × 1 column vector of projection weights (unknown at this point) that result in the largest variance when the data **X** are projected along **a**. The projection of any particular data vector **x** is the linear combination $\mathbf{a}^T\mathbf{X} = \sum_{j=1}^{p} a_j \mathbf{x}_j$. Note that we can express the projected values onto **a** of all data vectors in **X** as **Xa** (n × p by p × 1, yielding an n × 1 column vector of projected values). Furthermore, we can define the variance along a as

$$\begin{aligned} \sigma_{\mathbf{a}}^2 &= \left(\mathbf{Xa}\right)^T \left(\mathbf{Xa}\right) \\ &= \mathbf{a}^T \mathbf{X}^T \mathbf{X} \mathbf{a} \\ &= \mathbf{a}^T V \mathbf{a}, \end{aligned}$$

where V = $\mathbf{X}^T\mathbf{X}$ is the p × p covariance matrix of the data (since **X** has zero mean), as defined in chapter 2. Thus, we can express $\sigma_{\mathbf{a}}^2$ (the variance of the projected data (a scalar) that we wish to maximize) as a function of both **a** and the covariance matrix of the data **V**.

Of course, maximizing $\sigma_{\mathbf{a}}^2$ directly is not well-defined, since we can increase $\sigma_{\mathbf{a}}^2$ without limit simply by increasing the size of the components of **a**. Some kind of constraint must be imposed, so we impose a normalization constraint on the **a** vectors such that $\mathbf{a}^T\mathbf{a} = 1$.

With this normalization constraint we can rewrite our optimization problem as that of maximizing the quantity

$$u = \mathbf{a}^T \mathbf{V} \mathbf{a} - \lambda(\mathbf{a}^T \mathbf{a} - 1),$$

where λ is a Lagrange multiplier. Differentiating with respect to a yields

$$\frac{\partial u}{\partial \mathbf{a}} = 2\mathbf{V}\mathbf{a} - 2\lambda\mathbf{a} = 0,$$

which reduces to the familiar eigenvalue form of

$$(\mathbf{V} - \lambda\mathbf{I})\mathbf{a} = 0.$$

Thus, the first principal component a is the eigenvector associated with the largest eigenvalue of the covariance matrix **V**. Furthermore, the second principal component (the direction orthogonal to the first component that has the largest projected variance) is the eigenvector corresponding to the second largest eigenvalue of **V**, and so on (the eigenvector for the k th largest eigenvalue corresponds to the k th principal component direction).

In practice of course we may be interested in projecting to more than two dimensions. A basic pro perty of this projection scheme is that if the data are projected into the first k eigenvectors, the variance of the projected data can be expressed as $\sum_{i=1}^{k} \lambda_j$, where λ_j is the jth eigenvalue. Equivalently, the squared error in terms of approximating the true data matrix X using only the first k eigenvectors can be expressed as

$$\frac{\sum_{j=k+1}^{p} \lambda_j}{\sum_{l=1}^{p} \lambda_l}.$$

Thus, in choosing an appropriate number k of principal components, one approach is to increase k until the squared error quantity above is smaller than some acceptable degree of squared error. For high -dimensional data sets, in which the variables are often relatively well -correlated, it is not uncommon for a relatively small number of principal components (say, 5 or 10) to capture 90% or more of the variance in the data.

A useful visual aid in this context is the scree plot which shows the amount of variance explained by each consecutive eigenvalue. This is necessarily nonincreasing with the number of the component, and the hope is that it demonstrates a sudden dramatic fall toward zero. A principal components analysis of the correlation matrix of the computer CPU data described earlier gives rise to eigenvalues proportional to 63.26, 10.70, 10.30, 6.68, 5.23, 2.18, 1.31, and 0.34 (see figure 3.15). The fall from the first to the second eigenvalue is dramatic, but after that the decline is gradual. (The weights that the first component puts on the eight variables are (0.199, -0.365, -0.399, -0.336, -0.331, - 0.298, -0.421, -0.423). Note that, it gives them all roughly similar weights, but gives the

NOTES

first variable (cycle time) a weight opposite in sign to those of the other variables.) If, instead of the correlation matrix, we analyzed the covariance matrix, the variables with larger ranges of values would tend to dominate. In the case of these data, the values given for memory are much larger than those for the other variables. (This is because they are given in kilobytes. Had they been given in megabytes, this would not be the case an example of the arbitrariness of the scaling of noncommensurate variables). Principal components analysis of the covariance matrix gives proportions of variation attributable to the different components as 96.02, 3.93, 0.04, 0.01, 0.00, 0.00, 0.00, and 0.00 (see figure 3.15). Here the fall from the first component is very striking the variability in the data can, indeed, be explained almost entirely by the differences in memory capacity. Often, however, there is no obvious fall such as this no point at which the remaining variance in the data can be attributed to random variation. Then the choice of how many components to extract is fairly arbitrary. The proportion of the total variance that we regard as providing an adequate simplified description of the data depends on the field of application. In some cases it might be sufficient for the first few components to describe 60% of the variance, but in other fields one might hope for 95% or more.

To illustrate the simple graphical use of principal components analysis, figure 3.16 shows the projections (indicated by the numbers) of 17 pills onto the space spanned by the first two principal components. The six measurements on each pill are the times at which a specified proportion (10%, 30%, 50%, 70%, 75%, and 90%) of the pill has dissolved. It is clear from this diagram that one of the pills is very different from the others, lying in the bottom right corner, far from the other points.

NOTES

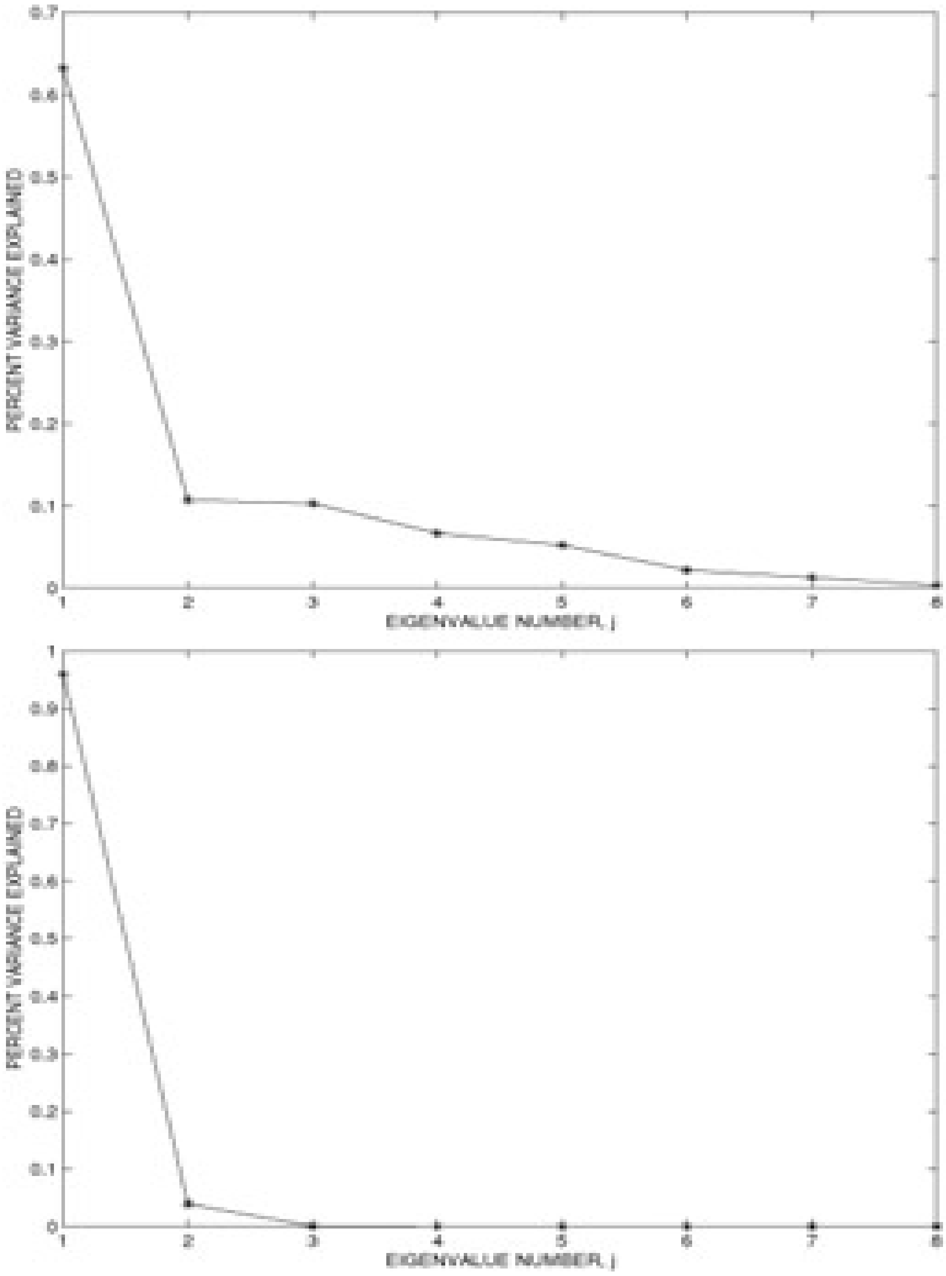


Figure 3.15: Scree plots for the computer cpu data set. the upper plot displays the eigenvalues from the correlation matrix, and the lower plot is for the covariance matrix.

Sometimes we can gain insights from the pattern of weights (or loadings, as they are sometimes called) defining the components of a principal components analysis. Huba et al, collected data on 1684 students in Los Angeles showing consumption of each of thirteen legal and illegal psychoactive substances: cigarettes, beer, wine, spirits, cocaine, tranquilizers, drug store medications used to get high, heroin and other opiates, marijuana, hashish,

NOTES

inhalants (such as glue), hallucinogenic, and amphetamines. They scored each as 1 (never tried), 2 (tried only once), 3 (tried a few times), 4 (tried many times), 5 (tried regularly). Taking these variables in order, the weights of the first component from a principal components analysis were (0.278, 0.286, 0.265, 0.318, 0.208, 0.293, 0.176, 0.202, 0.339, 0.329, 0.276, 0.248, 0.329). This component assigns roughly equal weights to each of the variables and can be regarded as a general measure of how often students use such substances. Thus, the biggest difference between the students is in terms of how often they use psychoactive substances, regardless of which substances they use.

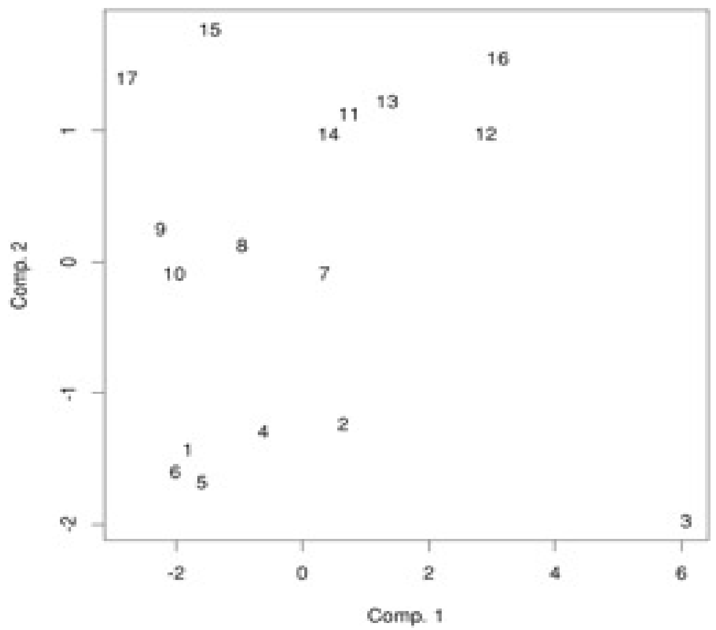


Figure 3.16: Projection onto the First Two Principal Components.

The second component had weights (0.280, 0.396, 0.392, 0.325, -0.288, -0.259, -0.189, -0.315, 0.163, -0.050, -0.169, -0.329, -0.232). This is interesting because it gives positive weights to the legal substances and negative weights to the illegal ones: therefore, once we have controlled for overall substance use, the major difference between the students lies in their use of legal versus illegal substances. This is just the sort of relationship one would hope to discover from a data mining exercise.

Another statistical technique, factor analysis, is often confused with principal components analysis, but the two have very different aims. As described above, principal components analysis is a transformation of the data to new variables. We can then select just some of these as providing an adequate description of the data. Factor analysis, on the other hand, is a model for data, based on the notion that we can define the measured variables $X_1, ..., X_p$ as linear combinations of a smaller number m (m < p) of "latent" (unobserved) factors variables that cannot be measured explicitly. The objective of factor analysis is to unearth information about these latent variables.

We can define $\mathbf{F} = (F_1, ..., F_m)^T$ as the m × 1 column vector of unknown latent variables, taking values $\mathbf{f} = (f_1, ..., f_m)$. Then a measured data vector $\mathbf{x} = (x_1, ..., x_p)^T$ (defined here as a p × 1 column vector) is regarded as a linear function of **f** defined by

$$X=\Lambda f+e.$$

Here Λ is a p × m matrix of factor loadings giving the weights with which each factor contributes to each manifest variable. The components of the p × 1 vector e are uncorrelated random variables, sometimes termed specific factors since they contribute only to single manifest (observed) variables, X_j, $1 \le j \le p$. Factor analysis is a special case of structural linear relational models described in chapter 9, so we will not dwell on estimation procedures here. However, since factor analysis was the earliest model structure of this form to be developed, it has a special place, not only because of its history, but also because it continues to be among the most widely used of such models.

Factor analysis has not had an entirely uncontroversial history, partly because its solutions are not invariant to various transformations. It is easy to see that new factors can be defined from equation via m × m orthogonal matrices **M**, such that $\mathbf{x} = (\Lambda \mathbf{M})(\mathbf{Mf}) + \mathbf{e}$. This corresponds to rotating the factors in the space they span. Thus, the extracted factors are essentially nonunique, unless extra constraints are imposed. There are various constraints in general use, including methods that seek to extract factors for which the weights are as close to 0 or 1 as possible, defining the variables as clearly as possible in terms of a subset of the factors.

3.7 Multidimensional Scaling

In the preceding section we described how to use principal components analysis to project a multivariate data set onto the plane in which the data has maximum dispersion. This allows us to examine the data visually, while sacrificing the minimum amount of information. Such a method is effective only to the extent that the data lie in a two- dimensional linear subspace of the area spanned by the measured variables. But what if the data forms a set that is intrinsically two-dimensional, but instead of being "flat," is curved or otherwise distorted in the space spanned by the original variables? (Imagine a crumpled piece of paper, intrinsically two dimensional, but occupying three dimensions.) In this event it is quite possible that principal components analysis might fail to detect the underlying two-dimensional structure. In such cases, multidimensional scaling can be helpful. Multidimensional scaling methods seek to represent data points in a lower dimensional space while preserving, as far as is possible, the distances between the data points. Since, we are mostly concerned with two-dimensional representations; we shall restrict most of our discussion to such cases. The extension to higher dimensional representations is immediate.

Many multidimensional scaling methods exist, differing in how they define the distances that are being preserved, the distances they map to, and how the calculations are performed. Principal components analysis may be regarded as a basic form. In this approach the distances between the data points are taken as Euclidean (or Pythagorean), and they are mapped to distances in a reduced space that are also measured using the Euclidean metric. The sum of squared distances between the original data points and their projections provides a measure of quality of the representation. Other methods of multidimensional scaling also have associated measures of the quality of the representation.

Since multidimensional scaling methods seek to preserve interpoint distances, such distances can serve as the starting point for an analysis. That is, we do not need to know any measured values of variables for the objects being analyzed, only how similar the objects are, in terms of some distance measure. For example, the data may have been collected by asking respondents to rate the similarity between pairs of objects. (A classic example of this is a matrix showing the number of times the Morse codes for different letters are confused. There are no "variables" here; simply a matrix of "similarities" measuring how often letter is was mistaken for another.) The end point of the process is the same a configuration of data points in a two -dimensional space. In a sense, the objects and the raters are used to determine on what dimensions "similarity" is to be measured. Multidimensional scaling methods are widely used in areas such as psychometrics and market research, in attempts to understand perceptions of relationships and similarities between objects.

NOTES

From an n × p data matrix **X** we can compute an n × n matrix **B** = $\mathbf{XX}^T$. (Since this scales as $O(n^2)$ in both time and memory, it is clear that this approach is not practical for very large numbers of objects n). It is straightforward to see from this that the Euclidean distance between the ith and jth objects is given by

$$d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$$

If we could invert this relationship, then, given a matrix of distances **D** (derived from original data points by computing Euclidean distances or obtained by other means), we could compute the elements of **B. B** could then be factorized to yield the coordinates of the points. One factorization of **B** would be in terms of the eigenvectors. If we chose those associated with the two largest eigenvalues, we would have a two -dimensional representation that preserved the structure of the data as well as possible.

The feasibility of this procedure hinges upon our ability to invert equation Unfortunately, this is not possible without imposing some extra constraints. Because shifting the mean and rotating a configurati on of points does not affect the interpoint distances, for any given a set of distances there is an infinite number of possible solutions, differing in the location and orientation of the point configuration.

A sufficient constraint to impose is the assumption that the means of all the variables are 0. That is, we assume $\sum_i x_{ik} = 0$ for all k = 1... p. This means that is, $\sum_i b_{ij} = \sum_j b_{ij} = 0$ Now, by summing equation first over i, then over j, and finally over both i and j, we obtain

$$\begin{aligned} \sum_i d_{ij}^2 &= tr(\mathbf{B}) + nb_{jj} \\ &= tr(\mathbf{B}) + nb_{ii} \\ &= 2ntr(\mathbf{B}) \end{aligned}$$

where tr(**B**) is the trac e of the matrix **B**. The third equation expresses tr(**B**) in terms of the d_{ij}^2 the first and second express b_{jj} and b_{ii} in terms of d_{ii}^2 and tr(**B**), and hence in terms of d_{ij}^2 alone. Plugging these into equation expresses b_{ij} as a function of d_{ij}^2, yielding the required inversion. This process is known as the

NOTES

principal coordinates method. It can be shown that the scores on the components calculated from a principal components analysis of a data matrix **X** (and hence a factorization of the matrix $\mathbf{X}^T$) are the same as the coordinates of the above scaling analysis.

If course, if the matrix **B** does not arise not as a product $\mathbf{XX}^T$, but by some other route (such as simple subjective differences between pairs of objects), then there is no guarantee that all the eigenvalu es will be non-negative. If the negative eigenvalues are small in absolute value, they can be ignored.

Classical multidimensional scaling into two dimensions finds the projection into two dimensions that is most accurate in the sense that it minimizes

$$\sum_i \sum_j (\delta_{ij} - d_{ij})^2,$$

where d_{ij} is the observed distance between points i and j in the p-dimensional space and d_{ij} is the distance between the points representing these objects in the two-dimensional space. Expressed this way the process permits ready generalization. Given distances or dissimilarities, derived in one way or another, we can seek a distribution of points in a two-dimensional space that minimizes the sum of squared differences $\sum_i \sum_j (d_{ij} - d_{ij})^2$. Thus, we relax the restriction that the configuration must be found by projection. With this relaxation an exact algebraic solution will generally not be possible, so numerical methods must be used: we simply have a function of 2n parameters (the coordinates of the points in the two-dimensional space) that is to be minimized.

The score function $\sum_i \sum_j (d_{ij} - d_{ij})^2$, measuring how well the interposing distances in the derived configuration match those originally provided, is invariant with respect to rotations and translations. However, it is not invariant to rescaling: if the d_{ij} were multiplied by a constant, we would end up with the same solution, but a different value of $\sum_i \sum_j (d_{ij} - d_{ij})^2$. To permit different situations to be properly compared we divide $\sum_I \sum_j (d_{ij} - d_{ij})^2$

NOTES

by, $\sum_{i,j} d_{ij}^2$ yielding the standardized residual sum of squares. A common by score function is the square root of this quantity, the stress. A variant on the stress is the sstress, defined as

$$\sqrt{\sum_i \sum_j (\delta_{ij}^2 - d_{ij}^2)^2 / \sum_i \sum_j d_{ij}^4}.$$

These measures effectively assume that the differences between the original dissimilarities and the distances in the two dimensional configuration are due to random discrepancies and arbitrary distortions that is, that $d_{ij} = \delta_{ij} + d_{ij}$. More sophisticated models can also be built. For example, we might assume that $d_{ij} = a + b\delta_{ij} + d_{ij}$. Now a two-stage procedure is necessary. Beginning with a proposed configuration, we regress the distances d_{ij} in the two - dimensional space on the given dissimilarities, yielding estimates for a and b. We then find new values of the d_{ij} that minimize the stress

$$\sqrt{\sum_i \sum_j (d_{ij} - a - b\delta_{ij})^2 / \sum_i \sum_j d_{ij}^2},$$

and repeat this process until we achieve satisfactory convergence. Multidimensional scaling methods such as the above, which attempt to model the dissimilarities as given, are called metric methods. Sometimes, however, a more general approach is required. For example, we may not be given the precise similarities, only their rank order (objects A and B are more similar than B and C, and so on); or we may not be prepared to assume that the relationship between d_{ij} and d_{ij} has a particular form, just that some monotonic relationship exists. This requires a two -stage approach similar to that described in the preceding paragraph, but with a technique known as monotonic regression replacing simple linear regression, yielding non-metric multidimensional scaling. The term non-metric here indicates that the method seeks to preserve only ordinal relationships.

Multidimensional scaling is a powerful method for displaying data to reveal structure. However, as with the other graphical methods described in this chapter, if there are too many data points the structure becomes obscured. Moreover, since multidimensional scaling involves applying highly sophisticated transformations to the data (more so than a simple scatterplot or principal components analysis) there is a possibility that artifacts may be

NOTES

introduced. In particular, in some situations the dissimilarities between objects can be determined more accurately when the objects are similar than when they are quite different. Consider the evolution of the style of a manufactured object. Those objects that are produced within a short time of each other will probably have much in common, while those separated by a greater time gap may have very little in common. The consequence will be an induced curvature in the multidimensional scaling plot, where we might have hoped to achieve a more or less straight line. This phenomenon is known as the horseshoe effect.

Figure 3.17 shows a plot produced using non-metric scaling to minimize the sstress score function of equation 3.15. The data arose from a study of English dialects. Each pair of a group of 25 villages was rated according to the percentages of 60 items for which the villages used different words. The villages, and the counties in which they are located, are listed in table 3.1. The figure shows that villages from the same county (and hence that are relatively close geographically) tend to use the same words.

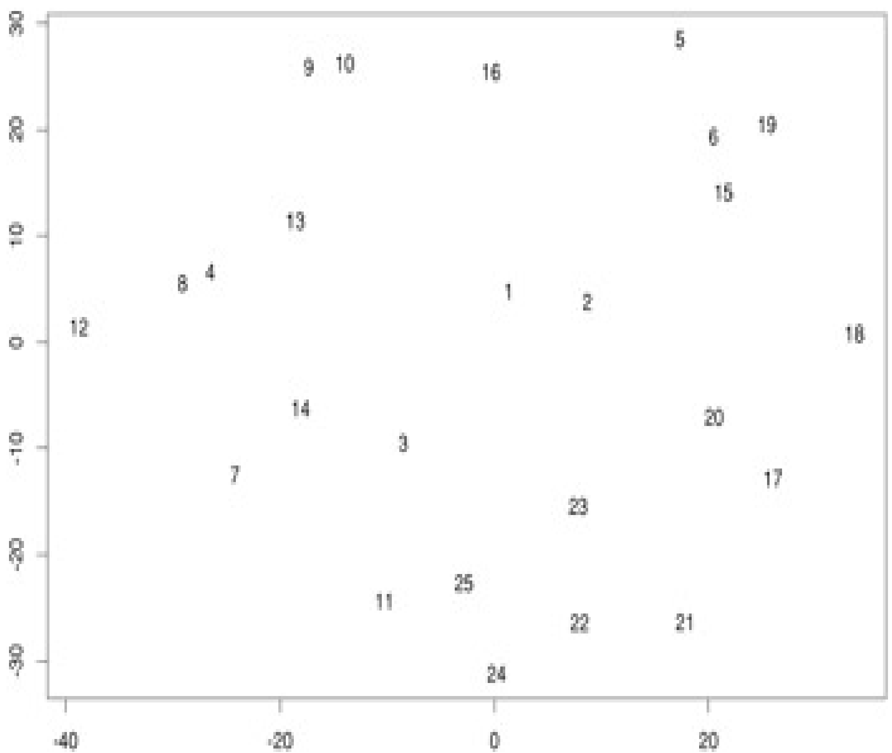


Figure 3.17: A Multidimensional Scaling Plot of the Village Dialect Similarities Data.

1	North Wheatley	Nottinghamshire
2	South Clifton	Nottinghamshire
3	Oxton	Nottinghamshire
4	Eastoft	Lincolnshire
5	Keelby	Lincolnshire
6	Wiloughton	Lincolnshire
7	Wragby	Lincolnshire
8	Old Bolingbroke	Lincolnshire
9	Fulbeck	Lincolnshire
10	Sutterton	Lincolnshire
11	Swinstead	Lincolnshire
12	Crowland	Lincolnshire
13	Harby	Leicestershire
14	Packington	Leicestershire
15	Goadby	Leicestershire
16	Ullesthorpe	Leicestershire
17	Empingham	Rutland

Table 3.1: Numerical Codes, Names, And Counties for the 25 Villages with Dialect Similarities Displayed in Figure 3.17.

Multidimensional scaling methods typically display the data points in a two-dimensional space. If the variables are also described in this space (provided the data are in vector form) the relationships between data points and variables may be clearly seen. Given the complicated nonlinear relationship between the space defined by the original variables and the space used to display the data, representing the original variables is a non -trivial task. Plots that display both data points and variables are known as biplots. The "bi" here signifies that there are two modes being displayed the points and the variables not that the display is two-dimensional. Indeed, three -dimensional biplots have also been developed. Forms of multidimensional scaling that involve nonlinear transformations produce nonlinear biplots. Biplots have even been produced for categorical data, and in this case the levels of the variables are represented by regions in the plot. Effective interpretation of multidimensional and biplot display s requires practice and experience.

4. Data Analysis and Uncertainty

Structure

Objectives

After going through this lesson, you should be able to:

- Discuss about dealing with uncertainty;
- Discuss about random variables and their relationships;
- Discuss about estimation and hypothesis testing;

4.1 Introduction

In this lesson, we focus on uncertainty and how to cope with it. Not only is the process of mapping from the real world to our databases seldom perfect, but the domain of the mapping the real world itself is beset with ambiguities and uncertainties. The basic tool for dealing with uncertainty is probability, and we begin by defining the concept and showing how it is used to construct statistical models.

NOTES

4.2 Dealing with Uncertainty

The ubiquity of the idea of uncertainty is illustrated by the rich variety of words used to describe it and related concepts. Probability, chance, randomness, luck, hazard, and fate are just a few examples. The omnipresence of uncertainty requires us to be able to cope with it: modeling uncertainty is a necessary component of almost all data analysis. Indeed, in some cases our primary aim is to model the uncertain or random aspects of data. It is one of the great achievements of science that we have developed a deep and powerful understanding of uncertainty. The capricious gods that were previously invoked to explain the lack of predictability in the world have been replaced by mathematical, statistical, and computer-based models that allow us to understand and manipulate uncertain events. We can even attempt the seemingly impossible and predict uncertain events, where prediction for a data miner either can mean the prediction of future events (where the notion of uncertainty is very familiar) or prediction in a nontemporal sense of a variable whose true value is somehow hidden from us (for example, diagnosing whether a person has cancer, based on only descriptive symptoms).

We may be uncertain for various reasons. Our data may be only a sample from the population we wish to study, so that we are uncertain about the extent to which different samples differ from each other and from the overall population. Perhaps our interest lies in making a prediction about tomorrow, based on the data we have today, so that our conclusions are subject to uncertainty about what the future will bring. Perhaps we are ignorant and cannot observe some value, and have to base our ideas on our "best guess" about it. And so on.

Many conceptual bases have been formulated for handling uncertainty and ignorance. Of these, by far the most widely used is probability. Fuzzy logic is another that has a moderately large following, but this area along with closely related areas such as possibility theory and rough sets remains rather controversial: it lacks the sound theoretical backbone and widespread application and acceptance of probability. These ideas may one day develop solid foundations, and become widely used, but because of their current uncertain status we will not consider them further in this book.

It is useful to distinguish between probability theory and probability calculus. The former is concerned with the interpretation of probability while the latter is concerned with the manipulation of

NOTES

the mathematical representation of probability. (Unfortunately, not all textbooks make this distinction between the two terms often books on probability calculus are given titles such as "Introduction to the Theory of Probability.") The distinction is an important one because it permits the separation of those areas about which there is universal agreement (the calculus) from those areas about which opinions differ (the theory). The calculus is a branch of mathematics, based on well-defined and generally accepted axioms (stated by the Russian mathematician Kolmogorov in the 1930s); the aim is to explore the consequences of those axioms. (There are some areas in which different sets of axioms are used, but these are rather specialized and generally do not impinge on problems of data mining.) The theory, on the other hand, leaves scope for perspectives on the mapping from the real world to the mathematical representation i.e., on what probability is.

A study of the history and philosophy of probability theory reveals that there are as many perspectives on the meaning of probability as there are thinkers. However, the views can be grouped into variants of a few different types. Here we shall restrict ourselves to discussing the two most important types (in terms of their impact on data mining practice).

The frequents view of probability takes the perspective that probability is an objective concept. In particular, the probability of an event is defined as the limiting proportion of times that the event would occur in repetitions of essentially identical situations. A simple example is the proportion of times a head comes up in repeatedly tossing a coin. This interpretation restricts our application of probability: for instance we cannot assess the probability that a particular athlete will win a medal in the next Olympics because this is a one-off event, where the notion of a "limiting proportion" makes no sense. On the other hand, we can certainly assess the probability that a customer in a supermarket will purchase a certain item, since we can use a large number of similar customers as the basis for a limiting proportion argument. It is clear in this last example that some idealization is going on: different customers are not really the same as repetitions of a single customer. As in all scientific modeling we need to decide what aspects are important for our model to be sufficiently accurate. In predicting customer behavior we might decide that the differences between customers do not matter.

The frequentist view was the dominant perspective on probability throughout most of the last century, and hence it underpins most widely used statistical software. However, in the last decade or so, a competing view has acquired increasing importance. This view, that of subjective probability, has been around since people first started formalizing probabilistic notions, but until recently it was primarily of theoretical interest. What revived the approach was the development of the computer and of powerful algorithms for manipulating and processing subjective probabilities. The principles and methodologies for data analysis that derive from the subjective point of view are often referred to as Bayesian statistics. A central tenet of Bayesian statistics is the explicit characterization of all forms of uncertainty in a data analysis problem, including uncertainty about any parameters we estimate from the data, uncertainty as to which among a set of model structures are best or closest to "truth," uncertainty in any forecast we might make, and so on. Subjective probability is a very flexible framework for modeling such uncertainty in different forms.

4.3 Random Variables and Their Relationships

A random variable is a mapping from a property of objects to a variable that can take one of a set of possible values, via a process that appears to the observer to have some element of unpredictability to it. The possible values of a random variable X are called the domain of X. We use uppercase letters such as X to refer to a random variable and lowercase letters such as x to refer to a value of a random variable.

An example of a random variable is the outcome of a coin toss (the domain is the set {heads, tails}). Less obvious examples of random variables include the number of times we have to toss a coin to obtain the first head (the domain is the set of positive integers) and the flying time of a paper aeroplane in seconds (the domain is the set of positive real numbers).

4.3.1 Multivariate Random Variables

Since data mining often deals with multiple variables, we must also introduce the concept of a multivariat e random variable. A multivariate random variable **X** is a set $X_1,..., X_p$ of random variables. We use the m-dimensional vector **x** = $\{x_1, ..., x_p\}$ to denote a set of values for **X**. The density function $f(\mathbf{X})$ of the multivariate random variable **X** is called the joint density function of **X**. We denote this as $f(\mathbf{X}) = f(X_1 = x_1, ..., X_p = x_p)$, or simply $f(x_1, ..., x_p)$. Similarly, we have joint probability distributions for variable staking values in a finite set. Note that $f(\mathbf{X})$ is a scalar function of p variables.

The density function of any single variable in the set **X** (or, more generally, any subset of the complete set of variables) is called a marginal density of the joint density. Technically, it is derived from the joint density by summing or integrating across the variables not included in the subset. For example, for a tri-variate random variable **X** = (X_1, X_2, X_3) the marginal density of $f(X_1)$ is given by $f(x_1) = \iint f(x_1, x_2, x_3)dx_2\, dx_3$.

The density of a single variable (or a subset of the complete set of variables) given (or "conditional on") particular values of the other variables is a conditional density. Thus we can speak of the conditional density of vari able X_1 given that X_2 takes the value 6, denoted $f(x_1 \mid x_2 = 6)$. In general, the conditional density of X_1 given some value of X_2 is denoted by $f(x_1 \mid x_2)$, and is defined as

$$f(x_1 \mid x_2) = \frac{f(x_1, x_2)}{f(x_2)}.$$

For discrete-valued random variables we have equivalent definitions ($p(a_1 \mid a_2)$, etc.). We can also use mixtures of the two e.g., a conditional probability density function $f(x1 \mid a_1)$ for a continuous variable conditioned on a categorical variable, and a conditional probability mass function $p(a_1 \mid x_1)$ for the reverse case.

Note that particular variables in the multivariate set **X** may well be related to each other in some manner. Indeed, a generic problem in data mining is to find relationships between variables. Is purchasing item A likely to be related to purchasing item B? Is detection of pattern A in the trace of a measuring instrument likely to be followed shortly afterward by a particular fault? Variables are said to be independent if there is no relationship between the occurrences of values of the variables; otherwise they are dependent. More formally, variables X and Y are independent if and only if $p(x, y) = p(x)p(y)$ for all values of X and Y . An equivalent formulation is that X and Y are independent if and only if $p(x \mid y) = p(x)$ or $p(y \mid x) = p(y)$ for all values of X and Y . (Note that these definitions hold whether each p in the expression is a probability mass function or a density function in the latter case the variables are independent if and only if $f(x, y) = f(x)f(y)$). The second form of the definition shows that when X and Y are independent the distribution of X is the same whether or not the value of Y is known. Thus, Y carries no information about X, in the sense that the value taken by Y does not influence the probability of X taking any value.

We can generalize these ideas to more than two variables. For example, we say that X is conditionally independent of Y given Z if for all values of X, Y, and Z we have that p(x, y | z) = p(x | z)p(y | z), or equivalently p(x | y, z) = p(x | z). To illustrate, suppose a person purchases bread (so that a random variable Z takes the value 1). Then subsequent purchases of butter (random variable X takes the value 1) and cheese (random variable Y takes the value 1) might be modeled as being conditionally independent the probability of purchasing cheese is unaffected by whether or not butter was purchased, once we know that bread has been purchased.

Note that conditional independence need not imply marginal (unconditional) independence. That is, the conditional independence relations above do not imply p(x, y) = p(x)p(y). For example, in our illustration we might reasonably expect purchases of butter and cheese to be dependent in general (since they are both dependent on bread purchases). The reverse also applies: X and Y may be (unconditionally) independent, but conditionally dependent given a third variable Z. The subtleties of these dependence and independence relations have important consequences for data miners. In particular, even though two observed variables (such as butter and cheese) may appear to be dependent given the data, their true relationship may be masked by a third (potentially unobserved) variable (such as bread in our illustration).

The assumption of conditional independence is widely used in the context of sequential data, for which the next value in the sequence is often independent of all of the past values in the sequence given only the current value in the sequence. In this context, conditional independence is known as the first-order Markov property.

The notions of independence and conditional independence (which can be viewed as a generalization of independence) are central to many of the key concepts in data analysis, as we shall see in later chapters. The assumptions of independence and conditional independence enable us to factor the joint densities of many variables into much more tractable products of simpler densities, e.g.,

$$f(x_1, \ldots, x_n) = f(x_1) \prod_{j=2}^{n} f(x_j | x_{j-1}),$$

where each variable x_j is conditionally independent of variables x_1,

..., x_{j-2}, given the value of x_j (this is an example of a first-order Markov model). In addition to the computational benefits provided by such simplifications, it also provides important modeling gains by allowing us to construct more understandable models with fewer parameters. Nonetheless, independence is a very strong assumption that is frequently violated in practice (for example, assuming sequences of letters in text are first-order Markov may not be realistic). Still, keeping in mind that our models are inevitably approximations to the real world, the benefits of appropriate independence assumptions often outweigh the alternative of building more complex but less stable models.

A special case of dependency is correlation or linear dependency (Note that statistical dependence is not the same as correlation: two variables may be dependent but not linearly correlated). Variables are said to be positively correlated if high values of one variable tend to be associated with high values of the other, and to be negatively correlated if high values of one tend to be associated with low values of the other. It is important not to confuse correlation with causation. Two variables may be highly positively correlated without any causal relationship between them. For example, yellow-stained fingers and lung cancer may be correlated, but are causally linked only via a third variable, namely whether a person smokes or not. Similarly, human reaction time and earned income may be negatively correlated, but this does not mean that one causes the other. In this case a more convincing explanation is that a third variable, age, is causally related to both of these variables.

4.4 Samples and Statistical Inference

As many data mining problems involve the entire population of interest, while others involve just a sample from this population. In the latter case, the samples may arise at the start perhaps only a sample of tax-payers is selected for detailed investigation; perhaps a complete census of the population is carried out only occasionally, with just a sample being selected in most years; or perhaps the data set consists of market research results. In other cases, even though the complete data set is available, the data mining operation is carried out on a sample. This is entirely legitimate if the aim is modeling, which seeks to represent the prominent structures of the data, and not small idiosyncratic deviations. Such structures will be preserved in a sample, provided it is not too small. However, working with a small sample of a large data set may be less appropriate if the aim is pattern detection: in this case the aim may be to detect small deviations from the bulk

NOTES

of the data, and if the sample is too small such deviations may be excluded. Moreover, if the aim is to detect records that show anomalous behavior, the analysis must be based on the entire sample.

It is when a sample is used that the power of inferential statistics comes into play. Statistical inference allows us to make statements about population structures, to estimate the size of these structures, and to state our degree of confidence in them, all on the basis of a sample. (See figure 4.1 for a simple illustration of the roles of probability and statistics). Thus, for example, we could say that our best estimate of a population value is 6.3, and that one is 95% confident that the true population value lies between 5.9 and 6.7. (Definition and interpretation of intervals such as these is a delicate point, and depends on what philosophical basis we adopt frequentist or Bayesian, for example. We shall say more about such intervals later in this chapter.) Note the use of the word estimate for the population value here. If we were basing our analysis on the entire population, we would use the word calculate: if all the constituent numbers are known, we can actually calculate the population value, and no notion of estimation arises.

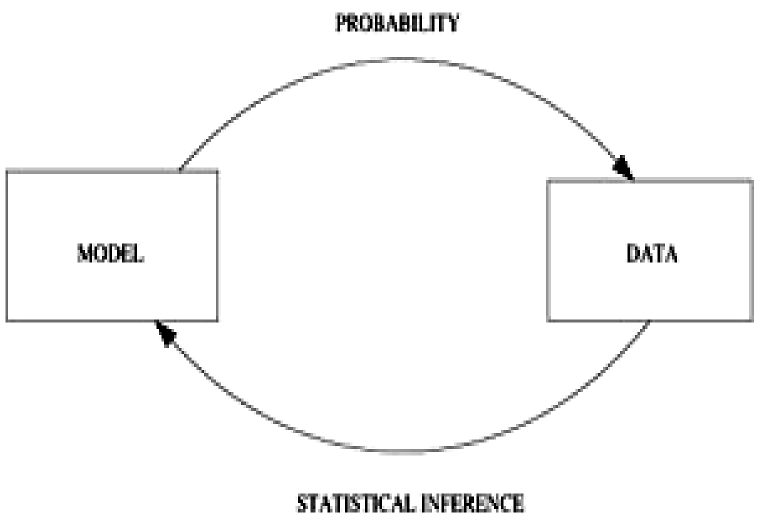


Figure 4.1: An illustration of the dual roles of probability and statistics in data analysis. probability specifies how observed data can be generated from models. statistical inference allows us to infer models from observed data.

In order to make an inference about a population structure, we must have a model or pattern structure in mind: we would not be able to assess the evidence for some structure underlying the data if we never contemplated the existence of such a structure. So, for example, we might hypothesize that the value of some variable Z depends on the values of two other variables X and Y . Our model is that Z is related to X and Y . Then we can estimate the strength of these relationships in the population. (Of course, we may conclude that one or both of the relationships are of strength zero that there is no relationship.)

Statistical inference is based on the premise that the sample has been drawn from the population in a random manner that each member of the population had a particular probability of appearing in the sample. The model will specify the distribution function for the population the probability that a particular value for the random variable will arise in the sample. For example, if the model indicates that the data have arisen from a Normal distribution with a mean of 0 and a standard deviation of 1, it also tells us that the probability of observing a value as large as +20 is very small. Indeed, under the assumption that the model is correct, a precise probability can be put on observing a value greater than +20. Given the model, we can generally compute the probability that an observation will fall within any interval. For samples from categorical distributions, we can estimate the probability that values equal to each of the observed values would have arisen. In general, if we have a model M for the data we can state the probability that a random sampling process would lead to the data D = {**x**(1), ..., **x**(n)}, here **x**(i) is the ith p- dimensional vector of measurements (the ith row in our n × p data matrix). This probability is expressed as p(D | M). Often we do not make dependence on the model M explicit and simply write p(D), relying on the context to make it clear. Let p(**x**(i)) be the probability of individual i having vector measurement **x**(i) (here p could be a probability mass function or a density function, depending on the nature of **x**). If we further assume that the probability of each member of the population being selected for inclusion in the sample has no effect on the probability of other members being selected (that is, that the separate observations are independent, or that the data are drawn "at random"), the overall probability of observing the entire distribution of values in the sample is simply the product of the individual probabilities:

$$p(D \mid \theta, M) = \prod_{i=1}^{n} p\left(\mathbf{x}(i) \mid \theta, M\right),$$

where M is the model and ө are the parameters of the model (assumed fixed at this point). (When regarded as a function of the parameters ө in the model M, this is called the likelihood function. We discuss it in detail below.) Methods have been developed to cope with situations in which observing one value alters the chance of observing another, but independence is by far the most commonly used assumption, even when it is only approximately true.

4.5 Estimation

Inseveral techniques for summarizing a given set of data. When we are concerned with inference, we want to make more general statements, statements about the entire population of values that might have been drawn. These are statements about the probability distribution or probability density function (or, equivalently, about the cumulative distribution function) from which the data are assumed to have arisen.

4.5.1 Desirable Properties of Estimators

In the following subsections we describe the two most important methods of estimating the parameters of a model: maximum likelihood estimation and Bayesian estimation. It is important to be aware of the differing properties of different methods so that we can adopt a method suited to our problem. Here we briefly describe some attractive properties of estimators. Let $\hat{\theta}$ be an estimator of a parameter θ. Since $\hat{\theta}$ is a number derived from the data, if we were to draw a different sample of data, we would obtain a different value for $\hat{\theta}$. Thus, $\hat{\theta}$ is a random variable. Therefore, it has a distribution, with different values arising as different samples are drawn. We can obtain descriptive summaries of that distribution. It will, for example, have a mean or expected value,E[$\hat{\theta}$]. Here the expectation function E is taken with respect to the true (unknown) distribution from which the data are assumed to be sampled that is, over all possible data sets of size n that could occur weighted by their probability of occurrence.

The bias of is defined as Bias(θ) = E[$\hat{\theta}$] - θ the difference between the expected value of the estimator E[$\hat{\theta}$] and the true value of the parameter θ. Estimators for which E[$\hat{\theta}$] = θ have bias 0 are said to be unbiased. Such estimators show no systematic departure from the true parameter value on average, although for any particular single data set D we might have that $\hat{\theta}$ is far away from θ. Note that since both the sampling distribution and the true value of θ are unknown in practice, we cannot typically calculate the actual bias for a given data set. Nonetheless, the general concept of bias (and variance, below) is of fundamental importance in estimation.

NOTES

Just as the bias of an estimator can be used as a measure of its quality, so also can its variance:

$$Var(\hat{\theta}) = E[\hat{\theta} - E[\hat{\theta}]]^2.$$

The variance measures the random, data-driven component of error in our estimation procedure; it reflects how sensitive our estimator will be to the idiosyncrasies of individual data sets. Note that the variance does not depend on the true value of θ it simply measures how much our estimates will vary across different observed data sets. Thus, although the true sampling distribution is unknown, we can in principle get a data- driven estimate of the variance of an estimator, for a given value of n, by repeatedly subsampling our original data set and calculating the variance of the estimated $\hat{\theta}$s across these simulated samples. We can choose between estimators that have the same bias by choosing one with minimum variance. Unbiased estimators that have minimum variance are called, unsurprisingly, best unbiased estimators.

As an extreme example, if we were to completely ignore our data D and simply say arbitrarily that $\hat{\theta} = 1$ for every data set, then var($\hat{\theta}$) is zero since the estimate never changes as D changes however this would be a very the estimate ineffective estimator in practice since unless we made a very lucky guess we are almost certainly wrong in our estimate of ɵ, i.e., there will be a non -zero (and potentially very large) bias.

The mean squared error of $\hat{\theta}$ is $E[(\hat{\theta} - \theta)^2]$ the mean of the squared difference between the value of the estimator and the true value of the parameter. Mean squared error has a natural decomposition as the sum of the squared bias of $\hat{\theta}$ and its variance:

$$\begin{aligned} E\left[(\hat{\theta} - \theta)^2\right] &= E\left[(\hat{\theta} - E[\hat{\theta}] + E[\hat{\theta}] - \theta)^2\right] \\ &= \left(E[\hat{\theta}] - \theta\right)^2 + E\left[(\hat{\theta} - E[\hat{\theta}])^2\right] \\ &= \left(Bias(\hat{\theta})\right)^2 + Var(\hat{\theta}), \end{aligned}$$

where in going from the first to second lines above we took advantage of the fact that various cross-terms in the squared expression cancel out, noting (for example) that E[θ]=θ since θ is a constant, etc. Mean squared error is a very useful criterion since it incorporates both systematic (bias) and random (variance) differences between the estimated and true values. (Of course it too is primarily of theoretical interest, since to calculate it we need to know θ, which we don't in practice). Unfortunately, bias and variance often work in different directions: modifying an estimator to reduce its bias increases its variance, and vice versa. The trick is to arrive at the best compromise.

There are also more subtle aspects to the use of mean squared error in estimation. For example, mean squared error treats equally large departures from θ as equally serious, regardless of whether they are above or below θ. This is appropriate for measures of location, but may not be appropriate for measures of dispersion (which, by definition, have a lower bound of zero) or for estimates of probabilities or probability densities.

Suppose that we have sequence $\hat{\theta}_{n1}, \ldots \hat{\theta}_{nm}$ of estimators, based on increasing sample sizes n_1, ..., n_m. The sequence is said to be consistent if the probability of the difference between $\hat{\theta}$ and the true value θ being greater than any given value tends to 0 as the sample size increases. This is clearly an attractive property (especially in data mining contexts, with large samples) since the larger the sample is the closer the estimator is likely to be to the true value.

4.5.2 Maximum Likelihood Estimation

Maximum likelihood estimation is the most widely used method of parameter estimation. Consider a data set of n observations D = {**x**, ..., **x**(n)}, independently sampled from the same distribution $f(\mathbf{x} \mid \theta)$ (as statisticians say, independently and identically distributed or iid). The likelihood function $L(\theta \mid \mathbf{x}(1), \ldots, \mathbf{x}(n))$ is the probability that the data would have arisen, for a given value of θ, regarded as a function of θ, i.e., $p(D \mid \theta)$. Note that although we are implicitly assuming a particular model M here, as defined by $f(\mathbf{x} \mid \theta)$, for convenience we do not explicitly condition on M in our likelihood definitions below later, when we consider multiple models we will need to explicitly keep track of which model we are talking about.

Since we have assumed that the observations are independent we have

$$\begin{aligned} L(\theta \mid D) &= L(\theta \mid \mathbf{x}(1), \ldots, \mathbf{x}(n)) \\ &= p(\mathbf{x}(1), \ldots, \mathbf{x}(n) \mid \theta) \\ &= \prod_{i=1}^{n} f(\mathbf{x}(i) \mid \theta), \end{aligned}$$

which is a scalar function of θ (where θ itself may be a vector of parameters rather than a single parameter). The likelihood of a data set $L(\theta \mid D)$, the probability of the actual observed data D for a particular model, is a fundamental concept in data analysis. Defining a likelihood for a given problem amounts to specifying a probabilistic model for how the data were generated. It turns out that once we can state such a likelihood, the door is opened to the application of many general and powerful ideas from statistical inference. Note that since likelihood is defined as a function of θ the convention is that we can drop or ignore any terms in $p(D \mid \theta)$ that do not contain θ, i.e., likelihood is only defined within an arbitrary scaling constant, so it is the shape as a function of θ that matters and not the actual values that it takes. Note also that the idd assumption above is not necessary to define a likelihood: for example, if our n observations had a Markov dependence (where each **x**(i) depends on **x**(i - 1), we would define the likelihood as a product of terms such as $f(\mathbf{x}(i) \mid \mathbf{x}(i-1), \theta)$.

The value for θ for which the data has the highest probability of having arisen is the maximum likelihood estimator (or MLE). We will denote the maximum likelihood estimator for θ as $\hat{\theta}_{ML}$.

Maximum likelihood estimators are intuitively and mathematically attractive; for example, they are consistent estimators in the sense defined earlier. Moreover, if $\hat{\theta}_{ML}$ is the MLE of a parameter θ, then $g(\hat{\theta}_{ML})$ is the MLE of the function g(θ), though some care needs to be exercised if g is not a one -to-one function. On the other hand, nothing is perfect maximum likelihood estimators are often biased (depending on the parameter and the underlying model), although this bias may be extremely small for large data sets, often scaling as O(1/n).

For simple problems (where "simple" refers to the mathematical structure of the problem, and not to the number of data points, which can be large), MLEs can be found using differential calculus. In practice, the log-likelihood l(θ) is usually maximized (as in the Binomial and Normal density examples above), since this replaces the awkward product in the definition with a sum; this process leads to the same result as maximizing L(θ) directly because the logarithm is a monotonic function. Of course we are often interested in models that have more than one parameter (models such as neural networks can have hundreds or thousands of parameters). The univariate definition of likelihood generalizes directly to the multivariate case, but in this situation the likelihood is a mulutivariate function of d parameters (that is, a scalar-valued function defined on a d-dimensional parameter space). Since d can be large, finding the maximum of this d- dimensional function can be quite challenging if no closed -form solution exists. Multiple maxima can present a difficult problem (which is why stochastic optimization methods are often necessary), as can situations in which optima occur at the boundaries of the parameter space.

Up to this now we have been discussing point estimates, single number estimates of the parameter in question. A point estimate is "best" in some sense, but it conveys no idea of the uncertainty associated with it perhaps there was a large number of almost equally good estimates, or perhaps this estimate was by far the best. Interval estimates provide this sort of information. In place of a single number they give an interval with a specified degree of confidence that this interval contains the unknown parameter. Such an interval is called a confidence interval, and the upper and lower limits of the interval are called confidence limits. Interpretation of confidence intervals is rather subtle. Here, since we are assuming that ɵ is unknown but fixed, it does not make sense to say that ɵ has a certain probability of lying within a given interval: it either does or it does not. However, it does make sense to say that an interval calculated by the given procedure contains ɵ with a certain probability: after all, the interval is calculated from the sample, and is thus a random variable.

4.5.3 Bayesian Estimation

In the frequentist approach to inference described so far the parameters of a population are fixed but unknown, and the data comprise a random sample from that population (since the sample was drawn in a random way). The intrinsic variability thus lies in the data D = {**x**(1), ..., **x**(n)}. In contrast, Bayesian statistics treats the data as known after all, they have been observed and recorded and the parameters θ as random variables. Thus, whereas frequentists regard a parameter θ as a fixed but unknown quantity, Bayesians regard θ as having a distribution of possible values and see the observed data as possibly shedding light on this distribution. p(θ) reflects our degree of belief on where the true (unknown) parameters θ may be. If p(θ) is very peaked about some value of θ then we are very sure about our convictions (although of course we may be entirely wrong!). If p(θ) is very broad and flat (and this is the more typical case) then we are expressing a prior belief that is less certain on the location of θ.

NOTES

Note that while the term Bayesian has a fairly precise meaning in statistics, it has sometimes been used in a somewhat looser manner in the computer science and pattern recognition literature to refer to the use of any form of probabilistic model in data analysis. In this text we adopt the more standard and widespread statistical definition, which is described below.

Before the data are analyzed, the distribution of the probabilities that θ will take different values is known as the prior distribution p(θ). Analysis of the data D leads to modification of this distribution to take into account the information in the empirical data, yielding the posterior distribution, p(θ | D). The modification from prior to posterior is carried out by means of a theorem named after Thomas Bayes:

$$p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{p(D)} = \frac{p(D \mid \theta)p(\theta)}{\int_{\psi} p(D \mid \psi)p(\psi)d\psi}.$$

Note that this updating procedure leads to a distribution, rather than a single value, for θ However, the distribution can be used to yield a single value estimate. We could, for example, take the mean of the posterior distribution, or its mode (the latter technique is known as the maximum a posteriori method, or MAP). If we choose the prior p(θ) in a specific manner (e.g., p(θ) is uniform over some range), the MAP and maximum likelihood estimates of θ may well coincide (since in effect the prior is "flat" and prefers no one value of θ over any other). In this sense, maximum likelihood can be viewed as a special case of the MAP procedure, which in turn is a restricted ("point estimate") form of Bayesian estimation.

For a given set of data D and a particular model, the denominator in above equation is a constant, so we can alternatively write the expression as

$$p(\theta \mid D) \propto p(D \mid \theta)p(\theta).$$

Here we see that the posterior distribution of θ given D (that is, the distribution conditional on having observed the data D) is proportional to the product of the prior p(θ) and the likelihood p(D | θ). If we have only weak beliefs about the likely value of the parameter before collecting the data, we will want to choose a prior that spreads the probability widely (for example, a Normal distribution with large variance). In any case, the larger the set of observed data, the more the likelihood dominates the posterior distribution, and the lower the importance of the particular shape of the prior.

One of the primary distinguishing characteristics of the Bayesian approach is the avoidance of so-called point estimates (such as a maximum likelihood estimate of a parameter) in favor of retaining full knowledge of all uncertainty involved in a problem (e.g., calculating a full posterior distribution on θ).

As an example, consider the Bayesian approach to making a prediction about a new data point **x**(n + 1), a data point not in our training data set D. Here x might be the value of the Dow -Jones financial index at the daily closing of the stock-market and n + 1 is one day in the future. Instead of using a point estimate for $\hat{\theta}$ in our model for prediction (as we would in a maximum likelihood or MAP framework), the Bayesian approach is to average over all possible values of θ, weighted by their posterior probability p(θ | D):

$$\begin{aligned} p(\mathbf{x}(n+1) \mid D) &= \int p(\mathbf{x}(n+1), \theta \mid D) d\theta \\ &= \int p(\mathbf{x}(n+1) \mid \theta) p(\theta \mid D) d\theta, \end{aligned}$$

since **x**(n + 1) is conditionally independent of the training data D, given θ, by definition. In fact, we can take this further and also average over different models, using a technique known as Bayesian model averaging. Naturally, all of this averaging can entail considerably more computation than the maximum likelihood approach. This is a primary reason why Bayesian methods have become practical only in recent years (at least for small-scale data sets). For large-scale problems and high-dimensional data, fully Bayesian analysis methods can impose significant computational burdens.

Note that the structure of equations enables the distribution to be updated sequentially. For example, after we build a model with data D_1, we can update it with further data D_2:

$$p(\theta \mid D_1, D_2) \propto p(D_2 \mid \theta) p(D_1 \mid \theta) p(\theta).$$

This sequential updating property is very attractive for large sets of data, since the result is independent of the order of the data (provided, of course, that D_1 and D_2 are conditionally independent given the underlying model p).

NOTES

The denominator p(D) = $\int_{\varphi}$ p(D | φ)p(φ)dφ, is called the predictive distribution of D, and represents our predictions about the value of D. It includes our uncertainty about ө, via the prior p (θ), and our uncertainty about D when θ is known, via p(D | θ). The predictive distribution changes as new data are observed, and can be useful for model checking: if observed data D has only a small probability according to the predictive distribution, that distribution is unlikely to be correct.

The choice of prior distribution can play an important role in Bayesian analysis (more for small samples than for large samples as mentioned earlier). The prior distribution represents our initial belief that the parameter takes different values. The more confident we are that it takes particular values, the more closely the prior will be bunched around those values. The less confident we are, the larger the dispersion of the prior. In the case of a Normal mean, if we had no idea of the true value, we would want to use a prior that gave equal probability to each possible value, i.e., a prior that was perfectly flat or that had infinite variance. This would not correspond to any proper density function (which must have some non-zero values and which must integrate to unity). Still, it is sometimes useful to adopt improper priors that are uniform throughout the space of the parameter. We can think of such priors as being essentially flat in all regions where the parameter might conceivably occur. Even so, there remains the difficulty that priors that are uniform for a particular parameter are not uniform for a nonlinear transformation of that parameter.

Another issue, which might be seen as either a difficulty or strength of Bayesian inference, is that priors show an individual's prior belief in the various possible values of a parameter and individuals differ. It is entirely possible that your prior will differ from mine and therefore we will probably obtain different results from an analysis. In some circumstances this is fine, but in others it is not. One way to overcome this problem is to use a so-called reference prior, a prior that is agreed upon by convention. A common form of reference prior is Jeffrey's prior. To define this, we first need to define the Fisher information:

$$I(\theta \mid \mathbf{x}) = -E\left[\frac{\partial^2 \log L(\theta \mid x)}{\partial \theta^2}\right]$$

for a scalar parameter θ that is, the negative of the expectation of the second derivative of the log-likelihood. Essentially this measures the curvature or flatness of the likelihood function. The flatter a likelihood function is, the less the information it provides about the parameter values. Jeffrey's prior is then defined as

$$p(\theta) \propto \sqrt{I(\theta \mid \mathbf{x})}.$$

This is a convenient reference prior since if $\Phi = \Phi(\theta)$ is some function of θ, this has a prior proportional to $\sqrt{I(\phi \mid \mathbf{x})}$. This means that a consistent prior will result no matter how the parameter is transformed.

The distributions in the examples display began with a Beta or Normal prior and ended with a Beta or Normal posterior. Conjugate families of distributions satisfy this property in general: the prior distribution and posterior distribution belong to the same family. The advantage of using conjugate families is that the complicated updating process can be replaced by a simple updating of the parameters.

4.6 Hypothesis Testing

Although data mining is primarily concerned with looking for unsuspected features in data (as opposed testing specific hypotheses that are formed before we see the data), practice we often do want to test specific hypotheses (for example, if our data mining algorithm generates a potentially interesting hypothesis that we would like to explore further).

In many situations we want to see whether the data support some idea about the value of a parameter. For example, we might want to know if a new treatment has an effect greater than that of the standard treatment, or if two variables are related in a population. Since we are often unable to measure these for an entire population, we must base our conclusions on a sample. Statistical tools for exploring such hypotheses are called hypothesis tests.

4.6.1 Classical Hypothesis Testing

The basic principle of hypothesis tests is as follows. We begin by defining two complementary hypotheses: the null hypothesis and the alternative hypothesis. Often the null hypothesis is some point value (e.g., that the effect in question has value zero that there is no treatment difference or regression slope) and the alternative hypothesis is simply the complement of the null hypothesis. Suppose, for example, that we are trying to draw conclusions about a parameter θ. The null hypothesis, denoted by H_0, might state that $\theta = \theta_0$ and the alternative hypothesis (H_1) might state that θ

NOTES

$\neq \theta_0$. Using the observed data, we calculate a statistic (what form of statistic is best depends on the nature of the hypothesis being tested; examples are given below). The statistic would vary from sample to sample it would be a random variable. If we assume that the null hypothesis is correct, then we can determine the expected distribution for the chosen statistic, and the observed value of the statistic would be one point from that distribution. If the observed value were way out in the tail of the distribution, we would have to conclude either that an unlikely event had occurred or that the null hypothesis was not, in fact, true. The more extreme the observed value, the less confidence we would have in the null hypothesis.

We can put numbers on this procedure. Looking at the top tail of the distribution of the statistic (the distribution based on the assumption that the null hypothesis is true), we can find those potential values that, taken together, have a probability of 0.05 of occurring. These are extreme values of the statistic values that deviate quite substantially from the bulk of the values, assuming the null hypothesis is true. If this extreme observed value did lie in this top region, we could reject the null hypothesis "at the 5% level": only 5% of the time would we expect to see a result in this region as extreme as this if the null hypothesis were correct. For obvious reasons, this region is called the rejection region or critical region. Of course, we might not merely be interested in deviations from the null hypothesis in one direction. That is, we might be interested in the lower tail, as well as the upper tail of the distribution. In this case we might define the rejection region as the union of the test statistic values in the lowest 2.5% of the probability distribution and the test statistic values in the uppermost 2.5% of the probability distribution. This would be a two tailed test, as opposed to the previously described one-tailed test. The size of the rejection region, known as the significance level of the test, can be chosen at will. Common values are 1%, 5%, and 10%.

We can compare different test procedures in terms of their power. The power of a test is the probability that it will correctly reject a false null hypothesis. To evaluate the power of a test, we need a specific alternative hypothesis so we can calculate the probability that the test statistic will fall in the rejection region if the alternative hypothesis is true.

A fundamental question is how to find a good test statistic for a particular problem. One strategy is to use the likelihood ratio. The likelihood ratio statistic used to test the hypothesis H_0: $\theta = \theta_0$ against the alternative H_1: $\theta \neq \theta_0$is defined as

$$\lambda = \frac{L(\theta_0 \mid D)}{\sup_\psi L(\psi \mid D)},$$

where D = {**x**(1), ..., **x**(n)}. That is, the ratio of the likelihood when $\theta = \theta_0$ to the largest value of the likelihood when θ is unconstrained. Clearly, the null hypothesis should be rejected when λ is small. This procedure can easily be generalized to situations in which the null hypothesis is not a point hypothesis but includes a set of possible values for λ.

Certain types of tests are used very frequently. These include tests of differences between means, tests to compare variances, and tests to compare an observed distribution with a hypothesized distribution (so-called goodness-of-fit tests). The common t-test of the difference between the means of two independent groups is described in the display below. Descriptions of other tests can be found in introductory statistics texts.

The hypothesis testing strategy outlined above is based on the assumption that a random sample has been drawn from some distribution, and the aim of the testing is to make a probability statement about a parameter of that distribution. The ultimate objective is to make an inference from the sample to the underlying population of potential values. For obvious reasons, this is sometimes described as the sampling paradigm. An alternative strategy is sometimes appropriate, especially when we are not confident that the sample has been obtained though probability sampling, and therefore inference to the underlying population is not possible. In such cases, we can still sometimes make a probability statement about some effect under a null hypothesis. Consider, for example, a comparison of a treatment and a control group. We might adopt as our null hypothesis that there is no treatment effect, so the distribution of scores of people who received the treatment should be the same as that of those who did not. If we took a sample of people (possibly not randomly drawn) and randomly assign them to the treatment and control groups, we would expect the difference of mean scores between the groups to be small if the null hypothesis was true. Indeed, under fairly general assumptions, it is not difficult to work out the distribution of the difference between the sample mea ns of the two groups we would expect if there were no treatment effect, and if such difference were just a consequence of an imbalance in the random allocation. We can then explore how unlikely it is that a difference as large or larger than that actually obtained would be seen. Tests based on this principle are termed randomization tests or permutation tests. Note that they make no statistical inference from

NOTES

the sample to the overall population, but they do enable us to make conditional probability statements about the treatment effects, conditional on the observed values.

Many statistical tests make assumptions about the forms of the population distributions from which the samples are drawn. For example, in the two-sample t-test, illustrated above, an assumption of Normality was made. Often, however, it is inconvenient to make such assumptions. Perhaps we have little justification for the assumption, or perhaps we know that the data do not to follow the form required by a standard test. In such circumstances we can adopt distribution-free tests. Tests based on ranks fall into this class. Here the basic data are replaced by the numerical labels of the positions in which they occur. For example, to explore whether two samples arose from the same distribution, we could replace the actual numerical values by their ranks. If they did arise from the same distribution, we would expect the ranks of the members of the two samples to be well mixed. If, however, one distribution had a larger mean than the other, we would expect one sample to tend to have large ranks and the other to have small ranks. If the distributions had the same means but one sample had a larger variance than the other, we would expect one sample to show a surfeit of large and small ranks and the other to dominate the intermediate ranks. Test statistics can be constructed based on the average values or some other measurements of the ranks, and their significance levels can be evaluated using randomization arguments. Such test statistics include the sign test statistic, the rank sum test statistic, the Kolmogorov-Smirnov test statistic, and the Wilcoxon test statistic. Sometimes the term nonparametric test is used to describe such tests the rationale being that these tests are not testing the value of a parameter of any assumed distribution.

Comparison of hypotheses H_0 and H_1 from a Bayesian perspective is achieved by comparing their posterior probabilities:

$$p(H_i|x) \propto p(x|H_i)\,p(H_i)$$

Taking the ratio of these leads to a factorization in terms of the prior odds and the likelihood ratio, or Bayes factor:

$$\frac{p(H_0|x)}{p(H_1|x)} \propto \frac{p(H_0)}{p(H_1)} \cdot \frac{p(x|H_0)}{p(x|H_1)}.$$

There are some complications here, however. The likelihoods are marginal likelihoods obtained by integrating over parameters not

NOTES

specified in the hypotheses, and the prior probabilities will be zero if the H_i refer to particular values from a continuum of possible values (e.g., if they refer to values of a parameter θ, where θ can take any value between 0 and 1). One strategy for dealing with this problem is to assign a discrete non- zero prior probability to the given values of θ.

4.6.2 Hypothesis Testing in Context

This section has so far described the classical (frequentist) approach to statistical hypothesis testing. In data mining, however, analyses can become more complicated.

Firstly, because data mining involves large data sets, we should expect to obtain statistical significance: even slight departures from the hypothesized model form will be identified as significant, even though they may be of no practical importance. (If they are of practical importance, of course, then well and good.) Worse, slight departures from the model arising from contamination or data distortion will show up as significant. We have already remarked on the inevitability of this problem.

Secondly, sequential model fitting processes are common. For various stepwise model fitting procedures, which gradually refine a model by adding or deleting terms. Running separate tests on each model, as if it were de novo, leads to incorrect probabilities. Formal sequential testing procedures have been developed, but they can be quite complex. Moreover, they may be weak because of the multiple testing going on.

Thirdly, the fact that data mining is essentially an exploratory process has various implications. One is that many models will be examined. Suppose we test m true (though we will not know this) null hypotheses at the 5% level, each based on its own subset of the data, independent of the other tests. For each hypothesis separately, there is a probability of 0.05 of incorrectly rejecting the hypothesis. Since the tests are independent, the probability of incorrectly rejecting at least one is $p = 1 - (1 - 0.05)^m$. When $m = 1$ we have $p = 0.05$, which is fine. But when $m = 10$ we obtain $p = 0.4013$, and when $m = 100$ we obtain $p = 0.9941$. Thus, if we test as few as even 100 true null hypotheses, we are almost certain to incorrectly reject at least one. Alternatively, we could control the overall family error rate, setting the probability of incorrectly rejecting one of more of the m true null hypotheses to 0.05. In this case we use $0.05 = 1 - (1 - a)^m$ for each given m to obtain the level a at which each of the separate null hypotheses is tested. With m

= 10 we obtain a = 0.0051, and with m = 100 we obtain a = 0.0005. This means that we have a very small probability of incorrectly rejecting any of the separate component hypotheses.

Of course, in practice things are much more complicated: the hypotheses are unlikely to be completely independent (at the other extreme, if they are completely dependent, accepting or rejecting one implies the acceptance or rejection of all), with an essentially unknowable dependence structure, and there will typically be a mixture of true (or approximately true) and false null hypotheses.

Various simultaneous test procedures have been developed to ease these difficulties (even though the problem is not really one of inadequate methods, but is really more fundamental). A basic approach is based on the Bonferroni inequality. We can expand the probability $(1 - \alpha)^m$ that none of the true null hypotheses are rejected to yield $(1 - \alpha)^m = 1 - m\alpha$. It follows that $1 - (1 - \alpha)^m = m\alpha$ that is, the probability that one or more true null hypotheses is incorrectly rejected is less than or equal to ma. In general, the probability of incorrectly rejecting one or more of the true null hypotheses is smaller than the sum of probabilities of incorrectly rejecting each of them. This is a first-order Bonferroni inequality. By including other terms in the expansion, we can develop more accurate bounds though they require knowledge of the dependence relationships between the hypotheses.

With some test procedures difficulties can arise in which a global test of a family of hypotheses rejects the null hypothesis (so we believe at least one to be false), but no single component is rejected. Once again strategies have been developed for overcoming this in particular applications. For example, in multivariate analysis of variance, which compares several groups of objects that have been measured on multiple variables, test procedures have been developed that overcome these problems by comparing each test statistic with a single threshold value.

It is obvious from the above discussion that while attempts to put probabilities on statements of various kinds, via hypothesis tests, do have a place in data mining, and they are not a universal solution. However, they can be regarded as a particular type of a more general procedure that maps the data and statement to a numerical value or score. Higher scores (or lower scores, depending upon the procedure) indicate that one statement or model is to be preferred to another, without attempting any absolute probabilistic interpretation.

NOTES

4.7 Sampling Methods

As mentioned earlier, data mining can be characterized as secondary analysis, and data miners are not typically involved directly with the data collection process. Still, if we have information about that process that might be useful for our analysis, we should take advantage of it. Traditional statistical data collection is usually carried out with a view to answering some particular question or questions in an efficient and effective manner. However, since data mining is a process seeking the unexpected or the unforeseen, it does not try to answer questions that were specified before the data were collected. For this reason we will not be discussing the sub-discipline of statistics known as experimental design, which is concerned with optimal ways to collect data. The fact that data miners typically have no control over the data collection process may sometimes explain poor data quality: the data may be ideally suited to the purposes for which it was collected, but not adequate for its data mining uses.

We have already noted that when the database comprises the entire population, notions of statistical inference are irrelevant: if we want to know the value of some population parameter (the mean transaction value, say, or the largest transaction value), we can simply calculate it. Of course, this assumes that the data describe the population perfectly, with no measurement error, missing data, data corruption, and so on. Since, as we have seen, this is an unlikely situation, we may still be interested in making an inference from the data as recorded to the "true" underlying population values.

Furthermore, the notions of populations and samples can be deceptive. For example, even when values for the entire population have been captured in the database, often the aim is not to describe that population, but rather to make some statement about likely future values. For example, we may have available the entire population of transactions made in a chain of supermarkets on a given day. We may well wish to make some kind of inferential statement about the mean transaction value for the next day or some other future day. This also involves uncertainty, but it is of a different kind from that discussed above. Essentially, here, we are concerned with forecasting. In market basket analysis we do not really wish to describe the purchasing patterns of last month's shoppers, but rather to forecast how next month's shoppers are likely to behave.

NOTES

We have distinguished two ways in which samples arise in data mining. First, sometimes the database itself is merely a sample from some larger population. The implications of this situation and the dangers associated with it. Second the database contains records for every object in the population, but the analysis of the data is based on only a sample from it. This second technique is appropriate only in modeling situations and certain pattern detection situations. It is not appropriate when we are seeking individual unusual records.

Our aim is to draw a sample from the database that allows us to construct a model that reflects the structure of the data in the database. The reason for using just a sample, rather than the entire data set, is one of efficiency. At an extreme, it may be infeasible, in terms of time or computational requirements, to use the entirety of a large database. By basing our computations solely on a sample, we make the computations quicker and easier. It is important, however, that the sample be drawn in such a way that it reflects the structure of the complete set i.e., that it is representative of the entire database.

There are various strategies for drawing samples to try to ensure representativeness. If we wanted to take just 1 in 2 of the records (a sampling fraction of 0.5), we could simply take every other record. Such a direct app roach is termed systematic sampling. Often it is perfectly adequate. However, it can also lead to unsuspected problems. For instance, if the database contained records of married couples, with husbands and wives alternating, systematic sampling could be disastrous the conclusions drawn would probably be entirely mistaken. In general, in any sampling scheme in which cases are selected following some regular pattern there is a risk of interaction with an unsuspected regularity in the database. Clearly what we need is a selection pattern that avoids regularities a random selection pattern.

The word random is used here in the sense of avoiding regularities. This is slightly different from the usage employed previously in this chapter, where the term referred to the mechanism by which the sample was chosen. There it described the probability that a record would be chosen for the sample. As we have seen, samples that are random in this second sense can be used as the basis for statistical inference: we can, for example, make a statement about how likely it is that the sample mean will differ substantially from the population mean.

NOTES

If we draw a sample using a random process, the sample will satisfy the second meaning and is likely to satisfy the first as well. (Indeed, if we specify clearly what we mean by "regularities" we can give a precise probability that a randomly selected sample will not match such regularities.) To avoid biasing our conclusions, we should design our sample selection mechanism in such a way that that each record in the database has an equal chance of being chosen. A sample with equal probability of selecting each member of the population is known as an epsem sample. The most basic form of epsem sampling is simple random sampling, in which the n records comprising the sample are selected from the N records in the database in such a way that each set of n records has an equal probability of being chosen. The estimate of the population mean from a simple random sample is just the sample mean.

At this point we should note the distinction between sampling with replacement and sampling without replacement. In the former, a record selected for inclusion in the sample has a chance of being drawn again, but in the latter, once a record is drawn it cannot be drawn a second time. In data mining since the sample size is often small relative to the population size, the differences between the results of these two procedures are usually negligible.

Figure 4.2 illustrates the results of a simple random sampling process used in calculating the mean value of a variable for some population. It is based on drawing samples from a population with a true mean of 0.5. A sample of a specified size is randomly drawn and its mean value is calculated; we have repeated this procedure 200 times and plotted histograms of the results. Figure 4.2 shows the distribution of sample mean values (a) for samples of size 10, (b) size 100, and (c) size 1000. It is apparent from this figure that the larger the sample, the more closely the values of the sample mean is distributed around about the true mean. In general, if the variance of a population of size N is s^2, the variance of the mean of a simple random sample of size n from that population, drawn without replacement, is

$$\frac{\sigma^2}{n}\left(1-\frac{n}{N}\right).$$

Since we normally deal with situations in which N is large relative to n (i.e., situations that involve a small sampling fraction), we can usually ignore the second factor, so that, a good approximation of the variance is σ^2/n. From this it follows that the larger the sample is the less likely it is that the sample means will deviate significantly from the population mean which explains why the dispersion of the histograms decreases with increasing sample

NOTES

size. Note also that this result is independent of the population size. What matters here is the size of the sample, not the size of the sampling fraction, and not the proportion of the population that is included in the sample. We can also see that, when the sample size is doubled, the standard deviation is reduced not by a factor of 2, but only by a factor of $\sqrt{2}$ there are diminishing returns to increasing the sample size. We can estimate s^2 from the sample using the standard estimator

$$\sum (x(i) - \bar{x})/(n-1),$$

where x(i) is the value of the ith sample unit and ? is the mean of the n values in the sample.

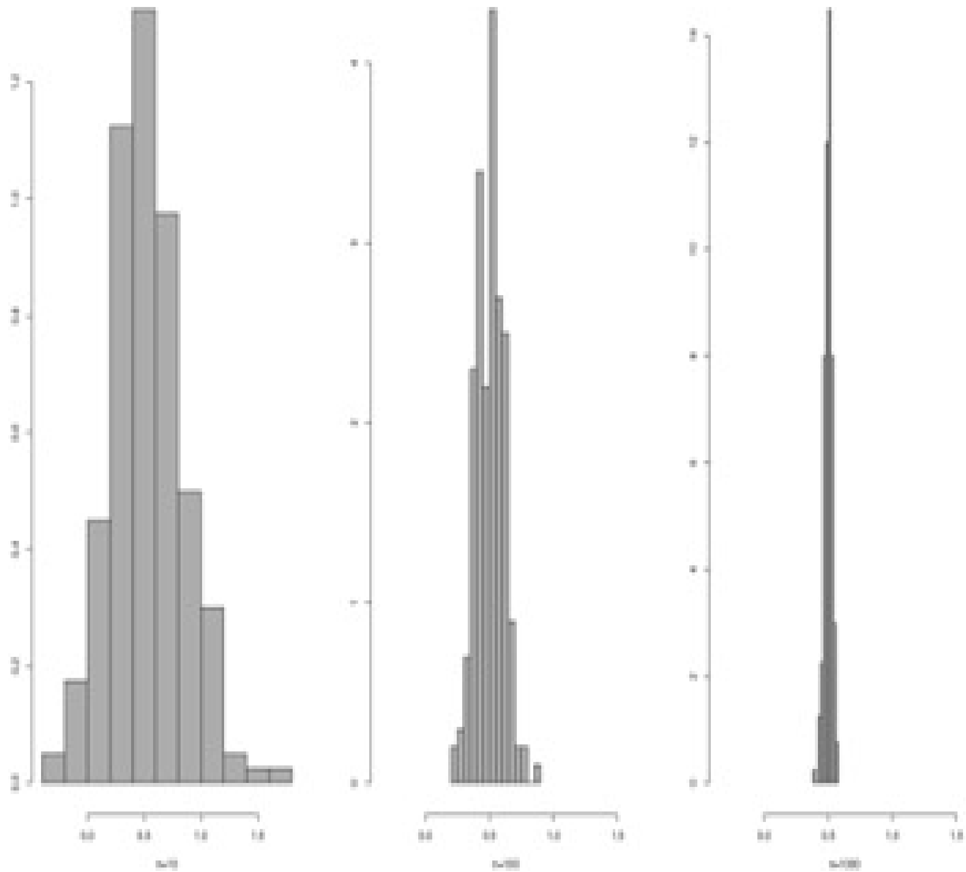


Figure 4.2: Means of Samples of Size 10(a), 100(b), and 1000(c) Drawn From a Population with a Mean of 0.5.

The simple random sample is the most basic form of sample design, but others have been developed that have desirable properties under different circumstances. Details can be found in books on survey sampling, such as those cited at the end of this chapter. Here we will briefly describe two important schemes.

In stratified random sampling, the entire population is split into non over -lapping subpopulations or strata, and a sample (often, but not necessarily, a simple random sample) is drawn separately from within each stratum. There are several potential advantages to using such a procedure. An obvious one is that it enables us to make statements about each of the subpopulations separately, without relying on chance to ensure that a reasonable number of observations come from each subpopulation. A more subtle, but often more important, advantage is that if the strata are relatively homogeneous in terms of the variable of interest (so that much of the variability between values of the variable is accounted for by differences between strata), the variance of the overall estimate may be smaller than that arising from a simple random sample. To illustrate, one of the credit card companies we work with categorizes transactions into 26 categories: supermarket, travel agent, gas station, and so on. Suppose we wanted to estimate the average value of a transaction. We could take a simple random sample of transaction values from the database of records, and compute its mean, using this as our estimate. However, with such a procedure some of the transaction types might end up being underrepresented in our sample, and some might be overrepresented. We could control for this by forcing our sample to include a certain number of each transaction type. This would be a stratified sample, in which the transaction types were the strata. This example illustrates why the strata must be relatively homogeneous internally, with the heterogeneity occurring between strata. If all the strata had the same dispersion as the overall population, no advantage would be gained by stratification.

In general, suppose that we want to estimate the population means for some variable, and that we are using a stratified sample, with simple random sampling within each stratum. Suppose that the kth stratum has N_k elements in it, and that n_k of these are chosen for the sample from this stratum. Denoting the sample mean within the k th stratum by $\bar{x}_k$, the estimate of the overall population mean is given by

$$\sum \frac{N_k \bar{x}_k}{N},$$

where N is the total size of the population. The variance of this estimator is

$$\frac{1}{N^2} \sum N_k^2 \, var(\overline{x_k})$$

where $var(\overline{x_k})$ is the variance of the simple random sample of size n_k for the kth stratum, computed as above.

Data often have a hierarchical structure. For example, letters occur in words, which lie in sentences, which are grouped into paragraphs, which occur in chapters, which form books, which sit in libraries. Producing a complete sampling frame and drawing a simple random sample may be difficult. Files will reside on different computers at a site within an organization, and the organization may have many sites; if we are studying the properties of those files, we may find it impossible to produce a complete list from which we can draw a simple random sample. In cluster sampling, rather than drawing a sample of the individual elements that are of interest, we draw a sample of units that contain several elements. In the computer file example, we might draw a sample of computers. We can the examine all of the files on each of the chosen computers, or move on to a further stage of sampling.

Clusters are often of unequal sizes. In the above example we can view a computer as providing a cluster of files, and it is very unlikely that all computers in an organization would have the same number of files. But situations with equal-sized clusters do arise. Manufacturing industries provide many examples: six-packs of beer or packets of condoms, for instance. If all of the units in each selected cluster are chosen (if the subsampling fraction is 1) each unit has the probability α/K of being selected, where α is the number of clusters chosen from the entire set of K clusters. If not all the units are chosen, but the sampling fraction in each cluster is the same, each unit will have the same probability of being included in the sample (it will be an epsem sample). This is a common design. Estimating the variance of a statistic based on such a design is less straightforward than the cases described above since the sample size is also a random variable (it is dependent upon which clusters happen to be included in the sample). The estimate of the mean of a variable is a ratio of two random variables: the total sum for the units included in the sample and the total number of units included in the sample. Denoting the size of the simple random sample chosen from the kth cluster by n_k, and the total sum for the units chosen from the kth cluster by s_k, the sample mean r is

$$\sum x_k / \sum n_k .$$

NOTES

If we denote the overall sampling fraction by f (often this is small and can be ignored) the variance of r is

$$\frac{1-f}{(\sum n_k)^2}\frac{a}{1-a}\left(\sum s_k^2 + r^2\sum n_k^2 - 2r\sum s_k n_k\right).$$

Summary

Data mining should not be seen as a simple one-time exercise. Huge data collections may be analyzed and examined in an unlimited number of ways. As time progresses, so new kinds of structures and patterns may attract interest, and may be worth seeking in the data.

Data mining has, for good reason, recently attracted a lot of attention: it is a new technology, tackling new problems, with great potential for valuable commercial and scientific discoveries. However, we should not expect it to provide answers to all questions. Like all discovery processes, successful data mining has an element of serendipity. While data mining provides useful tools that does not mean that it will inevitably lead to important, interesting, or valuable results. We must beware of over exaggerating the likely outcomes. But the potential is there.

We have restricted our discussion to numeric data. However, other kinds of data also arise. For example, text data is an important class of non-numeric data. Sometimes the definition of an individual data item (and hence whether it is numeric or non-numeric) depends on the objectives of our analysis: in economic contexts, in which hundreds of thousands of time series are stored in databases, the data items might be entire time series, rather than the individual numbers within those series. Even with non-numeric data, numeric data analysis plays a fundamental role. Often non-numeric data items, or the relationships between them, are reduced to numeric descriptions, which are subject to standard methods of analysis. For example, in text processing us might measure the number of times a particular word occurs in each document, or the probability that certain pairs of words appear in documents.

Nothing is certain. In the data mining context, our objective is to make discoveries from data. We want to be as confident as we can that our conclusions are correct, but we often must be satisfied with a conclusion that could be wrong though it will be better if we can also state our level of confidence in our conclusions. When we are analyzing entire populations, the uncertainty will creep in via less than perfect data quality: some values may be incorrectly recorded, some values may be missing, and some members of the population are omitted from the database entirely, and so on. When we are working with samples, our aim is often to draw a conclusion that applies to the broader population from which the sample was drawn. The fundamental tool in tackling all of these issues is probability. This is a universal language for handling uncertainty, a language that has been refined throughout this century and has been applied across a vast array of situations. Application of the ideas of probability enables us to obtain "best" estimates of values, even in the face of data inadequacies, and even when only a sample has been measured. Moreover, application of these ideas also allows us to quantify our confidence in the results.

Reference

1. Hand D, Mannila H. Smith P: Principles of Data Mining (PHI)

2. Pujari A: Data Mining Techniques, University Press(orient Longman

UNIT - II

5. A Systematic Overview of Data Mining Algorithms

Structure

Objective

After going through this lesson, you should be able to:

- Understand the nature of data sets;
- Discuss models and patterns;
- Discuss components of data mining algorithms;

5.1 Introduction

This lesson will examine what we mean in a general sense by a data mining algorithm as well as what components make up such algorithms. A working definition is as follows:

A data mining algorithm is a well-defined procedure that takes data as input and produces output in the form of models or patterns.

We use the term well-defined indicate that the procedure can be precisely encoded as a finite set of rules. To be considered an algorithm, the procedure must always terminate after some finite number of steps and produce an output. In contrast, a computational method has all the properties of an algorithm except a method for guaranteeing that the procedure will terminate in a finite number of steps. While specification of an algorithm typically involves defining many practical implementation details, a computational method is usually described more abstractly. For example, the search technique steepest descent is a computational method but is not in itself an algorithm (this search method repeatedly moves in parameter space in the direction that has the steepest decrease in the score function relative to the current parameter values). To specify an algorithm using the steepest descent method, we would have to give precise methods for determining where to begin descending, how to identify the direction of steepest descent (calculated exactly or approximated?), how far to move in the chosen direction, and when to terminate the search (e.g., detection of convergence to a local minimum).

As discussed briefly in lesson-1, the specification of a data mining algorithm to solve a particular task involves defining specific algorithm components:

1. the data mining task the algorithm is used to address (e.g., visualization, classification, clustering, regression, and so forth). Naturally, different types of algorithms are required for different tasks.

2. the structure (functional form) of the model or pattern we are fitting to the data (e.g., a linear regression model, a hierarchical clustering model, and so forth). The structure defines the boundaries of what we can approximate or learn. Within these boundaries, the data guide us to a particular model or pattern.

3. the score function we are using to judge the quality of our fitted models or patterns based on observed data (e.g., misclassification error or squared error). Therefore, it is important that the score function reflects the relative practical utility of different parameterizations of our model or pattern structures. Furthermore, the score function is critical for learning and generalization. It can be based on goodness-of-fit alone (i.e., how well the model can describe the observed data) or can try to capture generalization performance (i.e., how well will the model describe data we have not yet seen). As we will see in later chapters, this is a subtle issue.

4. the search or optimization method we use to search over parameters and structures, i.e., computational procedures and algorithms used to find the maximum (or minimum) of the score function for particular models or patterns. Issues here include

NOTES

computational methods used to optimize the score function (e.g., steepest descent) and search-related parameters (e.g., the maximum number of iterations or convergence specification for an iterative algorithm). If the model (or pattern) structure is a single fixed structure (such as a k th-order polynomial function of the inputs), the search is conducted in parameter space to optimize the score function relative to this fixed structural form. If the model (or pattern) structure consists of a set (or family) of different structures, there is a search over both structures and their associated parameter spaces. Optimization and search are traditionally at the heart of any data mining algorithm.

5. The data management technique to be used for storing, indexing, and retrieving data. Many statistical and machine learning algorithms do not specify any data management technique, essentially assuming that the data set is small enough to reside in main memory so that random access of any data point is free (in terms of time) relative to actual computational costs. However, massive data sets may exceed the

capacity of available main memory and reside in secondary (e.g., disk) or tertiary (e.g., tape) memory. Accessing such data is typically orders of magnitude slower than accessing main memory, and thus, for massive data sets, the physical location of the data and the manner in which it is accessed can be critically important in terms of algorithm efficiency.

Table 5.1 illustrates how three well-known data mining algorithms (CART, back propagation, and the A Priori algorithm) can be described in terms of these basic components. Each of these algorithms will be discussed in detail later in this chapter. (One of the differences between statistical and data mining perspectives is evident from this table. Statisticians would regard CART as a model, and back propagation as a parameter estimation algorithm. Data miners tend to see things more in terms of algorithms: processing the data using the algorithm to yield a result. The difference is really more one of perspective than substance.)

Table 5.1: Three Well-Known Data Mining Algorithms Broken Down in Terms of their algorithm Components.

	CART	Backpropagation	A Priori
Task	Classification and Regression	Regression	Rule Pattern Discovery
Structure	Decision Tree	Neural Network (Nonlinear functions)	Association Rules
Score Function	Cross- validated Loss Function	Squared Error	Support/Accuracy
Search Method	Greedy Search over Structures	Gradient Descent on Parameters	Breath-First with Pruning
Data Management Technique	Unspecified	Unspecified	Linear Scans

5.2 An Example: The CART Algorithm for Building Tree Classifiers

To clarify the general idea of viewing algorithms in terms of their components, we will begin by looking at one well-known algorithm for classification problems.

The CART (Classification And Regression Trees) algorithm is a widely used statistical procedure for producing classification and regression models with a tree-based structure. For the sake of simplicity we will consider only the classification aspect of CART, that is, mapping an input vector **x** to a categorical (class) output label y (see figure 5.1). In the context of the components discussed above, CART can be viewed as the "algorithm tuple" consisting of the following:

1. task = prediction (classification)
2. model structure = tree
3. score function = cross-validated loss function
4. search method = greedy local search
5. data management method = unspecified

The fundamental distinguishing aspect of the CART algorithm is the model structure being used; the classification tree. The CART tree model consists of a hierarchy of univariate binary decisions.

NOTES

Figure 5.2 shows a simple example of such a classification tree for the data in figure 5.1. Each internal node in the tree specifies a binary test on a single variable, using thresholds on real and integer-valued variables and subset membership for categorical variables. (In general we use b branches at each node, b ≥ 2.) A data vector **x** descends a unique path from the root node to a leaf node depending on how the values of individual components of **x** match the binary tests of the internal nodes. Each leaf node specifies the class label of the most likely class at that leaf or, more generally, a probability distribution on class values conditioned on the branch leading to that leaf.

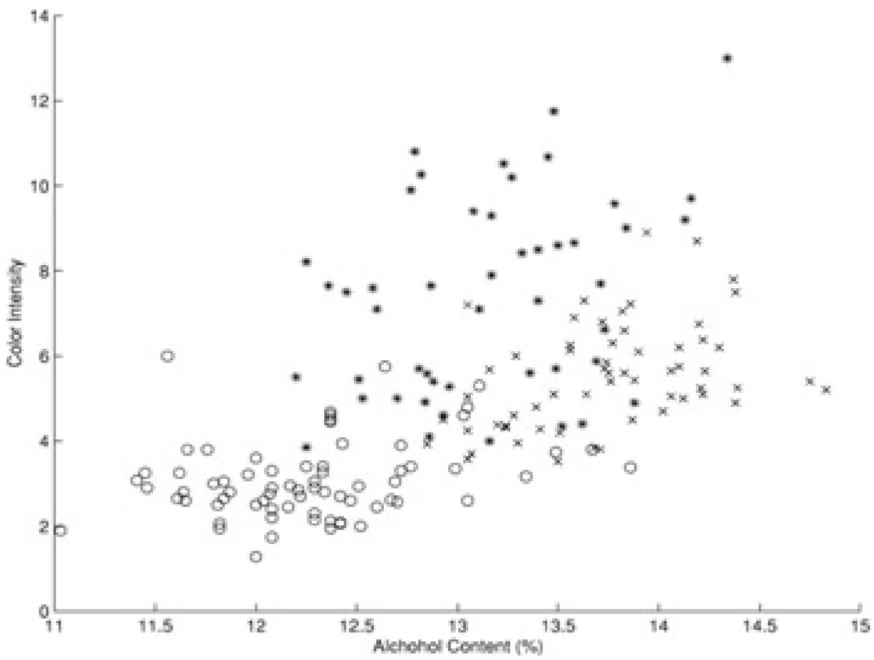


Figure 5.1: A scatterplot of data showing color intensity versus alcohol content for a set of wines. the data mining task is to classify the wines into one of three classes (three different cultivars), each shown with a different symbol in the plot. the data originate from a 13-dimensional data set in which each variable measures of a particular characteristic of a specific wine.

The structure of the tree is derived from the data, rather than being specified a priori (this is where data mining comes in). CART operates by choosing the best variable for splitting the data into two groups at the root node. It can use any of several different splitting criteria; all produce the effect of partitioning the data at an internal node into two disjoint subsets (branches) in such a way that the class labels in each subset are as homogeneous as

NOTES

possible. This splitting procedure is then recursively applied to the data in each of the child nodes, and so forth. The size of the final tree is a result of a relatively complicated "pruning" process, outlined below. Too large a tree may result in overfitting, and too small a tree may have insufficient predictive power for accurate classification.

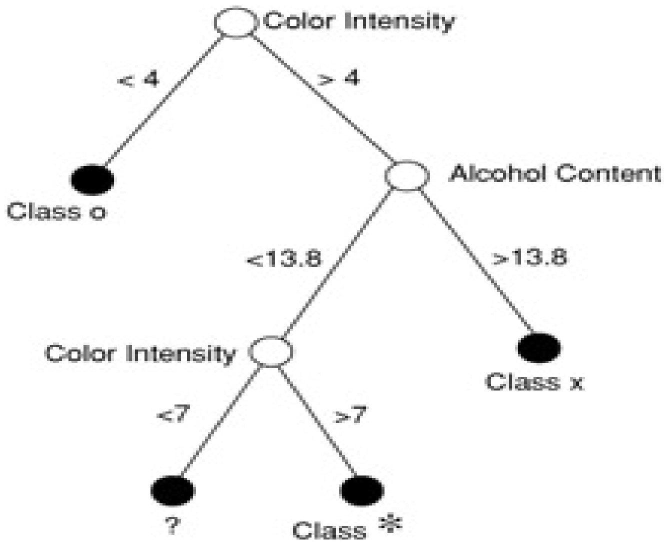


Figure 5.2: A classification tree for the data in figure 5.1 in which the tests consist of thresholds (shown beside the branches) on variables at each internal node and leaves contain class decisions. note that one leaf is denoted? to illustrate that there is considerable uncertainty about the class labels of data points in this region of the space.

The hierarchical form of the tree structure clearly separates algorithms like CART from classification algorithms based on non-tree structures (e.g., a model that uses a linear combination of all variables to define a decision boundary in the input space). A tree structure used for classification can readily deal with input data that contain mixed data types (i.e., combinations of categorical and real-valued data), since each internal node depends on only a simple binary test. In addition, since CART builds the tree using a single variable at a time, it can readily deal with large numbers of variables. On the other hand, the representational power of the tree structure is rather coarse: the decision regions for classifications are constrained to be hyper -rectangles, with boundaries constrained to be parallel to the input variable axes (as an example, see figure 5.3).

NOTES

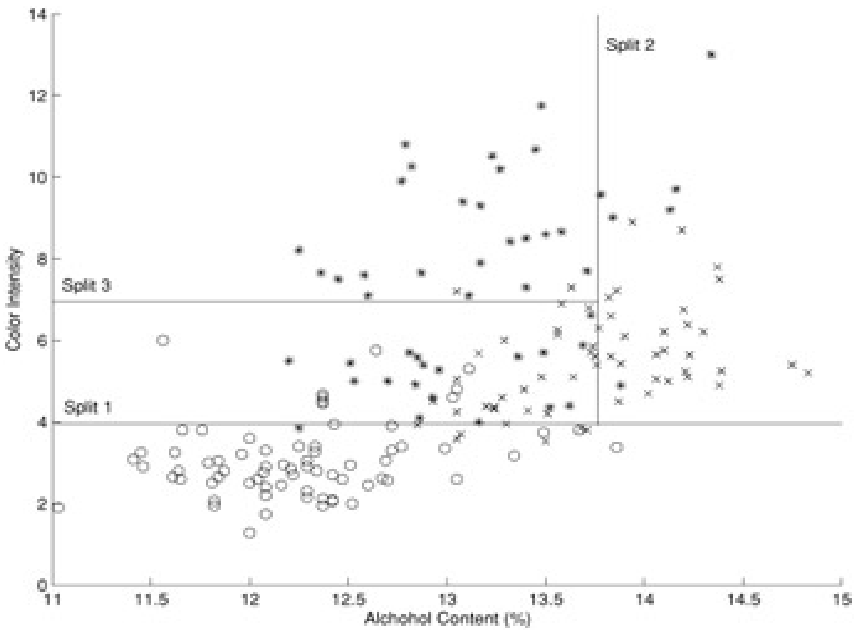


Figure 5.3: The decision boundaries from the classification tree in figure 5.2 are superposed on the original data. note the axis-parallel nature of the boundaries.

The score function used to measure the quality of different tree structures is a general misclassification loss function, defined as

$$\sum_{i=1}^{n} C\left(y(i), \hat{y}(i)\right),$$

where C (y(i), $\hat{y}$(i)) is the loss incurred (positive) when the class label for the ith data vector, y(i), is predicted by the tree to be $\hat{y}$(i). In general, C is specified by an m × m matrix, where m is the number of classes. For the sake of simplicity we will assume here a loss of 1 is incurred whenever $\hat{y}$(i) ≠ y(i), and the loss is 0 otherwise. (This is known as the "0–1" loss function or the misclassification rate if we normalize the sum above by dividing by n.)

CART uses a technique known as cross-validation to estimate this misclassification loss function. Basically, this method partitions the training data into a subset for building the tree and then estimates the misclassification rate on the remaining validation subset. This partitioning is repeated multiple times on different subsets, and the misclassification rates are then averaged to yield a cross-validation estimate of how well a tree of a particular size will perform on new, unseen data. The size of tree that produces the smallest cross-validated misclassification estimate is selected as the appropriate

NOTES

size for the final tree model. (This description captures the essence of tree selection via cross-validation, but in practice the process is a little more complex.)

Cross-validation allows CART to estimate the performance of any tree model on data not used in the construction of the tree i.e., it provides an estimate of generalization performance. This is critical in the tree-growing procedure, since the misclassification rate on the training data (the data used to construct the tree) can often be reduced by simply making the tree more complex; thus, the training data error is not necessarily indicative of how the tree will perform on new data.

Figure 5.4 illustrates this point with a hypothetical plot of typical error rates as a function the size of the tree. The error rate on the training data decreases monotonically (to an error rate of zero if the variables can produce leaves that each contains data from a only single class). The test error rate on new data (which is what we are typically interested in for prediction) also decreases at first. Very small trees (to the left) do not have sufficient predictive power to make accurate predictions. However, unlike the training error, the test error "bottoms out" and begins to increase again as the algorithm overfits the data and adds nodes that are merely predicting noise or random variation in the training data, and which is irrelevant to the predictive task. The goal of an algorithm like CART is to find a tree close to the optimal tree size (which is of course unknown ahead of time); it tries to find a model that is complex enough to capture any structure that exists, but not so complex that it overfits. For small to medium amounts of data it is preferable to do this without having to reserve some of our data to estimate this out-of-sample error. For very large data sets we can sometimes afford to simply partition the data into training and validation data sets and to monitor performance on the validation data.

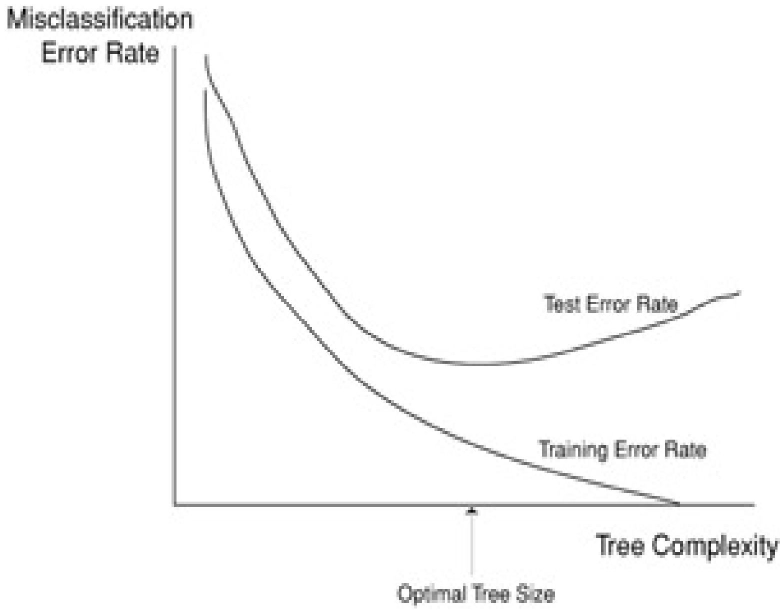


Figure 5.4: A Hypothetical Plot of Misclassification Error Rates for Both Training and Test data as a Function of Tree Complexity (e.g., Number of Leaves in the Tree).

5.3 The Reductionist Viewpoint on Data Mining Algorithms

The reductionist (i.e., a component -based) view for data mining algorithms is quite useful in practice. It clarifies the underlying operation of a particular data mining algorithm by reducing it to its essential components. In turn, this makes it easier to compare different algorithms, since we can clearly see similarities and differences at the component level (e.g., we were able to distinguish between CART and C4.5 primarily in terms of what score functions they use).

Even more important, this view places an emphasis on the fundamental properties of an algorithm avoiding the tendency to think of lists of algorithms. When faced with a data mining application, a data miner should think about which components fit the specifics of his or her problem, rather than which specific "off-the-shelf" algorithm to choose. In an ideal world, the data miners would have available a software environment within which they could compose components (from a library of model structures, score functions, search methods, etc.) to synthesize an algorithm customized for their specific applications. Unfortunately this

NOTES

remains a ideal state of affairs rather than the practical norm; current data analysis software packages often provide only a list of algorithms, rather than a component -based toolbox for algorithm synthesis. This is understandable given the aim of providing usable tools for data miners who do not have the background or the time to understand the underlying details at a component level. However these software tools may not be ideal for more skilled practitioners who wish to customize and synthesize problem-specific algorithms. The "cookbook" approach is also somewhat dangerous, since naive users of data mining tools may not fully understand the limitations (and underlying assumptions) of the particular black-box algorithms they are using. In contrast, a description based on components makes it relatively clear what is inside the black box.

5.3.1 Multilayer Perceptions for Regression and Classification

Feedforward multilayer perceptrons (MLPs) are the most widely used models in the general class of artificial neural network models. The MLP structure provides a nonlinear mapping from a real-valued input vector **x** to a real -valued output vector **y**. As a result, an MLP can be used as a nonlinear model for regression problems, as well as for classification, through appropriate interpretation of the outputs. The basic idea is that a vector of p input values is multiplied by a $p \times d_1$ weight matrix, and the resulting d_1 values are each individually transformed by a nonlinear function to produce d_1 "hidden node" outputs. The resulting d_1 values are then multiplied by a $d_1 \times d_2$ weight matrix (another "layer" of weights), and the d_2 values are each put through a non-linear function. The resulting d_2 values can either be used as the outputs of the model or be put through another layer of weight multiplications and non -linear transformations, and so on (hence, the "multilayer" nature of the model; the term perceptron refers to the original model of this form proposed in the 1960s, consisting of a single layer of weights followed by a threshold nonlinearity).

As an example, consider the simple network model in figure 5.5 with a single "hidden" layer. Two inner products, $s_1 = \sum_{i=1}^{4} \alpha_i x_1$ and $s_2 = \sum_{i=1}^{4} \beta_i x_1$, are calculated via the first layer of weights (the αs and the βs), and each in turn transformed by a nonlinear function at the hidden nodes to produce two scalar values: h_1 and h_2. The nonlinear logistic function, i.e., $h_1=h(s_1)=1/(1+e^{-s1})$, is widely used. Next h1 and h2 are weighted and combined to produce the output value $y = \sum_{i=1}^{42} w_i h_1$ (we could in principle perform a nonlinear transformation on y also). Thus, y is a nonlinear function of the input vector **x**. The hs can be viewed as nonlinear transformations of the four-dimensional input, a new set of two "basis functions," h_1 and h_2. The parameters of this model to be estimated from the data are the eight weights on the input layer (α_1, ..., α_4, β_1, ..., β_4) and the two weights on the output layer (w_1 and w_2). In general, with p inputs, a single hidden layer with h hidden nodes, and a single output, there are (p + 1)h parameters (weights) in all to be estimated from the data. In general we can have multiple layers of such weight multiplications and nonlinear transformations, but a single hidden layer is used most often since multiple hidden layer networks can be slow to train. The weights of the MLP are the parameters of the model and must be determined from the data.

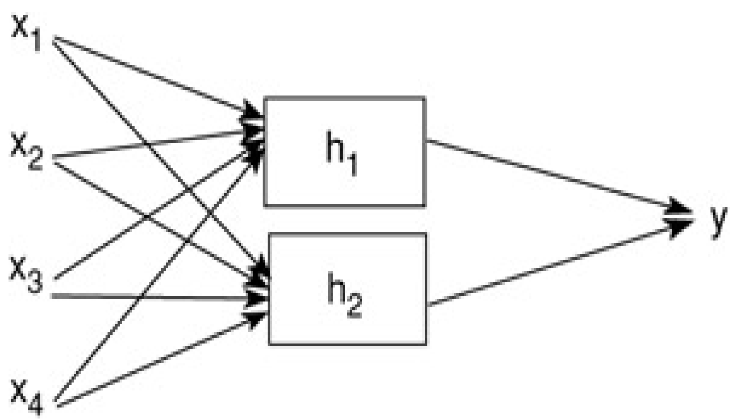


Figure 5.5: A diagram of a simple multilayer perceptron (or neural network) model with two hidden nodes (d_1 = 2) and a single output node (d_2 = 1).

Note that if the output **y** is a scalar y (i.e., d_2 = 1) and is bounded between 0 and 1 (we can just choose a nonlinear transformation of the weighted values coming from the previous layer to ensure this condition), we can use y as an indicator of class membership for two class problems and (for example) threshold at 0.5 to decide between class 1 and class 2. Thus, MLPs can easily be used for classification as well as for regression. Because of the nonlinear nature of the model, the decision boundaries between different classes produced by a network model can also be quite non-linear. Figure 5.6 provides an example of such decision boundaries. Note

NOTES

that they are highly nonlinear, in contrast to those produced by the classification tree in figure 5.3. Unlike the classification tree in figure 5.2, however, there is no simple summary form we can use to describe the workings of the neural network model.

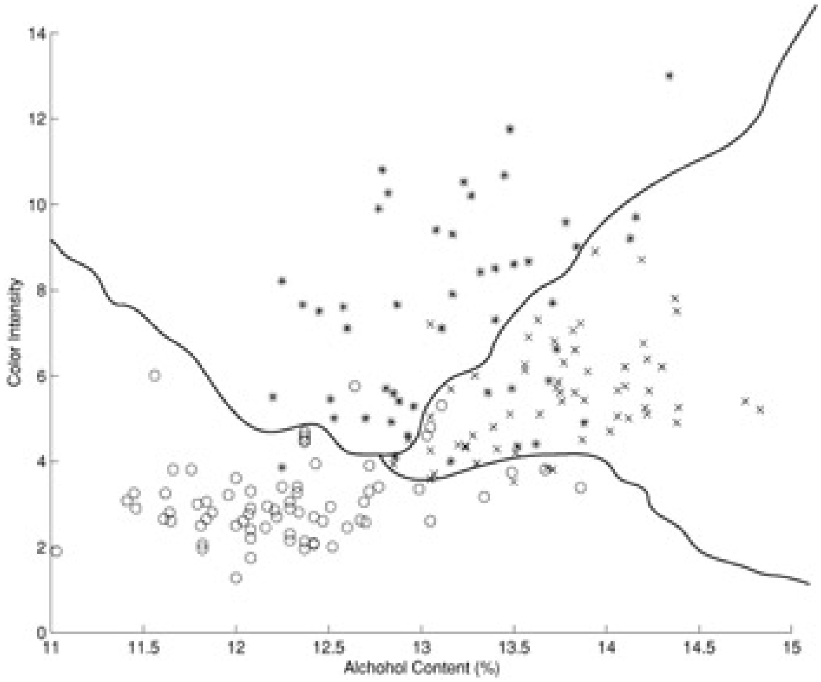


Figure 5.6: An example of the type of decision boundaries that a neural network model would produce for the two-dimensional wine data of figure 5.2(a).

The reductionist view of an MLP learning algorithm yields the following "algorithm-tuple":

1. task = prediction: classification or regression
2. structure = multiple layers of nonlinear transformations of weighted sums of the inputs
3. score function = sum of squared errors
4. search method = steepest-descent from randomly chosen initial parameter values
5. data management technique = online or batch

NOTES

The distinguishing feature of this algorithm is the multilayer, nonlinear nature of its model structure (note both that the output y is a nonlinear function of the inputs and that the parameters θ (the weights) appear nonlinearly in the score function). This clearly sets a neural network apart from more traditional linear and polynomial functional forms for regression and from tree -based models for classification.

The sum of squared errors (SSE), the most widely used score function for MLPs, is defined as:

$$S_{SSE} = \sum_{i}^{n} \left(y(i) - \hat{y}(i) \right)^2,$$

where y(i) and $\hat{y}$(i) are the true target value and the output of the network, respectively, for the ith data point, and where $\hat{y}$(i) is a function of the input vector **x**(i) and the MLP parameters (weights) θ . It is sometimes assumed that squared error is the only score function that can be used with a neural network model. In fact, as long as it is differentiable as a function of the model parameters (allowing us to determine the direction of steepest descent), any score function can be used as the basis for a steepest-descent search method such as backpropagation. For example, if we view squared error as just a special case of a more general log likelihood function, we can use a variety of other likelihood -based score functions in place of squared error, tailored for specific applications.

Training a neural network consists of minimizing S_{SSE} by treating it as a function of the unknown parameters θ (i.e., parameter estimation of θ given the data). Given that each $\hat{y}$(i) is typically a highly nonlinear function of the parameters θ , the score function S_{SSE} is also highly nonlinear as a function of θ . Thus, there is no closed-form solution for finding the parameters θ that minimize S_{SSE} for an MLP. In addition, since there can be many local minima on the surface of S_{SSE} as a function of θ , training a neural network (i.e., finding the parameters that minimize S_{SSE} for a particular data set and model structure) is often a highly non -trivial multivariate optimization problem. Iterative local search techniques are required to find satisfactory local minima.

The original training method proposed for MLPs, known as back propagation, is a relatively simple optimization method. It essentially performs steepest-descent on the score function (the sum of squared errors) in parameter space, solving this nonlinear optimization problem by descending to a local minimum given a randomly chosen starting point in parameter space. (In practice we usually descend from multiple starting points and select the best local minimum found overall.) In a more general context, there is a large family of optimization methods for such nonlinear optimization problems. It is often assumed that steepest-descent is the only optimization method that can be used to train an MLP, but in fact more powerful nonlinear optimization techniques such as conjugate gradient techniques can be brought to bear on this problem.

In terms of data management, a neural network can be trained either online (updating the weights based on cycling through one data point at a time) or in batch mode (updating the weights after seeing all of the data points). The online updating version of the algorithm is a special case of a more general class of online estimation algorithms.

5.3.2 The A Priori Algorithm for Association Rule Learning

An association rule is a simple probabilistic statement about the co-occurrence of certain events in a database, and is particularly applicable to sparse transaction data sets. For the sake of simplicity we assume that all variables are binary. An association rule takes the following form:

IF A = 1 AND B = 1 THEN C = 1 with probability p

where A, B, and C are binary variables and $p = p(C = 1|A = 1, B = 1)$, i.e., the conditional probability that C = 1 given that A = 1 and B = 5. The conditional probability p is sometimes referred to as the "accuracy" or "confidence" of the rule, and $p(A = 1, B = 1, C = 1)$ is referred to as the "support." This pattern structure or rule structure is quite simple and interpretable, which helps explain the general appeal of this approach. Typically the goal is to find all rules that satisfy the constraint that the accuracy p is greater than some threshold p_a and the support is greater than some threshold p_s (for example, to find all rules with support greater than 0.05 and accuracy greater than 0.8). Such rules comprise a relatively weak form of knowledge; they are really just summaries of co-occurrence patterns in the observed data, rather than strong statements that characterize the population as a whole. Indeed, in the sense that the term "rule" usually implies a causal interpretation (from the left to the right hand side), the term "association rule" is strictly speaking a misnomer since these patterns are inherently correlational but need not be causal.

The general idea of finding association rules originated in applications involving "market - basket data." These data are usually recorded in a database in which each observation consists of an actual basket of items (such as grocery items), and the variables indicate whether or not a particular item was purchased. We can think of this type of data in terms of a data matrix of n rows (corresponding to baskets) and p columns (corresponding to grocery items). Such a matrix can be very large, with n in the millions and p in the tens of thousands, and is generally very sparse, since a typical basket contains only a few items. Association rules were invented as a way to find simple patterns in such data in a relatively efficient computational manner.

In our reductionist framework, a typical data mining algorithm for association rules has the following components:

1. task = description: associations between variables
2. structure = probabilistic "association rules" (patterns)
3. score function = thresholds on accuracy and support
4. search method = systematic search (breadth-first with pruning)
5. data management technique = multiple linear scans

The score function used in association rule searching is a simple binary function. There are two thresholds: p_s is a lower bound on the support of the rule (e.g., $p_s = 0.1$ when we want only those rules that cover at least 10% of the data) and p_a is a lower bound on the accuracy of the rule (e.g., $p_a = 0.9$ when we want only rules that are at least 90% accurate). A pattern gets a score of 1 if it satisfies both of the threshold conditions, and gets a score of 0 otherwise. The goal is find all rules (patterns) with a score of 5.

The search problem is formidable given the exponential number of possible association rules namely, $O(p2^{p-1})$ for binary variables if we limit our attention to rules with positive propositions (e.g., A = 1) in the left and right -hand sides. Nonetheless, by taking advantage of the nature of the score function, we can reduce the average run -time of the algorithm to much more manageable proportions. Note that if either $p(A = 1) \leq p_s$ or $p(B = 1) \leq p_s$, clearly $p(A = 1, B = 1) \leq p_s$. We can use this observation in our search for association rules by first finding all of the individual events (such as A = 1) that have a probability greater than the threshold p_s (this takes one linear scan of the entire database). An event (or set of events) is called "frequent" if the probability of the event(s) is greater than the support threshold p_s. We consider all possible pairs of these frequent events to be candidate frequent sets of size 2.

In the more general case of going from frequent sets of size k - 1 to frequent sets of size k, we can prune any sets of size k that contain a subset of k - 1 items that themselves are not frequent at the k - 1 level. For example, if we had only frequent sets {A = 1, B = 1} and {B = 1, C = 1}, we could combine them to get the candidate k = 3 frequent set {A = 1, B = 1, C = 1}. However, if the subset of items {A = 1, C = 1} was not frequent (i.e., this item set were not on the list of frequent sets of size k = 2), then {A = 1, B = 1, C = 1} could not be frequent either, and it could safely be pruned. Note that this pruning can take place without searching the data directly, resulting in a considerable computational speedup for large data sets.

Given the pruned list of candidate frequent sets of size k , the algorithm performs another
linear scan of the database to determine which of these sets are in fact frequent. The confirmed frequent sets of size k (if any) are combined to generate all possible frequent sets containing k + 1 events, followed by pruning, and then another scan of the database, and so on until no more frequent sets can be generated. (In the worst case, all possible sets of events are frequent and the algorithm takes exponential time.

However, since in practice the data are often very sparse for the types of transaction data sets analyzed by these algorithms, the cardinality of the largest frequent set is usually quite small (relative to n), at least for relatively large support values.) The algorithm then makes one final linear scan through the data set, using the list of all frequent sets that have been found. It determines which subset combinations of the frequent sets also satisfy the accuracy threshold when expressed as a rule, and then returns the corresponding association rules.

Association rule algorithms comprise an interesting class of data mining algorithms in that the search and data management components are their most critical components. In particular, association rule algorithms use a systematic breadth-first, general-t o-specific search method that explicitly tries to minimize the number of linear scans through the database. While there exist numerous other rule-finding algorithms in the machine learning literature (with similar rule -based representations), association rule algorithms are designed specifically to operate on very large data sets in a relatively efficient manner. Thus, for example, research papers on association rule algorithms tend to emphasize computational efficiency rather than interpretation of the rules that the algorithms produce.

5.3.3 Vector-Space Algorithms for Text Retrieval

The general task of "retrieval by content" is loosely described as follows: we have a query object and a large database of objects, and we would like to find the k objects in the database that are most similar to the query object. We are all familiar with this problem in the context of searching through online collections of text. For example, our query could be a short set of keywords and the "database" could correspond to a large set of Web pages. Our task in this case would be to find the Web pages that are most relevant to our keywords.

Here we look at a generic text retrieval algorithm in terms of its components. One of the most important aspects of this problem is how similarity is defined. Text documents are of different lengths and structure. How can we compare such diverse documents? A key idea in text retrieval is to reduce all documents to a uniform vector representation, as follows. Let $t_1, ..., t_p$ be p terms (words, phrases, etc.). We can think of these as variables, or columns in our data matrix. A document (a row in our data matrix) is represented by a vector of length p, where the ith component contains the count of how often term t_i appears in the document. As with market -basket data, in practice we can have a very large data matrix (n in the millions, p in the tens of thousands) that is very sparse (most documents will have many zeros). Again, of course, we normally would not actually store the data as a large n × p matrix: a more efficient representation is to store a list for each term t_i of all the documents containing that term.

Given this "vector -space" representation, we can now readily define similarity. One simple definition is to make the similarity distance a function of the angle between the two vectors in p-space. The angle measures similarity in a given direction in "term - space" and factors out any differences arising from the fact that large documents tend to have more occurrences of a word than small documents. The vector-space representation and the angle similarity measure may seem relatively primitive, but in practice this scheme works surprisingly well, and there exists a multitude of variations on this basic theme in text retrieval.

With this information, we are ready to define the components of a simple generic text- retrieval algorithm that takes one document and finds the k most similar documents:

1. task = retrieval of the k most similar documents in a database relative to a given query
2. representation = vector of term occurrences
3. score function = angle between two vectors
4. search method = various techniques
5. data management technique = various fast indexing strategies

There are many variations on the specific definitions of the components given above. For example, in defining the score function, we can specify similarity metrics more general than the angle function. In specifying the search method, various heuristic search techniques are possible. Note that search in this context is real-time search, since the algorithm has to retrieve the patterns in real time for a user (unlike the data mining algorithms we looked at earlier, for which search meant off-line searching for the optimal parameters and model structures).

Different applications may call for different components to be used in a retrieval algorithm. For example, in searching through legal documents, the absence of particular terms might be significant, and we might want to reflect this in our definition of a score function. In a different context we might want the opposite effect, i.e., to down weight the fact that two documents do not contain certain terms (relative to the terms they have in common).

It is clear, however, that the model representation is really the key idea here. Once the use vector representation has been established, we can define a wide range of similarity metrics in vector -space, and we can use standard search and indexing techniques to find near neighbors in sparse p-dimensional space. Different retrieval algorithms may vary in the details of the score function or search methods, but most share the same underlying vector representation of the data. Were we to define a different representation for a document (say a generative model for the data based on some form of grammar), we would probably have to come up with fundamentally different score functions and search methods.

6. Models and Patterns

Structure

NOTES

Objective

After going through this lesson, you should be able to:

- Discuss about a model structures for prediction;
- Discuss models for probability distributions and density functions;
- Discuss about a curse of dimensionality;
- Discuss about a models for structured data and pattern structures;

6.1 Introduction

We have introduced the distinction between models and patterns in earlier chapters. Here we explore these ideas in more depth, and examine some of the major classes of models and patterns used in data mining, in preparation for a detailed examination in subsequent chapters.

A model is a high -level, global description of a data set. It takes a large sample perspective. It may be descriptive summarizing the data in a convenient and concise way or it may be inferential, allowing one to make some statement about the population from which the data were drawn or about likely future data values. In this chapter we will discuss a variety of basic model forms such as linear regression models, mixture models, and Markov models.

In contrast, a pattern is a local feature of the data, perhaps holding for only a few records or a few variables (or both). An example of a pattern would be a local "structural" feature in our p-dimensional variable space such as a mode (or a gap) in a density function or an inflexion point in a regression curve. Often patterns are of interest because they represent departures from the general run of the data: a pair of variables that have a particularly high correlation, a set of items that have exceptionally high values on some variables, a group of records that always score the same on some variables, and so on. As with models, we may want to find patterns for descriptive reasons or for inferential reasons. We may want to identify members of the existing database that have unusual properties, or we may want to predict which future records are likely to have unusual properties. Examples of patterns are transient waveforms in an EEG trace, unusual combinations of products that are frequently purchased together by retail customers, and outliers in a database of semiconductor manufacturing data.

Data compression can provide a useful way to illustrate the concept of patterns versus a model. Consider transmitter T that has an image I that is to be sent to a receiver R (though the principle holds for data sets that are not images). There are two main strategies: (a) send all of the data (the pixels in the image I) exactly, or (b) transmit some compressed version of the image that is, some summary of the image I. Data mining to a large extent corresponds to the second approach, the compression being achieved either by representing the original data as a model, or by identifying unusual features of the data through patterns.

In modeling, some loss in fidelity is likely to be incurred when we summarize the data this means that the receiver R will not be able to reconstruct the data precisely. An example of a model for the image data might be replacing each square of 16 × 16 pixels in the original image by the average values of these pixels. The "model" in this case would just be a set of smaller and lower resolution (1/16th) images. A more sophisticated model might adaptively partition each image into local regions of different sizes and shapes, where the pixel values can be fairly accurately described by constant pixel intensity within each such region. The "model" (or message) in this case would be both the values of the constants within each region and the description of the boundaries of the regions for each. For both types of models (the average-pixel model and the locally constant model) it is clear that the complexity of the image model (the number of pixels being averaged, the average size of the locally constant regions) can be traded for the amount of information being transmitted (or equivalently, the amount of information being lost in the transmission that is, the compression rate).

From a pattern detection viewpoint, a pattern in an image is some structure in the image that is purely local: for example, a partially obscured circular object in the upper -left corner of the image. This is clearly a different form of compression from the global compression models above. The receiver R can no longer reconstruct a summary of the whole image, but it does have a description of some local part of the image. Depending on the problem and objectives, local structure may be much more relevant than a global model. Rather than sending a summary model description of a vast noisy "sea" of pixel values, the

NOTES

transmitter T instead "focuses" the receiver R's attention on the important aspects.

The analogy between image coding and data analysis is not perfect (for example, compression, as we have described it, does not take into account the idea of generalization to unseen data), but nonetheless, it allows us to grasp the essential trade-offs between representing local structure at a fairly high resolution and lower resolution global structure.

6.2 Fundamentals of Modeling

A model is an abstract representation of a real-world process. For example, Y = 3X + 2 is a very simple model of how the variable Y might relate to the variable X. This particular model can be thought of as an instance of the more general model structure Y = aX + c, where for this particular model we have set a = 3 and c = 2. More generally still, we could put Y = aX + c + e, where e is a random variable accounting for a random component of the mapping from X to Y (we will return to this later). We often refer to a and c as the parameters of the model, and will often use the notation θ to refer to a generic parameter or a set (or vector) of parameters. In this example, θ = {a, c}. Given the form or structure of a model, we choose appropriate values for its parameters by estimation that is, by minimizing or maximizing an appropriate score function measuring the fit of the model to the data.

However, before we can estimate the parameters of a model, we must first choose an appropriate functional form of the mod el itself. The aim of this section is to present a high-level overview of the main classes of models used in data mining.

Model building in data mining is data-driven. It is usually not driven by the notion of any underlying mechanism or "reality," but simply seeks to capture the relationships in the data. Even in those cases in which there is a postulated true generative mechanism for the data, we should bear in mind that, as George Box put it, "All models are wrong but some are useful." For example, while we might postulate the existence of a linear model to explain the data, it is likely to be a fiction, since even in the best of circumstances there will be small nonlinear effects that we will be unable to capture in the model. We are looking for a model that encapsulates the main aspects of the data generating process.

NOTES

Since data mining is data-driven, the discovery of a highly predictive model (for example) should not be taken to mean that there is a causal relationship. For example, an analysis of customer records may show that customers who buy high quality wines are also more likely to buy designer clothes. Clearly one propensity is not causally related to the other propensity (in either direction). Rather, they are both more likely to be the consequence of a relatively high income. However, the fact that neither the wine nor the clothes variable causes the other does not mean that they are not useful for predictive purposes. Predicting the likely clothes-buying behavior from observed wine-buying behavior would be entirely legitimate (if the relationship were found in the data), from a marketing perspective. Since no causal relationship has been established, however, it would be false to conclude that manipulating one of the variables would lead to a change in the other. That is, inducing people to buy high-quality wines would be unlikely to lead them also to buy designer clothes, even if the relationship existed in the data.

6.3 Model Structures for Prediction

In a predictive model, one of the variables is expressed as a function of the others. This permits the value of the response variable to be predicted from given values of the others (the explanatory or predictor variables). The response variable in general predictive models is often denoted by Y, and the p predictor variables by $X_1 \dots X_p$. Thus, for example, we might want to construct a model for predicting the probability that an applicant for a loan will default, based on application forms and the behavior of past customers contained in a database. The record for the ith past customer can be conveniently represented as $\{(\mathbf{x}(i), y(i))\}$. Here y(i) is the outcome class (good or bad) of the ith customer, and $\mathbf{x}(i)$ is the vector $\mathbf{x} = (x_1(i), \dots, x_p(i))$ of application form values for the ith customer. The model will yield predictions, $y = f(x_1, \dots, x_p; \theta)$ where y is the prediction of the model and θ represents the parameters of the model structure. When Y is quantitative, this task of estimating a mapping from the p-dimensional **X** to Y is known as regression. When Y is categorical, the task of learning a mapping from **X** to Y is called classification learning or supervised classification. Both of these tasks can be considered function approximation problems in that we are learning a mapping from a p-dimensional variable **X** to Y. For simplicity of exposition in this chapter we will focus primarily on the regression task, since many of the same general principles carry over directly to the classification task.

NOTES

6.3.1 Regression Models with Linear Structure

We begin our discussion of predictive models with models in which the response variable is a linear function of the predictor variables:

$$\hat{Y} = a_0 + \sum_{j=1}^{p} a_j X_j$$

where θ = {a_0, . . . , a_p}. Again we note that the model is purely empirical, so that the existence of a well-fitting and highly predictive model does not imply any causal relationship. We have used Y rather than simply Y on the left of this expression because it is a model, which has been constructed from the data. That is, the values of Y are values predicted from the **X**, and not values actually observed.

Geometrically, this model describes a p-dimensional hyperplane embedded in a (p + 1)- dimensional space with slope determined by the a_j 's and intercept by a_0. The aim of parameter estimation is to choose the a values to locate and angle this hyperplane so as to provide the best fit to the data {(**x**(i), y(i))}, i = 1, . . . , n, where the quality of fit is measured in terms of the differences between observed y values and the values y predicted from the model.

Models with this type of linear structure hold a special place in the history of data analysis, partly because estimation of parameters is straightforward with appropriate score functions, and partly because the structure of the model is simple and easy to interpret. For example, the additive nature of the model means that the parameters tell us the effect of changing any one of the predictor variables "keeping the others constant." Of course, there are circumstances in which the notion of individual contribution makes little sense. In particular, if two variables are highly correlated, then it is not meaningful to talk of the contribution from changing one while "holding the other constant."

We can retain the additive nature of the model, while generalizing beyond linear functions of the predictor variables. Thus

$$\hat{Y} = a_0 + \sum_{j=1}^{p} a_j f_j(X_j)$$

where the f_j functions are smooth (but possibly nonlinear) functions of the X_js. For example, the f_js could be log, square-root, or related transformations of the original X variables. This model still assumes that the dependent variable Y depends on the independent variables in the model (the Xs) in an additive fashion. Again, this may be a strong assumption in practice, but it will lead to a model in which it may be easy to interpret the contribution of each individual X variable. The simplicity of the model also means that there are relatively few parameters (p + 1) to estimate from the data, making the estimation problem relatively straightforward.

We can also generalize this linear model structure to allow general polynomials in the Xs with cross-product terms to allow interaction among the Xjs in the model. The one -dimensional case is again familiar we can imagine a 2nd or 3rd or k th order polynomial interpolating the observed y values. The multidimensional case generalizes this so that we have a smooth surface defined on p variables in the (p + 1) dimensional space.

Note in passing that even though these predictive models are nonlinear in the variables X, they are still linear in the parameters. This makes estimation of these parameters much easier than in the case where the parameters enter in a nonlinear fashion.

Note that by allowing models with higher order terms and interactions between the components of **X** we can in principle estimate a more complex surface than with a simple linear model (a hyperplane). However, note that as p (the dimensionality of the input space) increases, the number of possible interaction terms in our model (such as X_jX_k) increases as a combinatorial function of p. Since each term has a weight coefficient (a parameter) in the additive model, the number of parameters to be estimated for the full model (with all possible interaction terms of order k among p variables) increases dramatically as p increases. The interpretation and understanding of such a model makes the estimation problem more difficult, and it also becomes increasingly difficult as p increases. A practical alternative is to select some small subset of the overall set of possible interactions to participate in the model. However, if the selection is carried out in a data-driven fashion (as is typically the fashion in a data mining application), the number of all possible interaction terms (the size of the search space) scales as 2^p, making the search problem exponentially more difficult as dimensionality p increases. We will return to this issue of how to handle dimensionality later in this chapter.

The generalization to polynomials brings up an important point, namely the complexity of the model. The more complex models contain the simpler models as special cases (so- called nesting).

For example, the first-order $\alpha_1 X_1 + \alpha_0$ model can be viewed as a special case of the 2nd order polynomial model $\alpha_2 X_1^2 + \alpha_1 X_1 + \alpha_0$ by setting a2 to zero. Thus, it is clear that a complex model (a high-order polynomial in the **X** variables) can always fit the observed data at least as well any simpler model can (since it includes any simpler model as a special case). In turn, this raises the complicated issue of how we should choose one model over another when the complexity (or expressive power) of each is different. This is a subtle question: we may want the model that is closest to some hypothesized unknown "truth"; we may want to find a model that captures the main features of the data without being too complicated; we may want to find the model that has the best predictive performance on data that it has not seen; and so on. We will return to this in later chapters. For now, however, we return to focus on the expressive capabilities of the models themselves without thinking yet of how we will choose among such models given observed data.

Transforming the predictor variables is one way to generalize a linear structure. Another way is to transform the response variable. sqrt(Y) may be perfectly related to a linear combination of the **X** variables, so that rather than fitting Y directly we may want to transform it by taking the square root first, and then use a linear combination of the **X** variables to predict sqrt(Y). Of course, we will not know beforehand that the square root is an appropriate transformation. We have to experiment, trying different transformations. This is why data mining is an exciting voyage of discovery, and not a mere exercise in applying standard tools in standard ways.

The simple linear regression model can be thought of as seeking to predict the expected value of the Y distribution at each value of the **X** predictors, namely E[Y|X]. That is, the regression model provides a prediction of a parameter of the conditional distribution of Y, where the parameter is the mean. More generally, of course, we can seek to predict other parameters of the conditional Y distribution from a linear combination of the **X** variables. This leads to the ideas of generalized linear models and neural networks.

We see that, although linear models are simple and easy to interpret (and, we will also see, their parameters can be easily estimated), they permit ready generalization to very powerful and flexible families of models. Any idea that the word linear implies a narrow class of models is illusory.

NOTES

6.3.2 Local Piecewise Model Structures for Regression

Yet further generalizations of the basic linear model can be achieved if we assume that Y is locally linear in the X's, with a different local dependence in various regions of the **X** space that is, a piecewise linear model. Geometrically, our model structure consists of a set of different p-dimensional hyperplanes, each covering a region of the input (**X**) space disjoint from the others. The parameters of this model structure include both the local parameters for each hyperplane as well as the locations (boundaries) of the hyperplanes. For a one -dimensional X the picture is quite easy to visualize: a curve is approximated by k different line segments (see figure 6.1 for an example). Note that, in this figure, the line is continuous, with the line segments joining up. We could define a model structure that relaxes this, not requiring continuity at the ends of the line segments. This can be a useful model form, but sometimes the discontinuities can be problematic and undesirable because they imply a sudden jump in the predicted value of the response variable for an infinitesimal change in a predictor variable. To take an example, if a split between two line segments occurs at the value $50,000 for the variable income, we might get widely varying y predictions of the response variable, probability of loan default, for two applicants who are identical except that one earns $50,001 and the other earns $49,999. If the discontinuities are regarded as undesirable, one can go further and enforce continuity of derivatives of various orders at the end of the segments (which would clearly no longer be straight lines). Such curve segments are termed splines , with the whole model being a spline function. Typically, each line segment is taken to be a low-degree (quadratic or cubic) polynomial. The result is a smooth curve, but one that may change direction many times the model would be highly flexible.

These ideas can be generalized to more than one predictor variable. Again the local segments (which will now be (hyper) surfaces, not merely lines) may, but need not, join at their edges. Tree structures provide an example of models of this form.

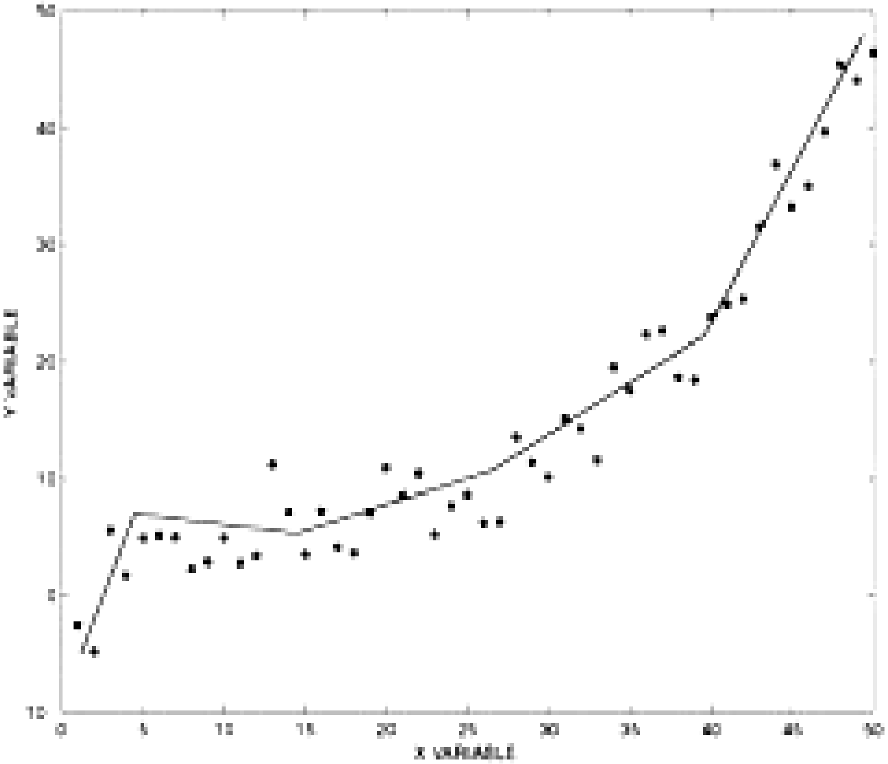


Figure 6.1: An example of a piecewise linear fit to the data of figure 6.1 with k = 5 linear segments.

6.3.3 Nonparametric "Memory-Based" Local Models

In the preceding subsection we gave some examples of how models that are based on local characteristics of the data are related to, indeed are on a continuum including, broad global models. In this subsection we develop the ideas of local modeling further. (We recall that patterns, while also local, are isolated structures, and are not components of a global summary of the data. Thus we can talk of local modeling techniques as distinct from patterns.)

Roughly speaking, the spline and tree models briefly described above replace the data points by a function estimated from a neighborhood of data points. An alternative strategy is to retain the data points, and to leave the estimation of the predicted value of Y until the time at which a prediction is actually required. No longer are the data replaced by a function and its estimated parameters. For example, to estimate the value of a response variable Y for a new case, we could take the average of the Y values of the most similar k objects in the data set, where most similar is defined in terms of the predictor variables.

This idea has been extended to include all of the data set objects, but to weight them according to how similar they are to the new object dissimilar ones will have small weight, similar ones large weight. The weight determines just how much their Y value contributes to the final estimate. An example of such an estimator is the locally weighted regression or loess regression model.

The obvious question with such estimators is how to determine the form of the weight function. A weight function that decays only slowly with decreasing similarity will lead to a smooth estimate, while one that decays rapidly will lead to a jagged estimate. A compromise must be found that is best suited to the aims of the analysis.

The weight function can be decomposed into two parts. One is its precise functional form, and the other is its "bandwidth." Thus, suppose that $k\left(\frac{x-z}{h}\right)$ is a smoothing function, which determines the contribution to the estimate at a new point z from a data set point at x. The size of this contribution will depend on the form of K and also on the size of the bandwidth h. A larger bandwidth h leads to a smoother function estimate, and a smaller bandwidth leads to a rougher, more jagged estimate. In practice, the precise form of the weight function turns out to be less important than the "band-width."

Kernel methods are closely related to nearest neighbor methods. Indeed, both classes of methods have now been extended and developed so that in some cases they are identical. Whereas kernel methods define the degree of smoothing in terms of a kernel function and bandwidth, nearest neighbor methods let the data determine the bandwidth by defining it in terms of the number of nearest neighbors. For example, the basic single nearest neighbor classifier (where Y is a class identifier) assigns a new object to the same class as its most similar object in the data set, and the k -nearest neighbor classifier assigns a new object to the most common class amongst the most similar k objects in the data set. More sophisticated nearest neighbor methods weight the contribution of according to distance from the point to be classified, and more sophisticated kernel methods let the bandwidth h depend on the data so that they can be seen to be almost identical in terms of model structure.

Local model structures such as kernel models are often described as non-parametric because the model is largely data-driven with no parameters in the conventional sense (except for the bandwidth

NOTES

h). Such data-driven smoothing techniques (such as the kernel models) are useful for data interpretation, at least in one or two dimensions.

It will be clear that local models have their attractions. However, no model provides an answer to all problems, and local models have weaknesses. In particular, as the number of variables in the predictor space increases, so the number of data points required to obtain accurate estimates increases exponentially. This means that these "local neighborhood" models tend to scale poorly to high dimensions.

Another drawback, particularly from a data mining viewpoint, is the lack of interpretability of the model. In low dimensions ($p \leq 3$ or so), we can plot the estimates. In high dimensions this is not possible, and there is no direct manner by which to summarize the model. Indeed, it is stretching the definition of a model to even call these representations models at all, since they are never explicitly defined as functions but instead are implicitly defined by the data.

6.3.4 Stochastic Components of Model Structures

Until this point, apart from a few brief references, we have ignored the fact that, with real data, we generally cannot find a perfect functional relationship between the predictor variables **X** and the response variable Y . That is, for any given vector of predictor variables **x**, more than one value of Y can be observed. The distribution of the values y at each value of **X** represents an aspect of variation that cannot be reduced by more sophisticated model building using just the variables in **X**. For this reason it is sometimes termed the unexplainable or nonsystematic or random component of the variation, with the variation in Y that can be explained in terms of the **X** variables being termed the explainable or systematic variation. (Of course merely because the systematic variation can be explained in principle by the variables in **X**, does not mean that we can necessarily build a model that will be able to do it).

In most of our discussion we have focused on the systematic component of the models, but we also need to consider the random component. The random component of models can arise from many sources. It can arise from simple measurement error repeated measurements of Y will give different results. The random component can also arise because our set of **X** variables does not include all of the variables that are required to make a perfect prediction of Y (for example, predicting whether a customer will purchase a particular product or not based only on past purchasing behavior will ignore potentially relevant demographic information about them such as age, income, and so on). Indeed, we should expect this usually to be the case it would be a rare situation in which all of the variability in a variable was perfectly explained by just a handful of other variables, down to the finest detail.

The random component is important when it comes to choosing suitable score functions for estimating parameters and choosing between models. The likelihood score function. Extensions of the likelihood function that include a smoothness penalty so that too complex a model is not fitted also require assumptions about the distribution of the random component. More advanced methods based on likelihood concepts (for example, so-called quasi-likelihood methods) relax detailed distributional assumptions, but still base their choice of parameter estimates on aspects of the distribution of the random component.

6.3.5 Predictive Models for Classification

So far we have concentrated on predictive models in which the variable to be predicted, Y, was quantitative. We now briefly consider the case of a categorical variable Y, taking only a few possible categorical values. This is a (supervised) classification problem, with the aim being to assign a new object to its correct class (that is, the correct Y category) on the basis of its observed **X** values.

In classification we are essentially interested in modeling the boundaries between classes. As with regression, we can could make simple parametric assumptions about the functional form of the boundaries. For example, a classic approach is to use a linear hyper plane in the p-dimensional **X** space to define a decision boundary between two classes. That is, the model partitions the **X**-space into disjoint decision regions (one for each class), where the decision regions are separated by linear boundaries. A more complex model might allow higher-order polynomial terms, yielding smooth polynomial decision boundaries.

There are a large number of different classification techniques, providing different ways to model decision boundaries. Something like nearest-neighbor is very flexible (allowing multiple local disjoint decision regions for each class, with flexible boundaries) whereas a single global hyperplane is a much simpler model.

NOTES

From a practical modeling standpoint, prior knowledge about the shape of classification boundaries may not be as readily available as knowledge we may have about how Y is related to **X** in a regression problem. Nonetheless, the functional forms used successfully for discrimination models are quite similar to those we discussed earlier for regression modeling, and the same general themes emerge.

6.3.6 An Aside: Selecting a Model of Appropriate Complexity

In our discussion so far we have seen that model structures range from the relatively simple to the complex. For example, in regression we saw that the complexity of a "piecewise-local" model structure is controlled by the number k of local regions (assuming that the complexity of the local function in each region is fixed). As we make k larger, we can obtain a curve that "follows" the observed data more closely. Put another way, the expressive power of the model structure increases in that it can represent more complex functions.

As we increase the expressive power of a model it is clear that we can in general continue to get a better fit to the available data. However, we need to be careful. While our score function on the training data may be improving, our model may actually be getting worse in terms of generalizing to new data. (Recall our discussion of this "overfitting" phenomenon in the context of classification trees. On the other hand, if we go the other direction and over-simplify our model structure, it may end being too simple. This issue of selecting a model of the appropriate complexity is always a key concern in any data analysis venture where we consider models of different complexities.

In practice how can we choose a suitable compromise between simplicity and complexity? From a data-driven viewpoint (i.e., data mining) we can define a score function that tries to estimate how well a model will perform on new data and not just on the training data. A commonly used approach is to combine both the usual goodness-of- fit term (on the training data) with an explicit second term to penalize model complexity. Another widely used approach is to partition the training data into two or more subsets and to train models on one subset and select models using a different validation data set.

Since the focus of this chapter is on the representational capabilities of different model and pattern structures (rather than on how they are scored relative to the data). However, for the reader who up to this point was wondering how we would be able to select among the many different models being discussed here, the answer is that there do indeed exist well-defined data-driven score functions that allow us to search over different model structures in a principled manner to find what appears to be the best model for a given task.

6.4 Models for Probability Distributions and Density Functions

6.4.1 General Concepts

In this section we focus on some of the general classes of models used for density estimation. While the functional form of the underlying models tend to be somewhat different from those we have seen earlier (for example, unimodal "bump" functions versus the linear and polynomial functions we saw for regression), several of the main concepts such as linear combinations of simpler models are once again widely applicable.

There are two general classes of distribution and density models:

1. Parametric Models:

Where a particular functional form is assumed. For real-valued variables the function is often characterized by a location parameter (the mean) and a scale parameter (characterizing the variability) for example, the Normal density function and Binomial distribution. Parametric models have the advantage of simplicity (easy to estimate and interpret) but may have relatively high bias because real data may not obey the assumed functional form. The appendix contains a brief review of some of the more well-known parametric density and distribution models.

2. Nonparametric Models:

Where the distribution or density estimate is data-driven and relatively few assumptions are made a priori about the functional form. For example, we can use the kernel estimates local density at x is defined as a weighted average of points near to x.

Taking the above as the extremes, we can also define intermediate models that lie between these parametric and nonparametric extremes: mixture models. These are discussed below.

6.4.2 Mixtures of Parametric Models

A mixture density for **x** is defined as

$$p(\mathbf{x}) = \sum_{k=1}^{K} p_k(\mathbf{x}|\theta_k)\pi_k.$$

This model decomposes the overall density (or distribution) for **x** into a weighted linear combination of K component or class densities (or distributions). Each of the component densities $p_k(\mathbf{x}|\theta_k)$ typically consists of a relatively simple parametric model (such as a Normal distribution) with parameters θ_k. π_k represents the probability that a randomly chosen data point was generated by component k ,

$$\sum_k \pi_k = 1$$

To illustrate, consider a single Normal distribution used as a model for a two-dimensional data set. This distribution can be thought of as a "symmetric bump function," whose location and shape we can to try to locate in the 2-space to model the density of the data as well as possible (see figure 6.2 for a simple example). An intuitive interpretation of the mixture model is that it allows us to place k of these bumps (or components) in the two - dimensional space to approximate the true density. The locations and shapes of the k bump functions can be fixed independently of each other. In addition, we are allowed to attach weights to the components. If the weights are positive and sum to 1 the overall function is still a probability density.

As k increases, the mixture model allows for quite flexible functional forms, as local bumps can be placed to capture local characteristics of the density (this is reminiscent of the local modeling ideas in regression). Clearly k plays the role of controlling complexity: for larger k we get a more flexible model but also one that it is more complicated to interpret and more difficult to fit.

The usual bias -variance trade -offs again apply. Of course, we are not constrained to use only Normal components (although these tend to be quite popular in practice). Mixtures of exponentials a nd other densities could equally well be used. The important point here is that mixtures provide a natural generalization of the simple parametric density model (which is global) to a weighted sum of these models, allowing local adaptation to the density of the data in p-space.

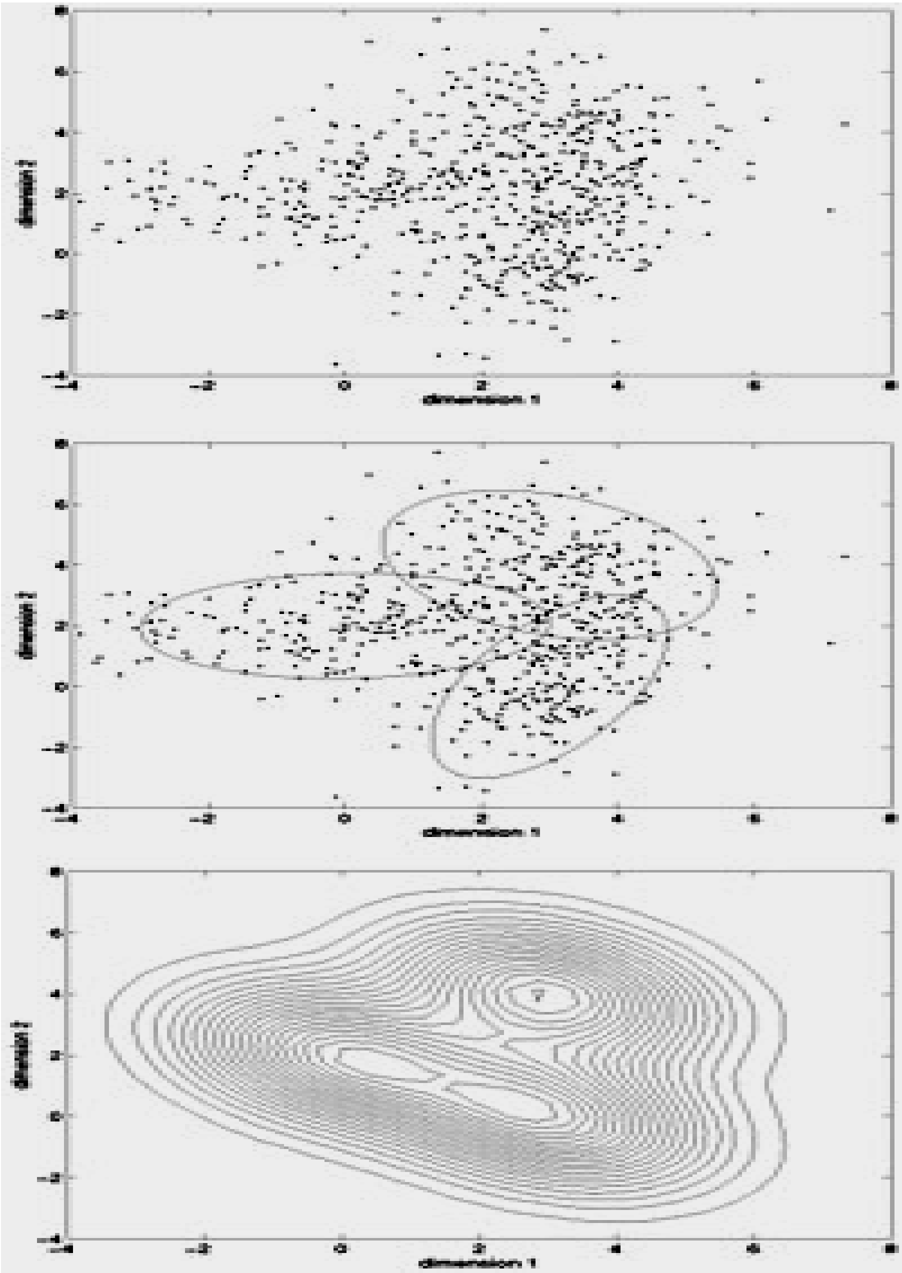


Figure 6.2: From the top: (a) data points generated from a mixture of three bivariate normal distributions (appendix 1) with equal weights, (b) the underlying component densities plotted as contours that are located 3s from the means, and (c) the resulting contours of the overall mixture density function.

NOTES

The general principles underlying a mixture model are broadly applicable, and the general idea occurs in many guises in probabilistic model building. For example, the idea of hierarchical structure can be nicely captured using mixture models.

In terms of interpretability, either mixture models can be used simply as "black boxes" that provide a flexible model form, or the individual mixture components can be given an explicit interpretation. For example, components of a mixture model fitted to customer data could be interpreted as characterizing different types of customers. One interpretation of a mixture model (particularly in a clustering context) is that the components are generated by a hidden variable taking K values, and the location and shapes in p-space of the components are unknown to us a priori, but may be revealed by the data. Thus, mixture models share with projection pursuit and related methods the general idea of hypothesizing a relatively simple latent or hidden structure that may be generating the observed data.

6.4.3 Joint Distributions for Unordered Categorical Data

For categorical data we have a joint distribution function defined in the cross-product of all possible values of the p individual ariables. For example, if A is a variable taking values {a_1, a_2, a_3} and B is a variable taking values {b_1, b_2}, then there are six possible values for the joint distribution of A and B. We will assume here (for simplicity) that the values are truly categorical and that there is (for example) no notion of scale or order. For small values of p, and for small numbers of variable values, it is convenient to display the values of the distribution in the form of a contingency table of cells, one cell per joint value, as shown in the example of table 6.1. This becomes impractical as the number of variables and values get beyond four or five. In addition, the contingency table does not really allow us to see any potential structure that might be in the data. For example, the data in table 6.1 have been constructed so that the variables are independent: however, this fact is not immediately apparent from looking at the table.

Table 6.1: A simple contingency table for two-dimensional categorical data for a hypothetical data set of medical patients who have been diagnosed for dementia.

Smoker	Dementia		
	none	mild	serve
No	426	66	132
Yes	284	44	88

In contrast to the case of quantitative variables, with categorical variables in which the categories are unordered there is no notion of a smooth probability function. Thus, if for example all variables each have m possible values, one would have to specify $m^p - 1$ independent probability values to specify the model fully (the -1 comes from the constraint that they sum to 1). Clearly this quickly becomes impractical as p and m increase.

6.4.4 Factorization and Independence in High Dimensions

Dimensionality is a fundamental challenge in density and distribution estimation. As the dimensions of the **x**, space grow it rapidly becomes more difficult to construct fully specified model structures since model complexity tends to grow exponentially with dimension.

Factorization of a density function into simpler component parts provides a general technique for constructing simple models for multivariate data. This is a simple yet powerful idea that recurs throughout multivariate modeling. For example, if we assume that the individual variables are independent , we can write the joint density function as

$$p(\mathbf{x}) = p(x_1, \ldots, x_p) = \prod_{k=1}^{p} p_k(x_k)$$

where $\mathbf{x} = (x_1, \ldots, x_p)$ and p_k is the one-dimensional density function for X_k. Typically it is much simpler to model the one-dimensional densities separately, than to model their joint density. Note that the independence model for log p(**x**) has an additive form, reminiscent of the linear and additive model structures we discussed for regression.

This factorization certainly simplifies things, but it has come at a modeling cost. The assumption that the variables are independent will not be even approximately true for many real problems. Thus, a full independence assumption is in essence one extreme end of a spectrum (the low -complexity end), a spectrum that extends to the fully specified joint density model at the other end (the high-complexity end). Of course, we do not have to choose models solely from the extremes of this complexity continuum, and can, instead, try to find something in between. The joint probability function p(**x**) can be written in general as

$$p(\mathbf{x}) = p_1(x_1)\prod_{k=2}^{p} p(x_k|x_1,\ldots,x_{k-1})$$

The righthand side factorizes the joint function into a sequence of conditional distributions. Now we can try to model each of those conditional distributions separately. Often considerable simplification results because each variable X_k is dependent on only a few of its predecessors. That is, in the conditional distribution for the k th variable, we can often ignore some of variables $X_1, \ldots, X_{k-1}$. Such factorizations permits a natural representation of the model as a directed graph, with the nodes corresponding to variables, and the edges showing dependencies between the variables. Thus the edges directed into the node for the k th variable will be coming from (a subset of) the variables $x_1, \ldots, x_{k-1}$. These variables are, naturally enough, called the parents of variable x_1.

Sometimes we have to experiment by fitting different models to the data to seek such simplifying factorizations. In other cases such simplifications will be evident from the structure of the data for example, if the variables represent the same property measured sequentially (for instance, at different times). In this case, a Markov chain model is often appropriate in which all of the previous information relevant to the k th variable is contained in the immediately preceding variable (so that the terms in this factorization simplify to $p(x_k|x_1, \ldots, x_{k-1}) = p(x_k|x_{k-1})$). The model structure for a first- order Markov model is shown in figure 6.3.

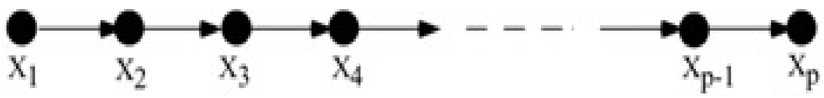


Figure 6.3: A graphical model structure corresponding to a first-order markov assumption.

Graphs that are used represent probability models, such as that in figure 6.3 are often referred to as graphical models. In the discussion below we focus specifically on the widely-used subclass of acyclic directed graphs (also sometimes known in computer science as belief networks when used as probability models). Note that this graph representation emphasizes the independence structure of the model (e.g., see figure 6.3 again) and leaves the actual functional and numeric parametrization of parent -child relationships unspecified.

For another example of a graphical model, consider the variables age, education (level of education a person has) and baldness (whether a person is bald or not). Clearly age cannot depend on either of the other two variables. Conversely, both education and baldness are directly dependent on age. Furthermore, it is quite implausible that education and baldness are directly dependent on each other given age that is, once we know the person's age, knowing whether or not they are bald tells us nothing about their education level (and vice versa). On the other hand, if we do not know a person's age, then baldness may provide information about education (for example, a bald person is more likely to be older, and hence, in turn, more likely to have a university degree). Thus, a plausible graphical model is the one in figure 6.4.

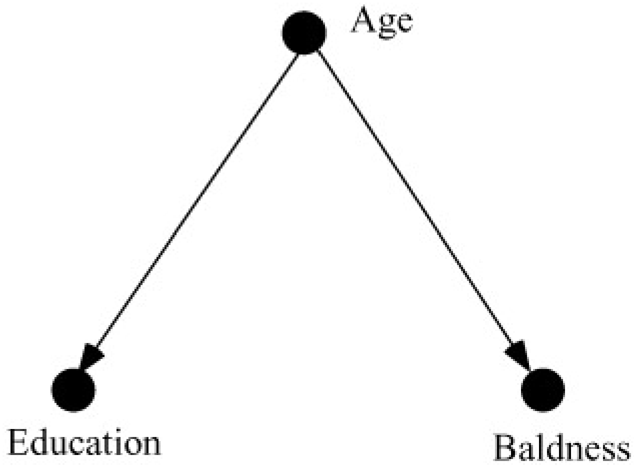


Figure 6.4: A plausible graphical model structure for two variables education and baldness that are conditionally independent given age.

These ideas can be taken further, by the postulation of the existence of unobserved hidden or latent variables, which explain many of the observe d relationships in the data. Figure 6.5 provides such an example. In this model structure a single latent variable has been introduced as an intermediate variable that simplifies the relationship between the observed data (in this case, medical symptom) and the underlying causal factors (here, two independent diseases). The introduction of hidden variables in a

NOTES

manner such as this can serve to simplify the relationships in a model structure; for example, given the values here of the intermediate variable, the symptoms become independent. However, we must exercise discretion in practice in terms of how many hidden variables we introduce into the model structure to avoid introducing spurious structure into the fitted model. In addition, parameter estimation and model selection with hidden variables is quite nontrivial.

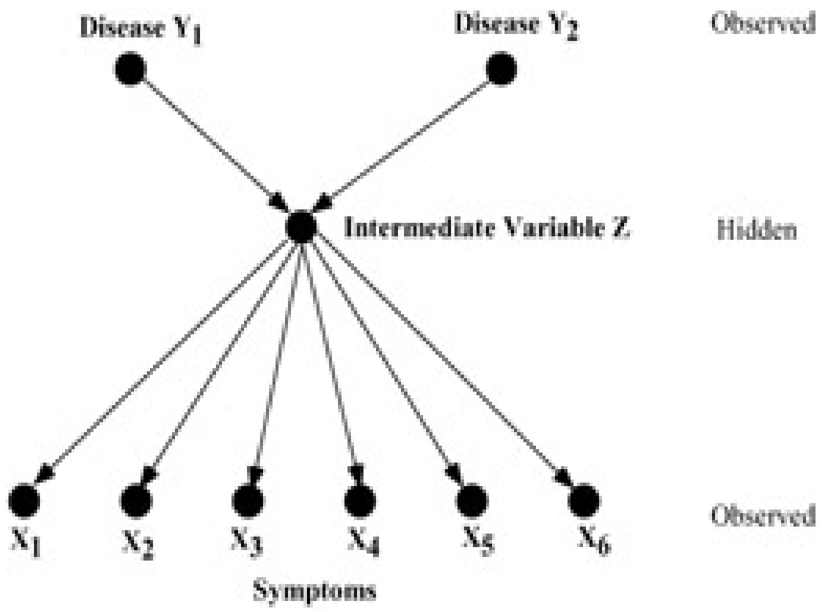


Figure 6.5: The graphical model structure for a problem with two diseases that are marginally (unconditionally) independent, a single intermediate variable z that directly depends on both diseases, and six symptom variables that are conditionally independent given z.

In the context of classification and clustering, it is often convenient to assume that the variables are conditionally independent of each other given the value of the class variable. That is,

$$p(\mathbf{x}|y) = \prod_{j=1}^{p} p_j(x_j|y),$$

where y is a particular (categorical) class value. This is simply the conditional independence ("naive") Bayes model introduced in the context of classification modeling. The graphical representation for such a model is shown in figure 6.6.

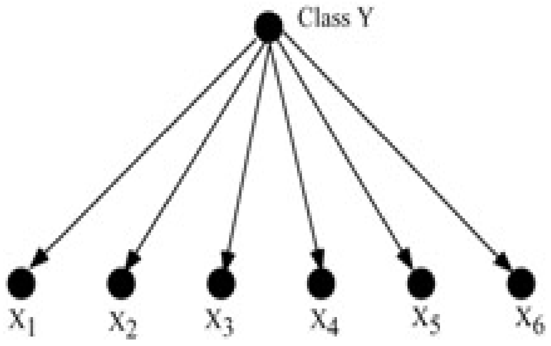


Figure 6.6: The first-order bayes graphical model structure, with a single class y and 6conditionally independent feature variables x_1, . . . , x_6.

The above equation can also be used in the case where Y is an unobserved (hidden, latent) variable that is introduced to simplify the modeling of p(**x**), i.e., we have a finite mixture of the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \left(\prod_{j=1}^{p} p_j(x_j | y = k) \right) p(y = k),$$

where Y takes K values, and each component p(**x**|y = k) is modeled using the conditional independence assumption of equation p(**x**|y). As an example, we might model the joint distribution of how customers purchase p products in this fashion, where (for example) if a customer belongs to a specific component k then the likelihood of purchasing certain subsets of products, i.e., $p_j(x_j|y = k)$, is increased for certain subsets of products xj. Thus, although the products (the x_j) are modeled as being conditionally independent given y = k, the mixture model induces an unconditional (marginal) independence by virtue of the fact that certain products co-occur with higher probability in certain components k . In effect, the hidden Y variable acts to group the variables x_j together into equivalence classes, where within each equivalence class the variables are modeled as being conditionally independent.

6.5 The Curse of Dimensionality

We have noted in various places that what works well in a one-dimensional setting may not scale up very well to multiple dimensions. In particular, the amount of data we need often

NOTES

increases exponentially with dimensionality if we are to maintain a specific level of accuracy in our parameter or function estimates. This is sometimes referred to as the "curse of dimensionality." This can be important, since data miners are often interested in finding models and patterns in high-dimensional problems. Note that "high - dimensional" can be as few as p = 10 variables or as many as p = 1000 variables or beyond it depends on the complexity of the models concerned and on the size of the available data.

There are two basic (and fairly obvious) strategies for coping with high-dimensional problems. The first is simply to use a subset of relevant variables to construct the model. That is, to find a subset of p' variables where p' << p. The second is to transform the original p variables into a new set of p' variables, where again p' << p. Examples of this approach include of p principal component analysis, projection pursuit, and neural networks.

6.5.1 Variable Selection for High-Dimensional Data

Variable selection is a fairly general (and sensible) strategy when dealing with high - dimensional problems. Consider for example the problem of predicting Y using $X_1, \dots, X_p$. It is often plausible that not all of the p variables are necessary for accurate prediction. Some X variables may be completely unrelated to the predictor variable Y (for example, the month of a person's birth is unlikely to be related to their creditworthiness). Others may be redundant in the sense that two or more X variables contain essentially the same predictive information. (For example, the variables income before tax and income after tax are likely to be highly correlated.)

We can use the notion of independence to gauge relevance in a quantitative manner. For example, if $p(y \mid x_1) = p(y)$ for all values of y and x_1, then the target variable Y is independent of input variable X_1. If $p(y \mid x_1, x_2) = p(y \mid x_2)$, then Y is independent of X_1 if the value of X_2 is already known. In practice, of course, we are not necessarily able to identify from a finite sample which variables are independent and which are not; that is, we must estimate this effect. Furthermore, we are interested not only in strict independence or dependence, but also in the degree of dependence. Thus, we could (for example) rank individual X variables in terms of their estimated linear correlation coefficient with Y: that would tell us about estimated individual linear dependence. If Y is categorical (as in classification), we could measure the average mutual information between Y and X':

NOTES

$$I(Y;X') = \sum_{i,j} p(y_i, x'_j) \log \frac{p(y_i, x'_j)}{p(y_i)p(x'_j)}$$

to provide an estimate of the dependence of X and Y, where X' here is a categorical variable (for example, a quantized version of a real-valued X). Other measures of the relationship between Y and the Xs can also be used.

However, the interaction of individual X variables with Y does not necessarily tell us anything about how sets of variables may interact with Y . The classic example, for Boolean variables, is the parity function, where Y is defined to be 1 if the sum of the (binary) values of the $X_1, \ldots, X_p$ variables in the set is an even integer, and Y is 0 otherwise. Y is independent of any individual X variable, yet is a deterministic function of the full set. While this is something of an extreme example, it nonetheless illustrates that such non-linear non -additive interactions can be masked if we only look at individual pair-wise interactions between Xs and Y. Thus, in the general case, the set of k best individual X variables (as ranked by correlation for example) is not the same as the best set of X variables of size k . Since one can have $2^p - 1$ different nonempty subsets of p variables, exhaustive search is not feasible except for very small p. Worse still, for many prediction problems, there is no optimal search algorithm (in the sense of being guaranteed to find the best set of variables) that has worst-case time complexity any better than $O(2^p)$.

This means that, in practice, subset selection methods tend to rely on heuristic search to find good model structures. Many algorithms are based on the simple heuristic of greedy selection, such as adding or deleting one variable at a time.

6.5.2 Transformations for High -Dimensional Data

The second general category of ideas is based on transforming the predictor variables. The intuitive idea here is to search for a set of p' variables (let us call them $Z_1, \ldots, Z_{p'}$),

where typically p' is much smaller than p, where the Z variables are defined as functions of the original X variables, and where the Zs are chosen in some sense to be the "best" set of p' variables for our task. This general theme, of replacing the observed variables with a smaller set of variables that are somehow more fundamental to the task at hand, shows up repeatedly in different

NOTES

branches of data analysis. The Zs are variously referred to as basis functions, factors, latent variables, principal components, and so forth, depending on the specific goals and methods used to derive them. We will examine some of these models (and their associated fitting algorithms) in detail in later chapters, but for now we illustrate the general idea with just two specific examples:

- **Projection Pursuit**: Regression uses a model structure of the form

$$\hat{y} = \sum_{j=1}^{p'} w_j h_j(\alpha_j^T \mathbf{x})$$

where **x** is the projection of the vector **x** onto the jth weight vector a_j (both vectors being p-dimensional, resulting in a scalar inner product), h_j is a nonlinear function of this scalar projection, and the w_j are scalar weights for the resulting nonlinear functions. The procedures for determining the w_j, the form of the h_j, and the "projection directions" a_j can be rather complex and algorithm-dependent, but the underlying idea is quite general.

For example, this is essentially the form of the model structure that underlies neural networks, where for such networks the functional forms of the h_j are usually chosen to be something like $h_j(t) = 1/(1 + e^{-t})$. One limitation of this class of models is the fact that they are quite difficult to interpret unless p' = 1. Another limitation is that the algorithms for estimating the parameters of these models can be computationally quite complex and may not be practical for very large data sets.

- **Principal Components Analysis**: We introduced principal components analysis (PCA). This is a classic technique in which the original p predictor variables are replaced by another set of p variables ($Z_1, \ldots, Z_p$) that are formed from linear combinations of the original variables. The data vectors comprising the original data set map to new vectors in the **Z** space and, as explained, the sets of weights defining the Zs are chosen so as to maximize the variance of the original data set when expressed in terms of these new variables. Principal components analysis is thus a special case of projection pursuit, where the projection index in this case is the variance along the projected direction. Principal components have two merits as a data reduction technique. Firstly, it sequentially extracts most of the variance of the data in the X space, so we might hope that only the first few components (far fewer than the full number p of original X variables) contain most of the information in the data. Secondly, by virtue of the way in which the components are extracted they are orthogonal, so that interpretation is eased. However, one should be aware that the principal component vectors in the **X** space may not necessarily be the ideal projection directions for optimizing predictive performance on a different variable Y (for example). For example, when we try to model differences among groups (or classes) in the data (for classification and clustering), the principal component projections need not emphasize group differences and indeed can even hide them. (Similar remarks can be made about more general projection pursuit methods.) Nonetheless,

PCA is widely used in data analysis and can be a very useful dimension - reduction tool. There are a wide number of other techniques (each with different properties) available for dimension reduction, including factor analysis, projection pursuit, independent component analysis, and so forth.

6.6 Models for Structured Data

In many situations either the individuals, the variables, or both, possess some well - defined relationships that are known a priori. Examples include linear chains or sequences (where the measurements are ordered for example, protein sequences), time series (where the measurements are ordered in time, perhaps on a uniform time scale), and spatial or image data (where the measurements are defined on a spatial grid). Even more complex structure is possible. For example, in medicine one can have imaging data of the brain measured on a three-dimensional grid, with repeated measurements over time.

Such structured data is inherently different from the types of measurements we have discussed in most places in this chapter. Up to this point we have implicitly assumed that the n individual objects (the patients, the customers) in our data set are a random sample from an underlying population. Specifically, we have assumed that the measurement vectors $\mathbf{x}(i)$, $1 \leq i \leq n$, are conditionally independent of each other given a particular fitted model (that is, that the likelihood of the data can be expressed as the product of individual $p(\mathbf{x}(i))$. For example, if we have a Normal

NOTES

density model for the variable weight, then we are assuming that knowing the weight of one person tells us nothing about the weight of any other person in the data set. (We are, of course, here ignoring subtle dependencies that may exist such as having members of the same family appear sequentially in our data set, where such family members might be predisposed to having similar overweight or underweight tendencies.) Thus, although it may be an approximation, we have been working with this assumption on the basis that it is a useful assumption for many practical situations.

However, there are problems for which the dependence is explicit and needs to be modeled. For example, if we take measurements of a person's blood pressure every five minutes over a 24-hour period, then clearly there is very likely to be some significant dependence between the successive values. How should we model such dependence?

One approach is to reduce the multiple observations on each object to one or a few variables (that is, a fixed multivariate description **x**), using ideas about the expected relationships between them (we referred to this possibility above). This is sometimes called the feature extraction approach. For example, we might expect blood pressure to decrease over the 24 -hour period as a medication begins to take effect, so we might replace the 5 times 12 times 24 observations for each person by just two numbers showing a starting value and the decreasing slope of a linear trend. Or we might use the same principle and fit a curve in which the rate of decrease reduces over time. The numbers describing the curves for each subject (which are often called derived variables) can then be analyzed in the standard way.

Note that this general approach (of converting sequential measurements into a non - sequential vector representation) may be sufficient for a given data mining task, but in general there is a loss of information in this process, in that we lose the timing and order information present in the original measurements. For certain applications this sequential information may be critical. As an example, we may have a population of Web users, among who are a group who navigate from Web page A, to page B, to page C, repeatedly in that order, in a cyclic fashion. If we were to reduce this information to a histogram of which pages were visited (yielding a histogram with three roughly equal bins), we would lose the ability to discover the dynamic cyclic pattern underlying the data.

NOTES

Let us consider an example of a sequential data model, namely a first-order Markov model for T data points observed sequentially, $y_1, \ldots, y_T$. Note that for even moderately large values of T , a full joint density for $p(y_1, y_2, \ldots, y_T)$ will be a very complex object (for example, if Y takes m discrete values, it will require the specification of $O(m^T)$ numbers). Thus, in modeling data with structure, we can take direct advantage of the ideas presented in the last section on factorization; that is, the structure of the data will suggest a natural structuring for any models we will build. Thus, we return to our first- order Markov model, again defined as:

$$p(y_1, \ldots, y_T) = p_1(y_1) \prod_{t=2}^{T} p_t(y_t | y_{t-1})$$

We can simplify this model considerably if we make the assumption of stationary, namely that the probability functions in the model do not depend on the specific time t, that is, $p_t(y_t|y_{t-1}) = p(y_t|y_{t-1})$. Thus, the same conditional probability function is used in different parts of the sequence. This drastically cuts down on the number of parameters we need for the model. For example, if Y is m-ary, the non-stationary model would require $O(m^2T)$ parameters (a matrix of m × m conditional probabilities for each time point in the sequence), while the stationary model only requires $O(m^2)$ probabilities (one matrix of m × m conditional probabilities that is used throughout the sequence). The notion of stationary can be applied to much more general Markov models than the first-order model above, and indeed extends naturally to spatial data models as well (for which we would assume stationary in space, rather than in time). If we assume stationary, then we cannot account for changes in the statistical model as a function of time or space. However, stationary is advantageous from a parameterization standpoint, making it a very useful and practical assumption in model building we will assume it throughout our discussion unless specifically stated otherwise.

The Markov model in above equation has a simple generative interpretation). The firs t value in the sequence y_1 is chosen by drawing a y_1 value randomly according to some initial distribution $p(y_1)$. The value at time t = 2 is randomly chosen according to the conditional density function $p(y_2 | y_1)$, where the value y_1 is known and fixed. Once y_2 has been chosen in this manner, y_3 is now generated according to $p(y_3|y_2)$ where the value y_2 is now fixed, and so on until time T.

However, the Markov model assumption is rather strong. In words, it says that the influence of the past is completely summarized by the value of Y at time t-1. Specifically, Y_t does not have any "long-range" dependencies other than its immediate dependence on Y_{t-1}. Clearly there are many situations in which this model may not be accurate. For example, consider modeling the grammatical structure of English text, where Y takes values such as verb, adjective, noun, and so on. The first-order Markov assumption is inadequate here since (for example) deciding whether a verb is singular or plural will depend on the subject of the verb that in turn may be much further back in the sentence than just one word back.

For real-valued Y s, the Markov model is often specified as a conditional Normal distribution:

$$p(y_t|y_{t-1}) = \frac{1}{\sqrt{2\pi}\sigma} exp - \frac{1}{2}\left(\frac{y_t - g(y_{t-1})}{\sigma}\right)^2$$

where $g(y_{t-1})$ plays the role of the mean of the Normal (it is a deterministic function linking the past y_{t-1} to the present y_t) and s is the noise in the model (assumed stationary here). A common choice for the function g is to make it a linear function of y_{t-1}, $g(y_{t-1}) = \alpha_0 + \alpha_1 y_{t-1}$, leading to the well-known first-order autoregressive model,

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + e$$

where e is zero -mean Gaussian noise with standard deviation s and the as are the parameters of the model. Note that the above equation can be expressed in the form of equation $p(y_t|y_{t-1})$ under these assumptions.

The model in equation y_t has a simple interpretation from a generative viewpoint; the value y_t at time t in the sequence is gene rated by taking the previous value y_{t-1}, multiplying it by a constant a_1, adding an offset a_0, and adding some amount of random noise e. For y to remain stable (bounded as $t \to \infty$) it is necessary that $-1 < \alpha_1 < 1$. Values of $|\alpha_1|$ closer to 1 imply stronger dependence among successive y values; values of $|\alpha_1|$ closer to 0 imply weaker dependence. Instead of regressing on independent X values, here Y is regressed on "lagged" values of itself. Thus, from our knowledge of regression model structures, we can immediately think of a multitude of generalizations of the simple first-order model above. For example, y_t can depend on earlier lags in the sequence; that is, we can replace the mean at time t by $g(y_{t-1})$ with $g(y_{t-1}, y_{t-2}, \ldots, y_{t-k})$, known as a kth order Markov model. Again, a common choice for $g(y_{t-1}, y_{t-2}, \ldots, y_{t-k})$ is a simple linear model of the form $a_0 + S\ a_i y_i$. In principle, however, rather than just linear regression, such as additive models, polynomial models, local linear models, data-driven local models, and so forth.

A further important generalization of the Markov model structures we have discussed so far is to explicitly model the notion of a hidden state variable. The general notion of hidden state for sequential and spatial models is prevalent in engineering and the sciences and recurs in a variety of functional model forms. Specific examples of such structures include hidden Markov models (HMMs) and Kalman filters. The HMM structure is easily explained by looking at its corresponding graphical model structure, shown in figure 6.7. From a generative viewpoint a first-order HMM operates as follows (picture the observations being generated by moving from left to right along the chain). The hidden state variable X is categorical (corresponding to m discrete states) and is first-order Markov. Thus, x_t is generated by sampling a value from the conditional distribution function $p(x_t|x_{t-1})$ in the usual Markov chain fashion, where $p(x_t|x_{t-1})$ is an m × m matrix of conditional probabilities. Once the state at time t is generated (with value x_t), an observation y_t is now generated with probability $p(y_t|x_t)$. Here y_t could be univariate or multivariate, or real-valued or categorical, or a combination of both. Thus, in a HMM, the observations y_t only depend on the state at time t, and the state sequence is a first-order Markov chain. The state sequence is unobserved or hidden, and the ys are directly observed: thus, there is uncertainty (given a model structure and a set of observed y's) about which particular state sequence generated the data.

NOTES

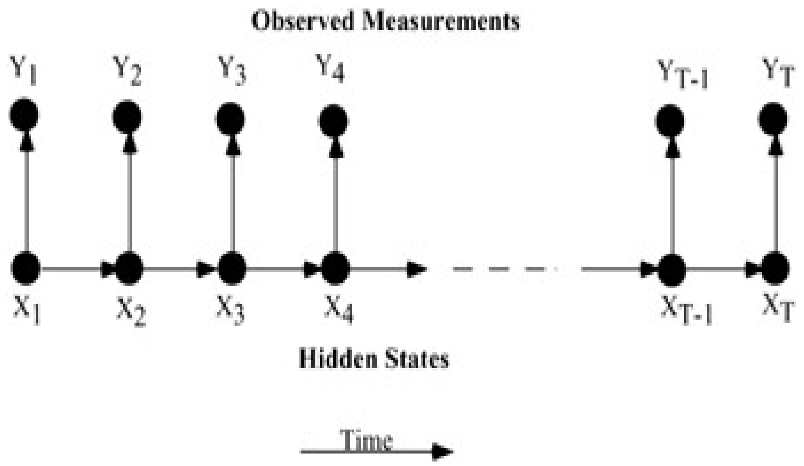


Figure 6.7: A graphical model structure corresponding to a first-order hidden markov assumption.

We can think of the HMM structure as a form of mixture model (m different density functions for the Y variable), where we have now added Markov dependence between "adjacent" mixture components x_t and x_{t+1}. For the record, the joint probability of an observed sequence and any particular hidden state sequence for a first-order HMM can be written as:

$$p(y_1, \ldots, y_T, x_1, \ldots, x_T) = p(x_1)p(y_1|x_1)\prod_{t=2}^{T} p(y_t|x_t)p(x_t|x_{t-1}).$$

The factorization on the right -hand side is apparent from the graphical model structure in figure 6.7. When regarded as a function of the parameters of the distributions, this is the likelihood of the variables ($Y_1, \ldots, Y_T, X_1, \ldots, X_T$). The likelihood of the observed ys is useful for fitting such model structures to data (that is, learning the parameters of $p(y_t|x_t)$ and $p(x_t|x_{t-1})$). To calculate $p(y_1, \ldots, y_T)$ (the likelihood of the observed data) one has to sum the left -hand side terms over the m^T possible state sequences, that appears at first glance to involve a sum over an exponential number of terms. Fortunately there is a convenient recursive way to perform this calculation in time proportional to $O(m^2T)$.

Again, it is clear that we can generalize the first-order HMM structure in different directions. A kth order Markov model corresponds to having x_t depend on the previous k states. The dependence of the ys can also be generalized, allowing for example y_t to have a linear dependence on the k previous ys (as in an auto regressive model) as well as direct dependence on x_t. This yields a natural generalization of the usual autoregressive model structure to a mixture of autoregressive models, which we can think of generatively as switching (in Markov fashion) among m different autoregressive models. Kalman filters are a closely related cousin of the HMM, where now the hiddenstates are real-valued (such as the unknown velocity or momentum of a vehicle, for example), but the independence structure of the model is essentially the same as we have described it for an HMM.

Computer scientists will recognize in our generative description of a hidden Markov model that it is quite reminiscent of a finite state machine (FSM). In fact, as we have described it here, a first-order HMM is directly equivalent to a stochastic FSM with m states; that is, the choice of the next state is governed by $p(x_{t+1}|x_t)$. This naturally suggests a generalization of model structures in terms of different grammars. Finite-state machines are simple forms of grammar known as regular grammars. The next level up (in the so called Chomsky hierarchy of grammars) is the context-free grammar, which can be thought of as augmenting the finite-state machine with a stack , permitting the model structure to "remember" long-range dependencies such as closing parentheses at the ends of clauses, and so forth. As we ascend the grammar hierarchy, our model structures become more expressive, but also become much more difficult to fit to data. Thus, despite the fact that regular grammars (or HMMs) are relatively simple in structure, this form of model structure has dominated the application of Markov models to sequential data (over other more complex grammar structures), due to the difficulties of fitting such complex structures to real data.

Finally, although we have only described simple data structures where the Y s exist in an ordered sequence, it is clear that for more general data dependencies (such as data on a two-dimensional grid) we can think of equivalent generalizations of the Markov model structures to model such dependence. For example, Markov random fields are essentially the multidimensional analogs of Markov chains (for example, in two dimensions we would have a grid structure rather than a chain for our graphical model).

It turns out that such models are much more difficult to analyze and work with than chain models. For example, problems such as summing out the hidden variables in the likelihood do not typically admit tractable solutions and must be approximated. Thus, spatial data can be more difficult to work with than sequential data, although conceptually the ideas of stationary, Markovianity, linear models, and so forth, can all still be applied. One common approach with gridded data, which may or may not make sense depending on the application, is to "shape" the two- dimensional grid data (say $n \times n$ grid points) into a single vector of length n^2, perform PCA on these vectors, project each set of grid measurements onto a small set of PCA vectors, and model the data using standard multivariate models in this reduced dimensional space. This approach ignores much of the inherent spatial information in the original grid, but nonetheless can be quite practical in many situations. Similarly, for multivariate time series or sequences , where we have p different time series or sequences measured over the same time frame (corresponding for example to different biomedical monitors on the same patient), we can use PCA to reduce the p original time series to a much smaller number of "component" series for further analysis.

6.7 Pattern Structures

Throughout this book, we have characterized a model as describing the whole (or a large part of the) data set, and a pattern as characterizing some local aspect of the data. A pattern can be considered to be a predicate that returns true for those objects or parts of objects in the data for which the pattern occurs, and false otherwise. To define a class of patterns we need to specify two things: the syntax of the patterns (the language specifying how they are defined) and their semantics (our interpretation of what they tell us about data to which they are applied). In this section we consider patterns for two different types of discrete-valued data: data in standard matrix form and data described as strings.

6.7.1 Patterns in Data Matrices

A generic approach for building patterns is to start from primitive patterns and combine them using logical connectives. (An alternative is to build a special class of patterns for a particular application.) Returning again to our data matrix notation, assume we have p variables $X_1, \ldots, X_p$. Let $\mathbf{x} = (x_1, \ldots, x_p)$ be a p-dimensional vector of measurements of these variables. We denote the ith individual in the data set as $\mathbf{x}(i)$, where $1 \leq I \leq n$. The entire data set $D = \{\mathbf{x}(1), \ldots, \mathbf{x}(n)$. In turn, $x_k(i)$ is the value of the kth measurement on the ith individual.

In general, a pattern for the variables $X_1, \ldots, X_p$ identifies a subset of all possible observations over these variables. A general

NOTES

language for expressing patterns can be built by starting from primitive patterns . These are simply conditions on the values of the variables. For example, if c is a possible value of X_k, then X_k = c is a primitive pattern. If the values of X_k are ordered (for example, numbers on the real line), we can also include inequalities such as $X_k \leq c$ as primitive conditions. If needed, the primitive patterns could also include multivariate conditions such as $X_k X_j > 2$ for numeric data or $X_k = X_j$ for discrete data.

Given a set of primitive patterns, we can form more complex patterns by using logical connectives such as AND (Λ) and OR (V). For example, we can form a pattern

(age = 40) Λ (income = 10)

that describes a certain subset of the input records in a payroll database. Note, for example, that each branch of a classification tree forms a conjunctive pattern of this form. Another example is the pattern

(chips = 1) Λ (beer = 1 V soft drink = 1)

describing a subset of rows in a market basket database.

A pattern class is a set of legal patterns. A pattern class C is defined by specifying the collection of primitive patterns and the legal ways of combining primitive patterns. For example, if the variables $X_1, \ldots, X_p$ all range over {0,1}, we can define a class of patterns C consisting of all possible conjunctions of the form

$$(X_{j_1} = 1) \wedge (X_{j_2} = 1) \wedge \cdots \wedge (X_{j_k} = 1).$$

Patterns in this class that occur frequently in a data set D are called frequent sets (of variables), since each such pattern is uniquely determined by a sub -set of the variables: this pattern could be written just as $X_{i_1}, X_{i_2}, \ldots, X_{i_k}$. Conjunctive patterns such as frequent sets are relatively easy to discover from data.

Given a pattern class and a dataset D, one of the important properties of a pattern is its frequency in the data set. The frequency fr(ρ) of a pattern ρ can be defined as the relative number of observations in the dataset about which ρ is true. In some cases, only patterns that occur reasonably often are of interest in data mining. However, having a frequency of a pattern close to 0 can also be quite informative in its own right. (Indeed, sometimes it is the rare but unusual pattern that is of particular interest.) Of course, the frequency of a pattern is not the only important property of the pattern. Properties such as semantic simplicity, understandability, and the novelty or surprise of the pattern are obviously also of interest. As an example, for any particular observation ($x_1, \ldots, x_p$) in the data set we can write a conjunctive pattern $(X_1 = x_1) \wedge \ldots \wedge (X_p = x_p)$ that matches exactly that observation. The disjunction of all of such conjunctive patterns forms a pattern that has frequency 1 for the data set. However, the pattern would be just a bloated way of writing out the entire data set and would be quite uninteresting.

Given a class of patterns, a pattern discovery task is to find all patterns from that class that satisfy certain conditions with respect to the data sets. For example, we might be interested in finding all the frequent set patterns whose frequency is at least 0.1 and where the variable X_7 occurs in the pattern. More generally, the definition of the pattern discovery task might include also conditions on the informativeness, novelty, and understandability of the pattern. In defining the pattern class and the pattern discovery task the challenge is to find the right balance between expressivity of the patterns, their comprehensibility, and the computational complexity of solving the discovery task.

Given a class of patterns C, we can easily define rules. A rule is simply an expression $\rho \Rightarrow \varphi$, where ρ and φ are patterns from a pattern class C. The semantics of a logical rule are that if the expression ρ is true for an object, then φ is also true. We can relax this definition to allow for uncertainty in the mapping from ρ to φ, where φ is true with some probability if ρ is true. The accuracy of such a rule is defined as $p(\varphi \mid \rho)$, the conditional probability that φ is true for an object, given that ρ is true. We can easily estimate such probabilities from a data set using appropriate frequency counts; that is

$$\hat{p}(\varphi|\rho) = \frac{fr(\rho \wedge \varphi)}{fr(\rho)}.$$

The support $fr(\rho \Rightarrow \varphi)$ of the rule $\rho \Rightarrow \varphi$ can be defined either as $fr(\rho)$ (the fraction of objects to which the rule applies) or $fr(\rho \wedge \varphi)$ (the fraction of objects for which both the left and right-hand sides of the rule are true).

6.7.2 Patterns for Strings

In the last section we discussed examples of patterns for data in the traditional matrix form. Other types of data require other types of patterns. To illustrate, we consider patterns for strings. Formally, a string over alphabet S is a sequence $a_1 \ldots a_n$ of elements (also called letters) of S. The alphabet S can be the binary alphabet {0,1}, the set of all ASCII codes, the DNA alphabet {A,C,G,T}, or the set of all words consisting of ASCII characters. The set of all strings built from letters from S is denoted by S*.

Note how string data differs from data in standard matrix form: for a string, there is no fixed set of variables. If and when we want to use the notions of probability to describe string data, we typically consider each of the letters of the string to be a random variable. The data can be one or several strings, and in most cases we are interested in finding

out how many times a certain pattern occurs in the strings. (For example, we might want

to compute the number of exact occurrences of a certain DNA sequenc e in a large collection of sequences.) The simplest string pattern is a substring: the pattern $b_1 \ldots b_k$ occurs in the string $a_1 \ldots a_n$ at position i, if $a_{i+j-1} = b_j$ for all j = 1, . . . , k . For example, for DNA sequences we might be interested in fin ding occurrences of the substring pattern ATTATTAA, and for strings over the ASCII alphabet we might be interested in whether or not the pattern data mining occurs in a given string.

For strings we might, however, be interested in a larger class of patterns. A regular expression E is an expression that defines a set L(E) of strings. The expression E is one of

1. a string s; then L(s) = {s}

2. a concatenation E_1E_2; in this case the set $L(E_1E_2)$ consists of all strings that are a concatenation of a string in $L(E_1)$ and a string in $L(E_2)$

3. a choice $E_1 \mid E_2$; then $L(E_1 \mid E_2) = L(E_1) \cup L(E_2)$

4. an iteration E*; then L(E*) consists of all strings that can be

written as a concatenation of 0 or more strings from L(E)

Thus, 10(00|11)*01 is a regular expression that describes all strings that start with 10 and end with 01 and in between contain a sequence of pairs 00 and 11.

Regular expressions are a form of patterns that are quite well suited to describing interesting classes of strings. While there are simple classes of strings that cannot be described by regular expressions (such as the set of strings consisting of all balanced sequences of parentheses), many quite complicated phenomena of strings can still be captured by using them.

While regular expressions are fine for defining patterns over strings, they are not sufficiently expressive for expressing variations in the occurrence times of events. A simple class of patterns that can take the occurrence times into account is the episode. At a high level, an episode is a partially ordered collection of events occurring together. The events may be of different types, and may refer to different variables. For example, in biostatistical data an event might be a headache followed by a sense of disorientation occurring within a given time period. It is also useful for them to be insensitive to intervening events as with, for example, alarms in a telecommunications network, logs of user interface actions, and so on. Episodes can also be incorporated into the type of rules discussed earlier.

Summary

This unit examined what we mean in a general sense by a data mining algorithm as what components make up such algorithms. A working definition is as follows: A data mining algorithm is a well-defined procedure that takes data as input and produces output in the form of models or patterns. We use the term well-defined indicate that the procedure can be precisely encoded as a finite set of rules. To be considered an algorithm, the procedure must always terminate after some finite number of steps and produce an output.

We have introduced the distinction between models and patterns in more depth, and examine some of the major classes of models and patterns used in data mining, in preparation for a detailed examination. A model is a high-level, global description of a data set. It takes a large sample perspective. It may be descriptive

NOTES

summarizing the data in a convenient and concise way or it may be inferential, allowing one to make some statement about the population from which the data were drawn or about likely future data values. We have discussed a variety of basic model forms such as linear regression models, mixture models, and Markov models.

Reference

1. Hand D, Mannila H. Smith P: Principles of Data Mining (PHI).

2. Pujari A: Data Mining Techniques, University Press (orient Longman.

UNIT – III

7. Score Functions for Data Mining Algorithms

Structure

Objective

After going through this lesson, you should be able to:

- Discuss about a Scoring Patterns;
- Discuss predictive versus descriptive score functions;
- Discuss about a scoring models with different complexities;
- Discuss about a evaluation of models, patterns, and robust methods

NOTES

7.1 Introduction

In lesson 6 we focused on different representations and structures that are available to us for fitting models and patterns to data. Now we are ready to consider how we will match these structures to data. Recall that a model or pattern structure is the functional form, with the parameters left "floating." For example, Y = aX + b might be one such model structure, with a and b the parameters. Given a model or pattern structure, we must score different settings of the parameter values with respect to data, so that we can choose a good set (or even "the best"). We saw how a least squares principle could be used to choose between different parameter values. This involved finding the values of the parameters a and b that minimized the sum of squared differences between the predicted values of y (from the model) and the observe d (data) values of y. In this case, the score function is thus the sum of squared errors between model predictions and actual target measurements. Our goal in this chapter is to broaden the reader's horizon in terms of the score functions that can be used for data mining. We will see that the venerable squared error score function is but one of many, and indeed can be viewed as a special case arising from more general principles.

It is important to bear in mind why we are interested in score functions in the first place. Ultimately the purpose of a score function should be to rank models as a function of how useful the models are to the data miner. Unfortunately in practice it can be quite difficult to measure "utility" in terms of direct practical usefulness to the person building the model. For example, in predicting stock market returns one might use squared error between predictions and actual data as a score function to train one's model. However, if the model is then used in a real financial environment, a host of other factors such as trading costs, risks, diversity, and so forth, comes into play to determine the true utility of the model. This illustrates that we often settle for simpler "generic" score functions (such as squared error) that have many desirable well-understood properties and are relatively easy to work with. Of course, one should not take this to an extreme: the score function being used should reflect the overall goals of the data mining task as far as are possible. One should try to avoid the situation, unfortunately all too common in practice, of using a convenient score function (perhaps because it is the default score function in the software package being used) that is completely inappropriate for the task.

Different score functions have different properties, and are useful in different situations. One of the goals of this chapter is to make the reader aware of these differences and of the implications of using one score function rather than another. Just as there are a few fundamental principles underlying model and pattern structures, so there are some basic principles underlying the different score functions. These are outlined in this chapter.

It is useful to make three distinctions at the outset. The first is between score functions for models, and score functions for patterns. The second is between score functions for predictive structures, and score functions for descriptive structures. And the third is between score functions for models of fixed complexity, and score functions for models of different complexity. These distinctions will be illustrated below.

A minor comment on the terminology used below is in order. In some places we will refer to score functions (such as error) that we clearly wish to minimize, whereas in other places we will refer to score functions (such as log-likelihood) that we clearly wish to maximize. The general concept is the same in either case, since the negative (or "inverse") of an "error-based" score function can always be maximized, and vice versa.

7.2 Scoring Patterns

Since the whole idea of searching for local patterns in data is relatively recent, there is a far smaller toolbox of techniques available for scoring patterns compared to the plethora of techniques for scoring models. Indeed, there is really no general consensus on how patterns should be scored. This is largely a result of the fact that the usefulness of a pattern lies in the eye of the beholder. One person's noisy outlier may be another person's Nobel Prize. Fundamentally, patterns might be evaluated in terms of how interesting or unexpected they are to the data analyst. But we could only hope to quantify this if some-how we had a precise model of what the user actually knows already. We are all familiar with the experience that the first time we learn something surprising is a lot more informative than the fifth or tenth time we hear the same information again. Thus, the degree to which a pattern is interesting to a person must be a function of their prior knowledge.

In practice, however, we cannot hope (except in simple situations) to be able to model a person's prior knowledge. Faced with a data

NOTES

set, a scientist or a marketing expert would have difficulty in precisely formulating what it is that they already know about the problem. Even subjective Bayesians can have problems choosing priors for complex multiparameter models and evade them by choosing standard forms for the priors that are only very simplistic representations of prior knowledge. We have found that, once certain patterns begin to emerge from the data (via visualization, descriptive statistics, or rules found by a data mining algorithm), database owners often say "Ah yes, but of course we knew that already," changing their minds about what they claim to have expected once they have seen the data.

Having said all of this, the fact remains that most techniques currently used in data mining for scoring patterns essentially assume that they are measuring degree of informativeness relative to a completely uninformed prior model; that is, it is effectively assumed that the data analyst has no prior information at all about the problem, beyond perhaps a few simple marginal and descriptive statistics. The hope is that this will eliminate the very obvious patterns (by focusing attention on patterns that are different from the known simple ones) and that the user can effectively "post-prune" the remaining patterns found by the algorithm to retain the truly interesting ones. The danger, of course, is that for some data sets and some forms of pattern searches, almost all patterns that are found by the data mining algorithm will essentially be uninteresting to the data analyst.

To illustrate these ideas we choose one simple (but widely used) pattern structure, the probabilistic rule. This has the form IF a THEN b with probability p

Where a and b are Boolean propositions (events) defined on a subset of the variables of interest and $p = p(b|a)$. How can we measure how interesting or informative this rule is to an uninformed observer? One simple approach is to assume that the observer already knows the marginal (unconditional) probability for the event b, p(b).

For example, suppose that we are studying a population of data miners. Let b represent the event that a randomly chosen person in this population is a data mining researcher, and let a be the event that such a person has read this book. Suppose we find that $p(b) = 0.25$ and that $p(b|a) = 0.75$; that is, 25% of this population are researchers and 75% of people who have read this book are researchers. This is interesting because it tells us that the proportion of people who undertake research is higher among

NOTES

those who have read the book than it is in this population of data miners in general (and hence, by implication, it is higher than among the people who have not read the book). Note, as an aside, that there are no causal implications to this. It could be that the book inspired a reader to take up research, or that a person involved in research hoped the book would help them.

The types of simple score functions that are used to capture the informativeness of such a rule rely in general on how "far" the posterior probability p(b|a) is (after learning that event a is true), from the prior probability p(b). Thus, for example, one could simply measure absolute distance between the probabilities |p(b|a) - p(b)|, or perhaps measure distance on log-odds scale, log $\frac{\log(p(b|a))}{p(\bar{b}|a)}$ where $\bar{b}$ represents the event that a person is not a researcher.

When we compare different patterns, such as p(b|a) and p(b|c), it is also useful to take into account the coverage of a pattern that is, the proportion of the data to which it applies. To continue our example above, let c be the condition that the randomly chosen data miner is one of the three authors of this book. A second pattern might be "if c then b" ("if a data miner is an author of this book then they are a researcher"), with p(b|c) = 1 since the three authors are all researchers. However, the condition c only applies to three data miners, which is a very small fraction of the universe of data miners. On the other hand, (we hope that) the coverage of event a will be much larger; that is, p(a) is significantly greater than p(c). To illustrate, suppose that p(a) = 0.2 and p(c) = 0.003. Then, although the second pattern is very accurate (p(b|c) = 1) it is not particularly useful since it only applies to a very small fraction of the population (0.3%), whereas the first pattern is not as accurate (p(b|a) = 0.75) but it has much broader applicability (to 20% of the population). It is easy to develop a variety of measures that augment the score function to take coverage into account. For example, we could multiply the previously defined scores by the probability of the conditioning event; p(a)|p(b|a) - p(b)| = |p(b, a) - p(b)p(a)| that can be interpreted as measuring the difference in probability between an independence assumption and the observed joint probability for the two events a and b. Alternatively, the approach used in association rule mining defines a threshold p_t, and only seeks patterns with coverage greater than p_t.

There are numerous other score functions for patterns that have been proposed in the data mining literature. None have gained widespread acceptance or general use, largely because judging the novelty and utility of a pattern is often quite subjective and application-specific. Thus, human interpretation by a domain expert remains the most practical way to evaluate patterns at present (e.g., having a human search through and interpret a set of candidate patterns produced by a data mining algorithm).

7.3 Predictive versus Descriptive Score Functions

We now turn to score functions for models, where there is a much greater selection of useful methods available compared to patterns.

7.3.1 Score Functions for Predictive Models

A convenient place to begin is by considering the distinction between prediction and description. Score functions for predictive problems are relatively straightforward. In a prediction task, our training data comes with a "target" value Y, this being a quantitative variable for regression or a categorical variable for classification, and our data set D = {(**x**(1), y(1)), ... ,(**x**(n), y(n))} consists of pairs of input vectors and target values. Let f(**x**(i); ө) be the prediction generated by the model for individual i, 1 ≤ i ≤ n, using parameter values ө. Let y(i) be the actual observed value (or "target") for the ith individual in the training data set.

Clearly our score function should be a function of the difference between the predictions f(**x**(i); ө) and the targets y(i). Commonly used score functions include the sum of squared errors,

$$S_{SSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{f}(\mathbf{x}(i);\theta) - y(i)\right)^2$$

for quantitative Y, and the misclassification rate (or error rate or "zero -one" score function) for categorical Y, namely,

$$S_{0/1}(\theta) = \frac{1}{N}\sum_{i=1}^{N} I\left(\hat{f}(\mathbf{x}(i);\theta), y(i)\right)$$

where I(a, b) = 1 if a is not equal to b and 0 otherwise. These are the two most widely- used score functions for regression and classification respectively. They are simple to understand and (in the case of squared error at least) often lead to straightforward optimization problems.

NOTES

However, note that we have made some strong assumptions in how these score functions are defined above. For example, by summing over the individual errors we are assuming that errors for all individuals may be treated equally. This is a very common assumption and generally useful. However, if (for example) we have a data set in which the measurements were taken at different times, we might want to assign higher weight in the score function to predictions on more recent items. Similarly, we might have different subsets of items in the data set where the target values are more reliable in some subsets than others (for example, some quantification of measurement error in a subset). Here we might wish to assign lower weight in the score function to predictions on the items with less reliable measurements.

Furthermore, both are functions only of the difference between the predictions and targets in particular, they do not depend on the values of the target y(i). This is something we might want to take account of. For example, if Y were a categorical variable indicating whether or not a person had cancer, we might wish to give more weight to the error of not detecting a true cancer and less weight to errors that correspond to false alarms. For real-valued Y, squared-error may not be appropriate perhaps the quality of the model is more appropriately reflected in absolute error (squared-error gives greater weight to extreme differences between the observed and predicted Y values than doe's absolute error). And, as a third example, in an investment scenario, we might want be more tolerant (from a risk-taking standpoint) of predictions of Y that underestimate the true value than we are to predictions that overestimate, suggesting that an asymmetric function might be more appropriate.

7.3.2 Score Functions for Descriptive Models

For descriptive models, in which there is no "target" variable to be predicted, it is less clear how to define a score function. A fundamental approach is through the likelihood function, but which we here describe from a slightly different perspective. Let β(x;ɵ) be the estimated probability of observing a data point at **x**, as defined by our model $\hat{\mathbf{p}}$ with parameters ɵ, where X is categorical (the extension to continuous variables is straightforward, and p would then be a probability density function). If the model is a good one, then it might be expected to place a high probability at those values of X where a data point is observed. Thus $\hat{\mathbf{p}}(\mathbf{x})$ itself can be taken as a measure of quality of the model a score function at the point **x**. This is the basic idea of maximum likelihood once again:

NOTES

better models assign higher probability to observed data. (This is fine actually as long as we can assume that all the models we are considering have equal functional complexity, so that the comparison is "fair" t he case in which we are comparing models of different complexities.)

If we assume that the data points have arisen independently, we can define an overall score function for the model by combining these score functions for the individual data points simply by multiplying them together:

$$L(\theta) = \prod_{i=1}^{n} \hat{p}(\mathbf{x}(i); \theta).$$

This is a set of data points that we maximize to find an estimate of ɵ. As we noted there, it is typically more convenient to work with the log –likelihood. Now the contribution of an individual data point to the overall score function is $\log \hat{p}(\mathbf{x}(i); \theta)$, and the overall function is the sum of these:

$$\log L(\theta) = \sum_{i=1}^{n} \log \hat{p}(\mathbf{x}(i); \theta).$$

If we work with the negative of the $\log \hat{p}(\mathbf{x}(i); \theta)$, as is often done, then this function needs to be minimized. We define

$$S_L(\theta) = -\log L(\theta) = -\sum_{i=1}^{n} \log \hat{p}(\mathbf{x}(i); \theta).$$

Note again the intuitive interpretation: $-\log \hat{p}$ is our error term (it gets larger as $\hat{p}$ gets smaller), and we are summing this over all of our data points. The largest possible value for $\hat{p}$ is 1 (for categorical data) and, hence, S_L(ɵ) is lower bounded by 0.Thus, we can think of S_L(ɵ) as a type of entropy term that measures how well the parameters ɵ can compress (or predict) the training data.

A particularly useful feature of the likelihood (or, equivalently, the negative log-likelihood) is that it is very general. It can be defined for any problem in which the model or pattern being examined is expressed in terms of probability functions. For example, one might assume that Y in a predictive model is a perfect linear function of some predictor variable X, as well as extra randomly distributed errors, as discussed in the last section. If one can postulate a parametric form for the probability distribution of these errors, then one can compute the likelihood of the data for any

NOTES

proposed parameters in the model. If the error terms are supposed to be normally distributed with mean 0 about a deterministic function of X then the likelihood score function is equivalent to the sum of squared errors score function.

Although (negative log-) likelihood is a powerful and useful score function, it too has its limitations. In particular, if a parameterization assigns any data point a probability near 0, the log-likelihood will approach $-\infty$. Thus, the overall error can be dominated by extreme points. If the true probability of that same point is also very small, then the model is being penalized for a prediction in the tails of the density function (very unlikely events), that may have little relation to the practical utility of the model. Conversely, there may be problems (such as predicting the occurrence of rare events) in which it is precisely in the tails of the density that we are most interested in accurate prediction. Thus, while likelihood is based on strong theoretical foundations and is generally useful for scoring probabilistic models, it is import ant to realize that it may not necessarily reflect the true utility of a model for a particular task. Other score functions for determining the quality of probabilistic predictions are also possible, each with its own particular characteristics. For example we can define the integrated squared error between our estimate $\hat{p}(x;\theta)$ and the true probability p(x), $\int (\hat{p}(x;\theta) - p(x))^2 \, dx$. By completing the square, and ignoring terms not depending on θ, we get a score function of the form $\int \hat{p}(x;\theta)^2 \, dx - 2E[\hat{p}(x;\theta)]$, where each term can be empirically approximated to provide an estimate of the true integrated squared error as a function of θ.

For nonprobabilistic descriptive models, such as partition -based clustering, it is quite easy to come up with all sorts of score functions based on how well separated the clusters are, how compact they are, and so forth. For example, for simple prototype-based clustering, a simple and widely used score function is the sum of square errors within each cluster

$$S_{KSSE}(\theta) = \sum_{k=1}^{K} e_k, \quad e_k = \sum_{i \in cluster_k} ||\mathbf{x}(i) - \mu_k||^2$$

where θ is the parameter vector for the cluster model, $\theta = \{\mu_1, \ldots, \mu_K\}$, and the μ_ks are the cluster centers. However, it is quite difficult to formulate any score function for cluster models that reflect how close the clusters are to "truth" (if this is regarded as meaningful). The ultimate judgment on the utility of a given clustering depends on how useful the clustering is in the context of the particular application. Does it provide new insight into the data? Does it permit a meaningful categorization of the data? And so on. These are questions that typically can only be answered in the context of a particular problem and cannot be captured by a single score metric. To put it another way, once again the score functions for tasks such as clustering are not necessarily very closely related to the true utility function for the problem.

To summarize, there are simple "generic" score functions for tasks such as classification, regression, and density estimation, that are all useful in their own right. However, they do have limitations, and it is perhaps best to regard them as starting points from which to generalize to more application-specific score functions.

7.4 Scoring Models with Different Complexities

In the preceding sections we described score functions as minimizing some measure of discrepancy between the observed data and the proposed model. One might expect models that are close to the data (in the sense embodied in the score function) to be "good" models. However, we need to be clear about why we are building the model.

7.4.1 General Concepts in Comparing Models

We can distinguish between two types of situations (as we have in earlier chapters). In one type of situation we are merely trying to build a summarizing descriptive model of a data set that captures its main features. Thus, for example, we might want to summarize the main chemical compounds among the members of a particular family of compounds, where our database contains records for all possible members of this family. In this case, accuracy of the model is paramount though it will be mediated by considerations of comprehensibility. The best accuracy is given by a model that exactly reproduces the data, or describes the data in some equivalent form, but the whole point of the modeling exercise in this case is to reduce the complexity of the data to something that is more comprehensible. In situations like this, simple goodness of fit of the model to the data will be one part of an overall score measure, with comprehensibility being another part (and this part will be subjective). An example of a general technique in this context is based on data compression and information -theoretic arguments, where our score function is generally decomposed as

NOTES

$S_l(\theta, M)$ = number of bits to describe the data given the model + number of bits to describe the model (and parameters) where the first term measures the goodness of fit to the data and the second measures the complexity of the model M and its parameters θ. In fact, for the first term ("number of bits to describe the data given the model") we can use $S_L = -\log p(D \mid \theta, M)$ (negative log-likelihood, log base 2). For the second term ("number of bits to describe the model") we can use $-\log p(\theta, M)$. Intuitively, we can think of $-\log p(\theta, M)$ (the second term) as the communication "cost" in bits to transmit the model structure and its parameters from some hypothetical transmitter to a hypothetical receiver, and S_L (the first term) as the cost of transmitting the portion of the data (the errors) that the model and its parameters do not account for. These two parts will tend to work in opposite directions a good fit to the data will be achieved by a complicated model, while comprehensibility will be achieved by a simple model. The overall score function trades off what is meant by an acceptable model.

7.4.2 Bias-Variance Again

Before examining score functions that we might hope will provide a good fit to data as yet unseen, it will be useful to look in more detail at the need to avoid modeling the available data too closely. We discussed bias and variance in the context of estimates of parameters θ and we discuss it again here in the more general context of score functions.

As we have mentioned in earlier chapters, it is extremely unlikely that one's chosen model structure will be "correct." There are too many features of the real world for us to be able to model them exactly (and there are also deep questions about just what "correct" means). This implies that the chosen model form will provide only an approximation to the "truth." Let us take a predictive model to illustrate. Then, at any given value of X (which we take to be univariate for simplicity exactly the same argument holds for multivariate X), the model is likely to provide predicted values of Y that are not exactly right. More formally, suppose we draw many different data sets, fit a model of the specified structure (for example, a piecewise local model with given number of components, each of given complexity; a polynomial function of X of given degree;

and so on) to each of them, and determine the expected value of the predicted Y at any X. Then this expected predicted value is unlikely to coincide exactly with the true value. That is, the model

NOTES

is likely to provide a biased prediction of the true Y at any given X. (Recall that bias of an estimate was defined as the difference between the expected value of the estimate and the true value.) Thus, perfect prediction is too much to hope for!

However, we can make the difference between the expected value of the predictions and the unknown true value smaller (indeed, we can make it as small as we like for some classes of models and some situations) by increasing the complexity of the model structure.

At first glance, this looks great we can obtain a model that is as accurate as we like, in terms of bias, simply by taking a complicated enough model structure. Unfortunately, there is no such thing as a free lunch, and the increased accuracy in terms of bias is only gained at a loss in other terms.

By virtue of the very flexibility of the model structure, its predictions at any fixed X could vary dramatically between different data sets. That is, although the average of the predictions at any given X will be close to the true Y (this is what small bias means), there may be substantial variation between the predictions arising from the different data sets. Since, in practice, we will only ever observe one of these predictions (we really have only one data set to use to estimate the model's parameters) the fact that "on average" things are good will provide little comfort. For all we know we have picked a data set that yields predictions far from the average. There is no way of telling.

There is another way of looking at this. Our very flexible model (with, for example, a large number of piecewise components or a high degree) has led to one that closely follows the data. Since, at any given X, the observed value of Y will be randomly distributed about its mean, our flexible model is also modeling this random component of the observed Y value. That is, the flexible model is overfitting the data.

Finally (though, yet again, it is really just another way of looking at the same thing), increasing the complexity of the model structure means increasing the number of parameters to be estimated. Generally, if more parameters are being estimated, then the accuracy of each estimate will decrease (its variance, from data set to data set, will increase).

The complementarity of bias and variance in the above, is termed the bias -variance trade-off. We want to choose a model in which neither is too large but reducing either one tends to increase the other. They can be combined to yield an overall measure of discrepancy between the data and the model to yield the mean squared error (MSE). Consider the standard regression setting we have discussed before, where we are assuming that y is a deterministic function of **x** (where we now generalize to the vector case) with additive noise, that is, y = f(**x**; θ) + e, where e is (for example) Normal with zero mean. Thus, μ_y = E[y|**x**] represents the true (and unknown) expected value for any data point **x** (where here the expectation E is with respect to the noise e), and y = f(**x**; θ) is the estimate provided by our model and fitted parameters θ. The MSE at **x** is then defined as:

$$\begin{aligned} MSE(\mathbf{x}) &= E\left[\hat{y} - \mu_y\right]^2 \\ &= E\left[\hat{y} - E\left(\hat{y}\right)\right]^2 + E\left[E\left(\hat{y}\right) - \mu_y\right]^2 \end{aligned}$$

or MSE = Variance + Bias^2. (The expectation E here is taken with respect to p(D), the probability distribution over all possible data sets for some fixed size n). This equation bears close inspection. We are treating our prediction y here as a random quantity, where the randomness arises from the random sampling that generated the training data D. Different data sets D would lead to different models and parameters, and different predictions y. The expectation, E, is over different data sets of the same size n, each randomly chosen from the population in question. The variance term E [$\hat{y}$ - E ($\hat{y}$)]^2 tell us how much our estimate $\hat{y}$ will vary across different potential data sets of size n. In other words, it measures the sensitivity of $\hat{y}$ to the particular data set being used to train our model. As an extreme example, if we always picked a constant y_1 as our prediction, without regard to the data at all, then this variance would be zero. At the other extreme, if we have an extremely complex model with many parameters, our predictions y may vary greatly depending from one individual training data set to the next.

The bias term E [E($\hat{y}$) - μ_y] reflects the systematic error in our prediction that is how far away our average prediction is, E ($\hat{y}$), from truth μ_y. If we use a constant y_1 as our prediction, ignoring the data, we may have large bias (that is, this difference may be large). If we use a more complex model, our average prediction may get closer to the truth, but our variance may be quite large. The bias -variance quantifies the tension between simpler models (low variance, but potentially high bias) and more complex ones (potentially low bias but typically high variance).

NOTES

In practice, of course, we are interested in the average MSE over the entire domain of the function we are estimating, so we might define the expected MSE (with respect to the input distribution p(**x**)) as $\int MSE(\mathbf{x})p(\mathbf{x})\,d\mathbf{x}$, that again has the same additive decomposition (since expectation is linear).

Note that while we can in principle measure the variance of our predictions y (for example, by some form of resampling such as the bootstrap method), the bi as will always be unknown since it involves μ_y that is itself unknown (this is after all what we are trying to learn). Thus, the bias -variance decomposition is primarily of theoretical interest since we cannot measure the bias component explicitly, and in turn it does not provide a practical score function combining these two aspects of estimation error. Nonetheless, the practical implications in general are clear: we need to choose a model that is not too inflexible (because its predictions will then have substantial bias) but not too flexible (since then its predictions will have substantial variance). That is, we need a score function that can handle models of different complexities and take into account this compromise, and one that can be implemented in practice.

We should note that in certain data mining applications, the issue of variance may not be too important, particularly when the models are relatively simple compared to the amount of data being used to fit them. This is because variance is a function of sample size. Increasing the sample size decreases the variance of an estimator. Unfortunately, no general statements can be made about when variance and overfitting will be important issues. It depends on both the sample size of the training data D and the complexity of the model being fit.

7.4.3 Score Functions that Penalize Complexity

How, then, can we choose a suitable compromise between flexibility (so that a reasonable fit to the available data is obtained) and overfitting (in which the model fits chance components in the data)? One way is to choose a score function that encapsulates the compromise. That is, we choose an overall score function that is explicitly composed of two components: a component that measures the goodness of fit of the model to the data, and an extra component that puts a premium on simplicity. This yields an overall score function of the form

score(model) = error(model) + penalty-function(model), where we want to minimize this score. We have discussed several different ways to define the error component of the score in the preceding sections. What might the additional penalty component look like?

In general (though there are subtleties that mean that this is something of a simplification), the complexity of a model M will be related to the number of parameters, d, under consideration. We will adopt the following notation in this context. Consider that there are K different model structures, $M_1, \ldots, M_K$, from which we wish to choose one (ideally the one that predicts best on future data). Model M_k has d_k parameters. We will assume that for each model structure M_k, $1 \leq k \leq K$, the best fitting parameters $\hat{\theta}_k$ for that model (those that maximize goodness-of-fit to the data) have already been chosen; that is, we have already determined point estimates of these parameters for each of these K model structures and now we wish to choose among these fitted models. The widely used Akaike information criterion or AIC is defined as

$$S_{AIC}(M_k) = 2S_L(\hat{\theta}_k; M_k) + 2d_k, \quad 1 \leq k \leq K.$$

where S_L is the negative log-likelihood and the penalty term is $2d_k$. This can be derived formally using asymptotic arguments.

An alternative, based on Bayesian arguments, also takes into account the sample size, n. This Bayesian Information Criterion or BIC is defined as

$$S_{BIC}(M_k) = 2S_L(\hat{\theta}_k; M_k) + d_k \log n$$

where S_L is again the negative log -likelihood. Note the effect of the additive penalty term $d_k \log n$. For fixed n, the penalty term grows linearly in number of parameters d_k, which is quite intuitive. For a fixed number of parameters d_k, the penalty term increases in proportion to log n. Note that this logarithmic growth in n is offset by the potentially linear growth in S_L as a function of n (since it is a sum of n terms). Thus, asymptotically as n gets very large, for relatively small values of d_k, the error term S_L (linear in n) will dominate the penalty term (logarithmic in n). Intuitively, for very large numbers of data points n, we can "trust" the error on the training data and the penalty function term is less relevant. Conversely, for small numbers of data points n, the penalty function term $d_k \log n$ will play a more influential role in model selection.

There are many other penalized score functions with similar additive forms to those above (namely an error-based term plus a penalty term) include the adjusted R^2 and C_p scores for regression, the minimum description length (MDL) method and Vapnik's structural risk minimization approach (SRM).

Several of these penalty functions can be derived from fairly fundamental theoretical arguments. However, in practice these types of penalty functions are often used under far broader conditions than the assumptions used in the derivation of the theory justify. Nonetheless, since they are easy to compute they are often quite convenient in practice in terms of giving at least a general idea of what the appropriate complexity for a model is, given a particular data set and data mining task.

A different approach is provided by the Bayesian framework. We can try to compute the posterior probability of each model given the data directly, and select the one with the highest posterior probability; that is,

$$\begin{aligned} p(M_k|D) &\propto p(D|M_k)p(M_k) \\ &= \int p(D, \theta_k|M_k)p(M_k)d\theta_k \\ &= \int p(D|\theta_k)p(\theta_k|M_k)d\theta_k \; p(M_k) \end{aligned}$$

where the integral represents calculating the expectation of the likelihood of the data over parameter space (also known as marginal likelihood), relative to a prior in parameter space $p(\theta_k|M_k)$, and the term $p(M_k)$ is a prior probability for each model. This is clearly quite different from the "point estimate" methods the Bayesian philosophy is to fully acknowledge uncertainty and, thus, average over our parameters (since we are unsure of their exact values) rather than "picking" point estimates such as $\hat{\theta}_k$. Note that this Bayesian approach implicitly penalizes complexity, since higher dimensional parameter spaces (more complex models) will mean that the probability mass in $p(\theta_k|M_k)$ is spread more thinly than in simpler models.

Of course, in practice explicit integration is often intractable for many parameter spaces and models of interest and Monte Carlo sampling techniques are used. Furthermore, for large data sets, the $p(D|\theta_k)$ function may in fact be quite "peaked" about a single value $\hat{\theta}_k$, in which case we can reasonably approximate the Bayesian expression above by the value of the peak plus some estimate of the surrounding volume (for example, a Taylor series type of expansion around the posterior mode of $p(D|\theta)\,p(\theta)$) this type of argument can be shown to lead to approximations such as BIC above).

7.4.4 Score Functions using External Validation

A different strategy for choosing models is sometimes used, not based on adding a penalty term, but instead based on external validation of the model. The basic idea is to (randomly) split the data into two mutually exclusive parts, a design part D_d, and a validation part D_v. The design part is used to construct the models and estimate the parameters. Then the score function is recalculated using the validation part. These validation scores are used to select models (or patterns). An important point here is that our estimate of the score function for a particular model, say $S(M_k)$, is itself a random variable, where the randomness comes from both the data set being used to train (design) the model and the data set being used to validate it. For example, if our score is some error function between targets and model predictions (such as sum of squared errors), then ideally we would like to have an unbiased estimate of the value of this score function on future data, for each model under consideration. In the validation context, since the two data sets are independently and randomly selected, for a given model the validation score provides an unbiased estimate of the score value of that model for new ("out-of-sample") data points. That is, the bias in estimates, that inevitably arises with the design component, is absent from the independent validation estimate. It follows from this (and the linearity of expectation) that the difference between the scores of two models evaluated on a validation set will have an expected value in the direction favoring the better model. Thus, the difference in validation scores can be used to choose between models. Note that we have previously discussed unbiased estimates of parameters θ (unbiased estimates of what we are trying to predict μ_y (earlier in this chapter), and now unbiased estimates of our score function S. The same principles of bias and variance underly all three contexts, and indeed all three contexts are closely interlinked (accuracy in parameter estimates will affect accuracy of our predictions, for example) it is important, however, to understand the distinction between them.

This general idea of validation has been extended to the notion of cross-validation. The splitting into two independent sets is randomly repeated many times, each time estimating a new model (of the given form) from the design part of the data and obtaining an unbiased estimate the out -of-sample performance of each model from the validation component. These unbiased estimates are then averaged to yield an overall estimate. We described the use of cross-validation to choose between CART recursive partitioning models. Cross-validation is popular in practice, largely because it is simple and reasonably robust (in the sense that it relies on relatively few assumptions).However, if the partitioning is repeated m times it does come at a cost of (on the order of) m times the complexity of a method based on just using a single validation set. (There are exceptions in special cases. For example, there is an algorithm for the leave - one-out special case of cross-validation applied to linear discriminant analysis that has the same order of computational complexity as the basic model construction algorithm.)

For small data sets, the process of selecting validation subsets D_v can lead to significant variation across data sets, and thus, the variance of the cross-validation score also needs to be monitored in practice to check whether or not the variation may be unreasonably high. Finally, there is a subtlety in cross-validation scoring in that we are averaging over models that have potentially different parameters but the same complexity. It is important that we are actually averaging over essentially the same basic model each time. If, for example, the fitting procedure we are using can get trapped at different local maxima in parameter space, on different subsets of training data, it is not clear that it is meaningful to average over the validation scores for these models.

It is true, as stated above, that the estimate of performance obtained from such a process for a given model is unbiased. This is why such methods are very widely used and have been extensively developed for performance assessment. However, some care needs to be exercised. If the validation measures are subsequently used to choose between models (for example, to choose between models of different complexity), then the validation score of the model that is finally selected will be a biased estimate of this model's performance. To see this, imagine that, purely by chance some model did exceptionally well on a validation set. That is, by the accidental way the validation set happened to have fallen, this model did well. Then this model is likely to be chosen as the "best" model. But clearly, this model will not do so well with new out-of-sample data sets. What this means in practice is that, if an assessment of the likely future performance of a (predictive) model is needed, then this must be based on yet a

NOTES

third data set, the test set , about which we shall say more in the next subsection.

7.5 Evaluation of Models and Patterns

Once we have selected a model or pattern, based on its score function, we will often want to know (in a predictive context) how well this model or pattern will perform on new unseen data. For example, what error rate, on future unseen data, would we expect from a predictive classification model we have built using a given training set? We have already referred to this issue when discussing the validation set method of model selection above.

Again we note that if any of the same data that have been used for selecting a model or used for parameter estimation are then also used again for performance evaluation, then the evaluation will be optimistically biased. The model will have been chosen precisely because it does well on this particular data set. This means that the apparent or resubstitution performance, as the estimate based on reusing the training set is called, will tend to be optimistically biased.

If we are only considering a single model structure, and not using validation to select a model, then we can use subsampling techniques such as validation or cross-validation, splitting the data into training and test sets, to obtain an unbiased estimate of our model's future performance. Again this can be repeated multiple times, and the results averaged. At an extreme, the test set can consist of only one point, so that the process is repeated N times, with an average of the N single scores yielding the final estimate. This principle of leaving out part of the data, so that it can provide an independent test set, has been refined and developed to a great degree of technical depth and sophistication, notably in jackknife and bootstrap methods, as well as the leaving -one-out cross- validation method (all of these are different, though related and sometimes confused). The further reading section below gives pointers to publications containing more details.

The essence of the above is that, to obtain unbiased estimates of likely future performance of a model we must assess its performance using a data set which is independent of the data set used to construct and select the model. This also applies if validation data sets are used. Suppose, for example, we chose between K models by partitioning the data into two subsets, where we fit parameters on the first subset, and select the single "best" model using the model scores on the second (validation) subset. Then, since we will choose that model which does best on the validation data set, the model will be selected so that it fits the idiosyncrasies of this validation data set. In effect, the validation data set is being used as part of the design process and performance as measured on the validation data will be optimistic. This be comes more severe, the larger is the set of models from which the final model is chosen.

The message here is that if one uses a validation set to choose between models, one cannot also use it to provide an estimate of likely future performance. The very fact that one is choosing models which do well on the validation set means that performance estimates on this set are biased as estimates of performance on other unseen data. As we said above, the validation set, being used to choose between models, has really become part of the design process. This means that to obtain unbiased estimates of likely future performance we ideally need access to yet another data set (a "hold-out" set) that has not been used in any way in the estimation or model selection so far. For very large data sets this is usually not a problem, in that data is readily available, but for small data sets it can be problematic since it effectively reduces the data available for training.

7.6 Robust Methods

We have pointed out elsewhere that the notion of a "true" model is nowadays regarded as a weak one. Rather, it is assumed that all models are approximations to whatever is going on in nature, and our aim is to find a model that is close enough for the purpose to hand. In view of this, it would be reassuring if our model did not change too dramatically as the data on which it was based changed. Thus, if a slight shift in value of one data point led to radically different parameter estimates and predictions in a model, one might be wary of using it. Put another way, we would like our models and patterns to be insensitive to small changes in the data. Likewise, the score functions and models may be based on certain assumptions (for example, about underlying probability distributions). Again it would be reassuring if, if such assumptions were relaxed slightly, the fitted model and its parameters and predictions did not change dramatically.

NOTES

Score functions aimed at achieving these aims have been developed. For example, in a trimmed mean a small proportion of the most extreme data points are dropped, and the mean of the remainder used. Now the values of outlying points have no effect on the estimate. The extreme version of this (assuming a univariate distribution with equal numbers being dropped from each tail), arising as a higher and higher proportion is dropped from the tails, is the median which is well known to be less sensitive to changes in outlying points than is the arithmetic mean. As another example, the Winsorized mean involves reducing the most extreme points to have the same values as the next most extreme points, before computing the usual mean.

Although such modifications can be thought of as robust forms of score functions, it is sometimes easier to describe them (and, indeed think of them) in terms of the algorithms used to compute them.

8. Search and Optimization Methods

Structure

Objective

After going through this lesson, you should be able to:

- Discuss about a searching for models and patterns;
- Discuss parameter optimization methods;
- Discuss about a optimization with missing data: the EM algorithm;
- Discuss varies optimization methods

8.1 Introduction

This lesson focuses on the computational methods used for model and pattern-fitting in data mining algorithms; that is, it focuses on the procedures for searching and optimizing over parameters and structures guided by the available data and our score functions. The importance of effective search and optimization is often underestimated in the data mining, statistical and machine learning algorithm literatures, but successful applications in practice depend critically on such methods.

We recall that a score function is the function that numerically expresses our preference for one model or pattern over another. For example, if we are using the sum of squared errors, S_{SSE}, we will prefer models with lower S_{SSE} this measures the error of our model (at least on the training data). If our algorithm is searching over multiple models with different representational power (and different complexities), we may prefer to use a penalized score function such as S_{BIC} whereby more complex models are penalized by adding a penalty term related to the number of parameters in the model.

Regardless of the specific functional form of our score function S, once it has been chosen, our goal is to optimize it. (We will usually assume without loss of generality in this chapter that we wish to minimize the score function, rather than maximize it). So, let $S(\theta|D, M) = S(\theta_1, ..., \theta_d|D, M)$ be the score function. It is a scalar function of a d-dimensional parameter vector θ and a model structure M (or a pattern structure θ), conditioned on a specific set of observed data D.

This lesson examines the fundamental principles of how to go about finding the values of parameter(s) that minimize a general score function S. It is useful in practical terms, although there is no high -level conceptual difference, to distinguish between two situations, one referring to parameters that can only take discrete values (discrete parameters) and the other to parameters that can take values from a continuum (continuous parameters).

Examples of discrete parameters are those indexing different classes of models (so that 1 might correspond to trees, 2 to neural networks, 3 to polynomial functions, and so on) and parameters that can take only integral values (for example, the number of variables to be included in a model). The second example indicates the magnitude of the problems that can arise. We might

NOTES

want to use, as our model, a regression model based on a subset of variables chosen from possible p variables. There are $K = 2^p$ such subsets, which can be very large, even for moderate p. Similarly, in a pattern context, we might wish to examine patterns that are probabilistic rules involving some subset of p binary variables expressed as a conjunction on the left-hand side (with a fixed right-hand side). There are $J = 3^p$ possible conjunctive rules (each variable takes value 1, 0, or is not in the conjunction at all). Once again, this can easily be an astronomically large number. Clearly, both of these examples are problems of combinatorial optimization, involving searching over a set of possible solutions to find the one with minimum score.

Examples of continuous parameters are a parameter giving the mean value of a distribution or a parameter vector giving the centers of a set of clusters into which the data set has been partitioned. With continuous parameter spaces, the powerful tools of differential calculus can be brought to bear. In some special but very important special cases, this leads to closed form solutions. In general, however, these are not possible and iterative methods are needed. Clearly the case in which the parameter vector θ is unidimensional is very important, so we shall examine this first. It will give us insights into the multidimensional case, though we will see that other problems also arise in this situation. Both unidimensional and multidimensional situations can be complicated by the existence of local minima: parameter vectors with values smaller than any other similar vectors, but are not the smallest values that can be achieved. We shall explore ways in which such problems can be overcome.

Very often, the two problems of searching over a set of possible model structures and optimizing parameters within a given model go hand in hand; that is, since any single model or pattern structure typically has unknown parameters then, as well as finding the best model or pattern structure, we will also have to find the best parameters for each structure considered during the search. For example, consider the set of models in which y is predicted as a simple linear combination of some subset of the three predictor variables x_1, x_2, and x_3. One model would be $y(i) = ax_1(i) + bx_2(i) + cx_3(i)$, and others would have the same form but merely involving pairs of the predictor variables or single predictor variables. Our search will have to roam over all possible subsets of the x_j variables, as noted above, but for each subset, it will also be necessary to find the values of the parameters (a, b, and c in the case with all three variables) that minimize the score function.

This description suggests that one possible design choice, for algorithms that minimize score functions over both model structures and parameter estimates, is to nest a loop for the latter in a loop for the former. This is often used since it is relatively simple, though it is not always the most efficient approach from a computational viewpoint.

It is worth remarking at this early stage that in some data mining algorithms the focus is on finding sets of models, patterns, or regions within parameter space, rather than just the single best model, pattern, or parameter vector, according to the chosen score function. This occurs, for example, in Bayesian averaging techniques and in searching for sets of patterns. Usually (although, as always, there are exceptions) in such frameworks similar general principles of search and optimization will arise as in the single model/pattern/parameter case and, so in the interests of simplicity of presentation we will focus primarily on the problem of finding the single best model, pattern, and/or parameter-vector.

8.2 Searching for Models and Patterns

8.2.1 Background on Search

This subsection discusses some general high level issues of search. In many practical data mining situations we will not know ahead of time what particular model structure M or pattern structure ρ is most appropriate to solve our task, and we will search over a family of model structures $M = \{M_1,..., M_K\}$ or pattern structures $P = \{\rho_1,..., \rho_J\}$. We gave some examples of this earlier: finding the best subset of variables in a linear regression problem and finding the best set of conditions to include in the left -hand side of a conjunctive rule. Both of these problems can be considered "best subsets" problems, and have the general characteristic that a combinatorially large number of such solutions can be generated from a set of p "components" (p variables in this case). Finding "best subsets" is a common problem in data mining. For example, for predictive classification models in general (such as nearest neighbor, naive Bayes, or neural network classifiers) we might want to find the subset of variables that produces the lowest classification error rate on a validation data set.

A related model search problem, that we used as an illustration is that of finding the best tree-structured classifier from a "pool" of p variables. This has even more awesome combinatorial properties. Consider the problem of searching over all possible binary trees (that is, each internal node in the tree has two children). Assume that all trees under consideration have depth p so that there are p variables on the path from the root node to any leaf node. In addition, let any variable be eligible e to appear at any node in the tree, remembering that each node in a classification tree contains a test on a single variable, the outcomes of which define which branch is taken from that node. For this family of trees there are on the order of p^{2^p} different tree structures that is, p^{2^p} classification trees that differ from each other in the specification of at least one internal node. In practice, the number of possible tree structures will in fact be larger since we also want to consider various subtrees of the full-depth trees. Exhaustive search over all possible trees is clearly infeasible!

We note that from a purely mathematical viewpoint one need not necessarily distinguish between different model structures in the sense that all such model structures could be considered as special cases of a single "full" model, with appropriate parameters set to zero (or some other constant that is appropriate for the model form) so that they disappear from the model. For example, the linear regression model $y = ax_1 + b$ is a special case of $y = ax_1+cx_2+dx_3+b$ with $c = d = 0$. This would reduce the model structure search problem to the type of parameter optimization problem we will discuss later in this chapter. Although mathematically correct, this viewpoint is often not the most useful way to think about the problem, since it can obscure important structural information about the models under consideration.

In the discussion that follows we will often use the word models rather than the phrase models or patterns to save repetition, but it should be taken as referring to both types of structure: the same general principles that are outlined for searching for models are also true for the problem of searching for patterns.

Some further general comments about search are worth making here:

- We noted in the opening section that finding the model or structure with the optimum score from a family M necessarily involves finding the best parameters θ_k for each model structure M_k within that family. This means that, conceptually and often in practice, a nested loop search process is needed, in which an optimization over parameter values is nested within a search over model structures.

- As we have already noted, there is typically no notion of the score function S being a "smooth" function in "model space," and thus, many of the traditional optimization techniques that rely on smoothness information (for example, gradient descent) are not applicable. Instead we are in the realm of combinatorial optimization where the underlying structure of the problem is inherently discrete (such as an index over model structures) rather than a continuous function. Most of the combinatorial optimization problems that occur in data mining are inherently intractable in the sense that the only way to guarantee that one will find the best solution is to visit all possible solutions in an exhaustive fashion.

- For some problems, we will be fortunate in that we will not need to perform a full new optimization of parameter space as we move from one model structure to the next. For example, if the score function is decomposable, then the score function for a new structure will be an additive function of the score function for the previous structure as well as a term accounting for the change in the structure. For example, adding or deleting an internal node in a classification tree only changes the score for data points belonging to the subtree associated with that node. However, in many cases, changing the structure of the model will mean that the old parameter values are no longer optimal in the new model. For example, suppose that we want to build a model to predict y from x based on two data points (x, y) = (1, 1) and (x, y) = (3, 3). First let us try very simple models of the form y = a, that is y is a constant (so that all our predictions are the same). The value of a that minimizes the sum of squared errors $(1 - a)^2+(3 - a)^2$ is 2. Now let us try the more elaborate model y = bx+a. This adds an extra term into the model. Now the values of a and b that minimize the sum of squared errors (this is a standard regression problem, although a particularly simple example) are, respectively, 0 and 1. We see that the estimate of depends upon what else is in the model. It is possible to formalize the circumstances in which changing the model will leave parameter estimates unaltered, in terms of orthogonality of the data. In general, it is clearly useful to know when this applies, since much faster algorithms can then be developed (for example, if variables are orthogonal in a regression case, we can just examine them one at a time). However, such situations tend to arise more often in the cont ext of designed experiments than in the secondary data occurring in data mining situations. For this reason, we will not dwell on this issue here.

For linear regression, parameter estimation is not difficult and so it is straightforward (if somewhat time-consuming) to recalculate the

NOTES

optimal parameters for each model structure being considered. However, for more complex models such as neural networks, parameter optimization can be both computationally demanding as well as requiring careful "tuning" of the optimization method itself (as we will see later in this chapter). Thus, the "inner loop" of the model search algorithm can be quite taxing computationally. One way to ease the problem is to leave the existing parameters in the model fixed to their previous values and to estimate only the values of parameters added to the model. Although this strategy is clearly suboptimal, it permits a trade-off between highly accurate parameter estimation of just a few models or approximate parameter estimation of a much larger set of models.

- Clearly for the best subsets problem and the best classification tree problem, exhaustive search (evaluating the score function for all candidate models in the model family M) is intractable for any nontrivial values of p since there are 2^p and p^{2^p} models to be examined in each case. Unfortunately, this combinatorial explosion in the number of possible model and pattern structures will be the norm rather than the exception for many data mining problems involving search over model structure. Thus, without even taking into account the fact that for each model one may have to perform some computationally complex parameter optimization procedure, even simply enumerating the models is likely to become intractable for large p. This problem is particularly acute in data mining problems involving very high-dimensional data sets (large p).

- Faced with inherently intractable problems, we must rely on what are called heuristic search techniques. These are techniques that experimentally (or perhaps provably on average) provide good performance but that cannot be guaranteed to provide the best solution always. The greedy heuristic (also known as local improvement) is one of the better known examples. In a model search context, greedy search means that, given a current model M_k we look for other models that are "near" M_k (where we will need to define what we mean by "near") and move to the best of these (according to our score function) if indeed any are better than M_k.

8.2.2 The State-Space Formulation for Search in Data Mining

A general way to describe a search algorithm for discrete spaces is to specify the problem as follows:

1. State Space Representation:

We view the search problem as one of moving through a discrete set of states. For model search, each model structure M_k consists of a state in our state space. It is conceptually useful to think of each state as a vertex in a graph (which is potentially very large). An abstract definition of our search problem is that we start at some particular node (or state), say M_1, and wish to move through the state space to find the node corresponding to the state that has the highest score function.

2. Search Operators:

Search operators correspond to legal "moves" in our search space. For example, for model selection in linear regression the operators could be defined as either adding a variable to or deleting a variable from the current model. The search operators can be thought of as defining directed edges in the state space graph. That is, there is a directed edge from state M_i to M_j if there is an operator that allows one to move from one model structure M_i to another model structure M_j.

A simple example will help illustrate the concept. Consider the general problem of selecting the best subset from p variables for a particular classification model (for example, the nearest neighbor model). Let the score function be the cross-validated classification accuracy for any particular subset. Let M_k denote an individual model structure within the general family we are considering, namely all $K = 2^p - 1$ different subsets containing at least one variable. Thus, the state-space has $2^p - 1$ states, ranging from models consisting of subsets of single variables $M_1 = \{x_1\}$, $M_2 = \{x_2\}$,... all the way through to the full model with all p variables, $M_K = \{x_1,..., x_p\}$. Next we define our operators. For subset selection it is common to consider simple operators such as adding one variable at a time and deleting one variable at a time. Thus, from any state with p' variables (model structure) there are two "directions" one can "move" in the model family: add a variable to move to a state with p' + 1 variables, or delete a variable to move to a state with p' - 1 variables (figure 8.1 shows a state-space for subset selection for 4 variables with these two operators). We can easily generalize these operators to adding or deleting r variables at a time. Such "greedy local" heuristics are embedded in many data mining algorithms. Search algorithms using this idea vary in terms of what state they start from: forward selection algorithms work "forward" by starting with a minimally sized model and iteratively adding variables, whereas backward selection algorithms work in reverse from the full model. Forward selection is often the only tractable option in practice when p is very large since working backwards may be computationally impractical.

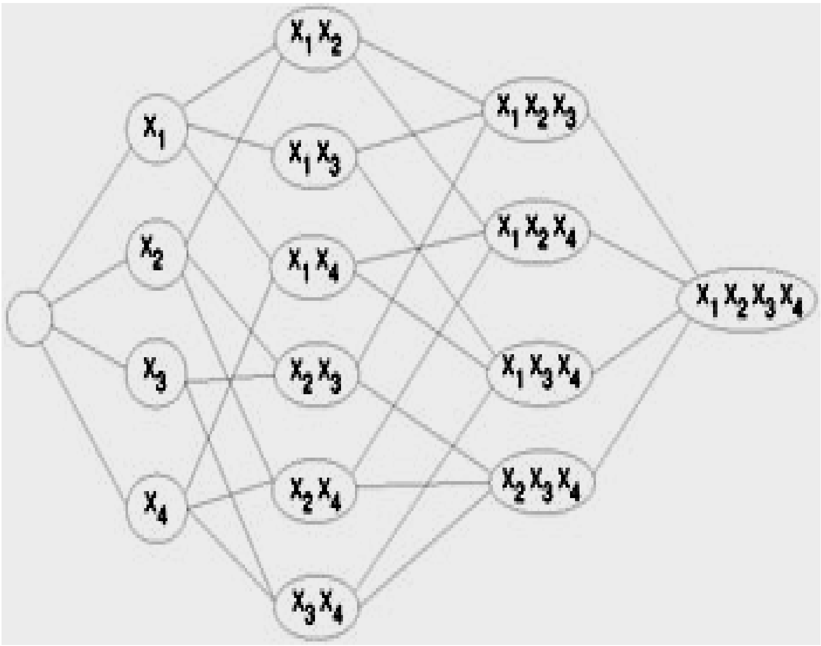


Figure 8.1: An example of a simple state-space involving four variables x_1, x_2, x_3, x_4. the node on the left is the null set i.e., no variables in the model or pattern.

It is important to note that by representing our problem in a state-space with limited connectivity we have not changed the underlying intractability of the general model search problem. To find the optimal state it will still be necessary to visit all of the exponentially many states. What the stat e-space/operator representation does is to allow us to define systematic methods for local exploration of the state-space, where the term "local" is defined in terms of which states are adjacent in the state-space (that is, which states have operators connecting them).

8.2.3 A Simple Greedy Search Algorithm

A general iterative greedy search algorithm can be defined as follows:

1. Initialize:

Choose an initial state $M^{(0)}$ corresponding to a particular model structure M_k.

2. Iterate:

Letting $M^{(i)}$ be the current model structure at the ith iteration, evaluate the score function at all possible adjacent states (as defined by the operators) and move to the best one. Note that this evaluation can consist of performing parameter estimation (or the change in the score function) for each neighboring model structure. The number of score function evaluations that must be made is the number of operators that can be applied to the current state. Thus, there is a trade-off between the number of operators available and the time taken to choose the next model in state-space.

3. Stopping Criterion:

Repeat step 2 until no further improvement can be attained in the local score function (that is, a local minimum is reached in state-space).

4. Multiple Restarts:

(optional) Repeat steps 1 through 3 from different initial starting points and choose the best solution found.

This general algorithm is similar in spirit to the local search methods we will discuss later in this chapter for parameter optimization. Note that in step 2 that we must explicitly evaluate the effect of moving to a neighboring model structure in a discrete space, in contrast to parameter optimization in a continuous space where we will often be able to use explicit gradient information to determine what direction to move. Step 3 helps avoid ending at a local minimum, rather than the global minimum (though it does not guarantee it, a point to which we return later). For many structure search problems, greedy search is provably suboptimal. However, in general it is a useful heuristic (in the sense that for many problems it will find quite good solutions on average) and when repeated with multiple restarts from randomly chosen initial states, the simplicity of the method makes it quite useful for many practical data mining applications.

8.2.4 Systematic Search and Search Heuristics

The generic algorithm described above is often described as a "hill-climbing" algorithm because (when the aim is to maximize a function) it only follows a single "path" in state- space to a local maximum of the score function. A more general (but more complexes) approach is to keep track of multiple models at once rather than just a single current model. A useful way to think about this approach is to think of a search tree, a data structure that is dynamically constructed as we search the state-space to keep track of the states that we have visited and evaluated. (This has nothing to do with classification trees, of course.) The search tree is not equivalent to the state-space; rather, it is a representation of how a particular search algorithm moves through a state-space.

An example will help to clarify the notion of a search tree. Consider again the problem of finding the best subset of variables to use in a particular classification model. We start with the "model" that contains no variables at all and predicts the value of the most likely class in the training data as its prediction for all data points. This is the root node in the search tree. Assume that we have a forward-selection algorithm that is only allowed to add variables one at a time. From the root node, there are p variables we can add to the model with no variables, and we can represent these p new models as p children of the original root node. In turn, from each of these p nodes we can add p variables, creating p children for each, or p^2 in total (clearly, $p^2 - \frac{p}{2}$ are redundant, and in practice we need to implement a duplicate-state detection scheme to eliminate the redundant nodes from the tree).

Figure 8.2 shows a simple example of a search tree for the state space of figure 8.1. Here the root node contains the empty set (no variables) and only the two best states so far are considered at any stage of the search. The search algorithm (at this point of the search) has found the two best states (as determined by the score function) to be X_2 and X_1, X_3, X_4.

NOTES

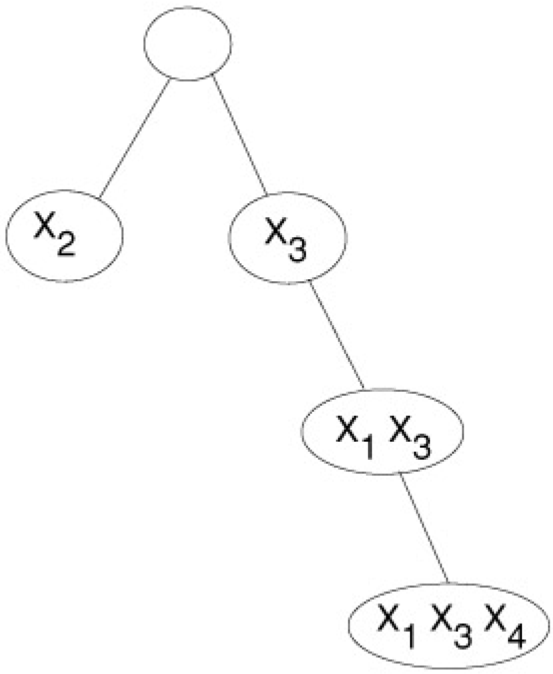


Figure 8.2: An example of a simple search tree for the state-space of figure 8.1.

Search trees evolve dynamically as we search the state-space, and we can imagine (hypothetically) keeping track of all of the leaf nodes (model structures) as candidate models for selection. This quickly becomes infeasible since at depth k in the tree there will be p^k leaf nodes to keep track of (where the root node is at depth zero and we have branching factor p). We will quickly run out of memory using this brute-force method (which is essentially breadth-first search of the search tree). A memory -efficient alternative is depth-first search, which (as its name implies) explores branches in the search tree to some maximum depth before backing up and repeating the depth-first search in a recursive fashion on the next available branch.

Both of these techniques are examples of blind search, in that they simply order the nodes to be explored lexicographically rather than by the score function. Typically, improved performance (in the sense of finding higher quality models more quickly) can be gained by exploring the more promising nodes first. In the search tree this means that the leaf node with the highest score is the one whose children are next considered; after the children are added as leaves, the new leaf with the highest score is examined. Again, this strategy can quickly lead to many more model structures (nodes in the tree) being generated than we will be feasibly able to keep in memory. Thus, for example, one can implement a beam search strategy that uses a beam width of size b to "track" only the b best models at any point in the search (equivalently to only keep track of the b best leaves on the tree). (In figure 8.2 we had b = 2.) Naturally, this might be suboptimal if the only way to find the optimal model is to first consider models that are quite suboptimal (and thus, might be outside the "beam"). However, in general, beam search can be quite effective. It is certainly often much more effective than simple hill -climbing, which is similar to depth-first search in the manner in which it explores the search tree: at any iteration there is only a single model being considered, and the next model is chosen as the child of the current model with the highest score.

8.2.5 Branch -and-Bound

A related and useful idea in a practical context is the notion of branch-and-bound. The general idea is quite simple. When exploring g a search tree, and keeping track of the best model structure evaluated so far, it may be feasible to calculate analytically a lower bound on the best possible score function from a particular (as yet unexplored) branch of the search tree. If this bound is greater than the score of the best model so far, then we need not search this subtree and it can be pruned from further consideration. Consider, for example, the problem of finding the best subset of k variables for classification from a set of p variables where we use the training set error rate as our score function. Define a tree in which the root node is the set of all p variables, the immediate child nodes are the p nodes each of which have a single variable dropped (so they each have p – 1 variables), the next layer has two variables dropped (so there are $\binom{p}{2}$ unique such nodes, each with p - 2 variables), and so on down to the $\binom{p}{k}$ leaves that each contain subsets of k variables (these are the candidate solutions). Note that the training set error rate cannot decrease as we work down any branch of the tree, since lower nodes are based on fewer variables.

Now let us begin to explore the tree in a depth-first fashion. After our depth-first algorithm has descended to visit one or more leaf nodes, we will have calculated scores for the models (leaves) corresponding to these sets of k variables. Clearly the smallest of these is our best candidate k -variable model so far. Now suppose that, in working down some other branch of the tree, we encounter a node that has a score larger than the score of our smallest k - variable node so far. Since the score cannot decrease as we continue to work down this branch, there is no point in looking further: nodes lower on this branch cannot have smaller training set error rate than the best k -variable solution we have already found. We can thus save the effort of evaluating nodes further down this branch. Instead, we back up to the nearest node above that contained an unexplored branch and begin to investigate that. This basic idea can be improved by ordering the tree so that we explore the most promising nodes first (where "promising" means they are likely to have low training set error rate). This can lead to even more effective pruning. This type of general branch and bound strategy can significantly improve the computational efficiency of model search. (Although, of course, it is not a guaranteed solution many problems are too large even for this strategy to provide a solution in a reasonable time.)

These ideas on searching for model structure have been presented in a very general form. More effective algorithms can usually be designed for specific model structures and score functions. Nonetheless, general principles such as iterative local improvement, beam search, and branch-and-bound have significant practical utility and recur commonly under various guises in the implementation of many data mining algorithms.

8.3 Parameter Optimization Methods

8.3.1 Parameter Optimization: Background

Let $S(\theta) = S(\theta|D, M)$ be the score function we are trying to optimize, where θ are the parameters of the model. We will usually suppress the explicit dependence on D and M for simplicity. We will now assume that the model structure M is fixed (that is, we are temporarily in the inner loop of parameter estimation where there may be an outer loop over multiple model structures). We will also assume, again, that we are trying to minimize S, rather than maximize it. Notice that S and g(S) will be minimized for the same value of θ if g is a monotonic function of S (such as log S).

In general θ will be a d-dimensional vector of parameters. For example, in a regression model θ will be the set of coefficients and the intercept. In a tree model, θ will be the thresholds for the splits at the internal nodes. In an artificial neural network model, θ will be a specification of the weights in the network.

In many of the more flexible models we will consider (neural networks being a good example), the dimensionality of our parameter vector can grow very quickly. For example, a neural network with 10 inputs and 10 hidden units and 1 output, could have 10 × 10 + 10 = 110 parameters. This has direct implications for our optimization problem, since it means that in this case (for example) we are trying to find the minimum of a nonlinear function in 110 dimensions.

Furthermore, the shape of this potentially high -dimensional function may be quite complicated. For example, except for problems with particularly simple structure, S will often be multimodal. Moreover, since S = S(?|D, M) is a function of the observed data D, the precise structure of S for any given problem is data-dependent. In turn this means that we may have a completely different function S to optimize for each different data set D, so that (for example) it may be difficult to make statements about how many local minima S has in the general case.

Commonly used score functions can be written in the form of a sum of local error functions (for example, when the training data points are assumed to be independent of each other):

$$S(\theta) = \sum_{i=1}^{N} e\left(y(i), \hat{y}_\theta(i)\right)$$

Where $\hat{y}_\theta(i)$ is our model's estimate of the target value y(i) in the training data, and e is an error function measuring the distance between the model's prediction and the target (such as square error or log-likelihood). Note that the complexity in the functional form S (as a function of θ) can enter both through the complexity of the model structure being used (that is, the functional form of $\hat{y}$) and also through the functional form of the error function e. For example, if $\hat{y}$ is linear in θ and e is defined as squared error, then S will be quadratic in θ , making the optimization problem relatively straightforward since a quadratic function has only a single (global) minimum or maximum. However, if $\hat{y}$ is generated by a more complex model or if e is more complex as a function of θ, S will not necessarily be a simple smooth function of θ with a single easy-to-find extremum. In general, finding the parameters θ that minimize S (θ) is usually equivalent to the problem of minimizing a complicated function in a high-dimensional space.

Let us define the gradient function of S as

$$g(\theta) = \nabla_\theta S(\theta) = \left(\frac{\partial S(\theta)}{\partial \theta_1}, \frac{\partial S(\theta)}{\partial \theta_2}, \ldots, \frac{\partial S(\theta)}{\partial \theta_d}\right),$$

which is a d-dimensional vector of partial derivatives of S evaluated at ?. In general, $\nabla_\theta S(\theta) = 0$ is a necessary condition for an extremum (such as a minimum) of S at θ. This is a set of d simultaneous equations (one for each partial derivative) in d variables. Thus, we can search for solutions θ (that correspond to extrema of S(θ)) of this set of d equations.

We can distinguish two general types of parameter optimization problems:

1. The first is when we can solve the minimization problem in closed form. For example, if S(θ) is quadratic in θ , then the gradient g(θ) will be linear in θ and the solution of ∇ S(θ) = 0 involves the solution of a set of d linear equations. However, this situation is the exception rather than the rule in practical data mining problems.

2. The second general case occurs when S(θ) is a smooth nonlinear function of θ such that the set of d equations g(θ) = 0 does not have a direct closed form solution. Typically we use iterative improvement search techniques for these types of problems, using local information about the curvature of S to guide our local search on the surface of S. These are essentially hill -climbing or descending methods (for example, steepest descent). The backpropagation technique used to train neural networks is an example of such a steepest descent algorithm.

Since the second case relies on local information, it may end up converging to a local minimum rather than the global minimum. Because of this, such methods are often supplemented by a stochastic component in which, to take just one example, the optimization procedure starts several times from different randomly chosen starting points.

8.3.2 Closed Form and Linear Algebra Methods

Consider the special case when S(θ) is a quadratic function of θ . This is a very useful special case since now the gradient g(θ) is linear in θ and the minimum of S is the unique solution to the set of d linear equations g() = 0 (assuming the matrix of second derivatives of S at these solutions satisfies the condition of being positive definite). This is illustrated in the context of multiple regressions (which usually uses a sum of squared errors score function). In general, since such problems can be framed as solving for the inverse of an d × d matrix, the complexity of solving such linear problems tends to scale in general as $O(nd^2 + d^3)$, where it takes order of nd^2 steps to construct the original matrix of interest and order of d^3 steps to invert it.

8.3.3 Gradient-Based Methods for Optimizing Smooth Functions

In general of course, we often face the situation in which S(θ) is not a simple function of θ with a single minimum. For example, if our model is a neural network with nonlinear functions in the hidden units, then S will be a relatively complex nonlinear function of θ with multiple local minima. We have already noted that many approaches are based on iteratively repeating some local improvement to the model.

The typical iterative local optimization algorithm can be broken down into four relatively simple components:

1. **Initialize:** Choose an initial value for the parameter vector θ = θ^0 (this is often chosen randomly).

2. **Iterate:** Starting with i = 0, let

$$\theta^{i+1} = \theta^i + \lambda^i \mathbf{v}^i$$

where **v** is the direction of the next step (relative to θ^i in parameter space) and θ^i determines the distance. Typically (but not necessarily) $\mathbf{v}^i$ is chosen to be in a direction of improving the score function.

3. **Convergence:** Repeat step 2 until S(θ^i) appears to have attained a local minimum.

4. **Multiple Restarts:** Repeat steps 1 through 3 from different initial starting points and choose the best minimum found.

Particular methods based on this general structure differ in terms of the chosen direction $\mathbf{v}^i$ in parameter space and the distance θ^i moved along the chosen direction, amongst other things. Note that this is this algorithm has essentially the same design as the one for local search among a set of discrete states, except that here we are moving in continuous d-dimensional space rather than taking discrete steps in a graph.

The direction and step size must be determined from local information gathered at the current point of the search for example, whether first derivative or second derivative information is gathered to estimate the local curvature of S. Moreover, there are important trade-offs between the quality of the information gathered and the resources (time, memory) required to calculate this information. No single method is universally superior to all others; each has advantages and disadvantages.

All of the methods discussed below require specification of initial starting points and a convergence (termination) criterion. The exact specifications of these aspects of the algorithm can vary from application to application. In addition, all of the methods are used to try to find a local extremum of S(θ). One must check in practice that the found solution is in fact a minimum (and not a maximum or saddle point). In addition, for the general case of a nonlinear function S with multiple minima, little can be said about the quality of the local minima relative to the global minima without carrying out a brute-force search over the entire space (or using sophisticated probabilistic arguments that are beyond this text). Despite these reservations, the optimization techniques that follow are extremely useful in practice and form the core of many data mining algorithms.

8.3.4 Univariate Parameter Optimization

Consider first the special case in which we just have a single unknown parameter θ and we wish to minimize the score function S(θ) (for example, figure 8.3). Although in practical data mining situations we will usually be optimizing a model with more than just a single parameter, the univariate case is nonetheless worth looking at, since it clearly illustrates some of the general principles that are relevant to the more general multivariate optimization problem. Moreover, univariate search can serve as a component in a multivariate search procedure, in which we first find the direction of search using the gradient and then decide how far to move in that direction using univariate search for a minimum along that direction.

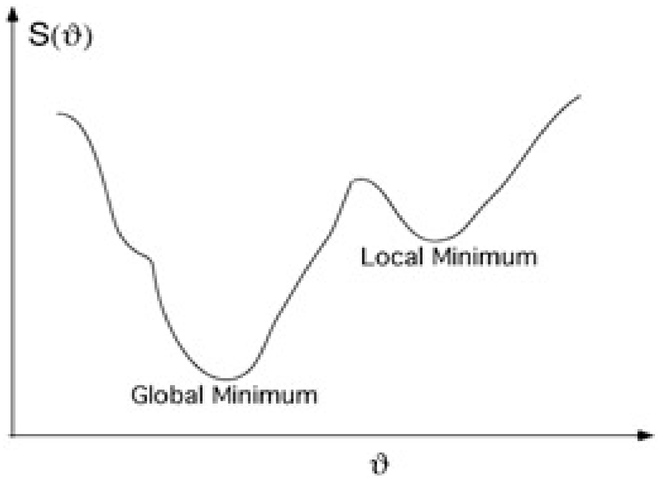


Figure 8.3: An Example of a Score Function S (θ) of a Single Univariate Parameter θ with both a Global Minimum and a Local Minimum.

Letting $g(\theta) = S'(\theta) = \frac{\partial S(\theta)}{\partial \theta}$, the minimum of S occurs wherever g(θ) = 0 and the second derivative g' (θ) > 0. If a closed form solution is possible, then we can find it and we are done. If not, then we can use one of the methods below.

The Newton-Raphson Method

Suppose that the solution occurs at some unknown point θ^s; that is, $g(\theta^s) = 0$. Now, for points θ^* not too far from θ^s we have, by using a Taylor series expansion

$$g(\theta^s) \approx g(\theta^*) + (\theta^s - \theta^*)g'(\theta^*),$$

where this linear approximation ignores terms of order $(\theta^s - \theta^*)^2$ and above. Since θ^s

NOTES

satisfies $g(\theta^S) = 0$, the left -hand side of this expression is zero. Hence, by rearranging terms we get

$$\theta^s \approx \theta^* - \frac{g(\theta^*)}{g'(\theta^*)}.$$

In words, this says that given an initial value θ^*, then an approximate solution of the equation $g(\theta^S) = 0$ is given by adjusting θ as indicated in equation. By repeatedly iterating this, we can in theory get as close to the solution as we like. This iterative

process is the Newton-Raphson (NR) iterative update for univariate optimization based on first and second derivative information. The ith step is given by

$$\theta^{i+1} = \theta^i - \frac{g(\theta^i)}{g'(\theta^i)}.$$

The effectiveness of this method will depend on the quality of the linear approximation in equation. If the starting value is close to the true solution θ^S then we can expect the approximation to work well; that is, we can locally approximate the surface around $S(\theta^*)$

as parabolic in form (or equivalently, the derivative $g(\theta)$ is linear near θ^* and θ^S). In fact, when the current θ is close to the solution θ^S, the convergence rate of the NR method is quadr atic in the sense that the error at step i of the iteration $e_i = |\theta^i - \theta^S|$ can be recursively written as

$$e_i \propto e_{i-1}^2.$$

To use the Newton-Raphson update, we must know both the derivative function $g(\theta)$ and the second derivative $g'(\theta)$ in closed form. In practice, for complex functions we may not have closed - form expressions, necessitating numerical approximation of $g(\theta)$ and $g'(\theta)$, which in turn may introduce more error into the determination of where to move in parameter space. Generally speaking, however, if we can evaluate the gradient and second derivative accurately in closed form, it is advantageous to do so and to use this information in the course of moving through parameter space during iterative optimization.

The drawback of NR is, of course, that our initial estimate θ^{i} may not be sufficiently close to the solution θ^{S} to make the approximation work well. In this case, the NR step can easily overshoot the true minimum of S and the method need not converge at all.

The Gradient Descent Method

An alternative approach, which can be particularly useful early in the optimization process (when we are potentially far from θ^{S}), is to use only the gradient information (which provides at least the correct direction to move in for a 1-dimensional problem) with a heuristically chosen step size θ:

$$\theta^{i+1} = \theta^{i} - \lambda g(\theta^{i}).$$

The multivariate version of this method is known as gradient (or steepest) descent. Here θ is usually chosen to be quite small to ensure that we do not step too far in the chosen direction. We can view gradient descent as a special case of the NR method, whereby the second derivative information $\frac{1}{g^{i}[(\theta]^{2})}$ is replaced by a constant θ.

Momentum-Based Methods

There is a practical trade-off in choosing θ. If it is too small, then gradient descent may converge very slowly indeed, taking very small steps at each iteration. On the other hand, if θ is too large, then the guarantee of convergence is lost, since we may overshoot the minimum by stepping too far. We can try to accelerate the convergence of gradient descent by adding a momentum term:

$$\theta^{i+1} = \theta^{i} + \Delta^{i}$$

where θ^{i} is defined recursively as

$$\Delta^{i} = -\lambda g(\theta^{i}) + \mu \Delta^{i-1}$$

and where μ is a "momentum" parameter, $0 \leq \mu \leq 1$. Note that $\mu = 0$ gives us the standard gradient descent method of equation, and $\mu > 0$ adds a "momentum" term in the sense that the current direction

Δ^i is now also a function of the previous direction Δ^{i-1}. The effect of μ in regions of low curvature in S is to accelerate convergence (thus, improving standard gradient descent, which can be very slow in such regions) and fortunately has little effect in regions of high curvature. The momentum heuristic and related ideas have been found to be quite useful in practice in training models such as neural networks.

Bracketing Methods

For functions which are not well behaved (if the derivative of S is not smooth, for example) there exists a different class of scalar optimization methods that do not rely on any gradient information at all (that is, they work directly on the function S and not its derivative g). Typically these methods are based on the notion of bracketing finding a bracket [θ_1, θ_2] that provably contains an extremum of the function. For example, if there exists a "middle" θ value θ_m, such that $\theta_1 > \theta_m > \theta_2$ and S(θ_m) is less than both S(θ_1) and S(θ_2), then clearly a local minimum of the function S must exist between θ_1 and θ_2 (assuming that S is continuous). One can use this idea to fit a parabola through the three points θ_1, θ_m, and θ_2 and evaluate S(θ_p) where θ_p is located at the minimum value of parabola. Either θ_p is the desired local minimum, or else we can narrow the bracket by eliminating θ_1 or θ_2 and iterating with another parabola. A variety of methods exist that use this idea with varying degrees of sophistication (for example, a technique known as Brent's method is widely used). It will be apparent from this outline that bracketing methods are really a search strategy. We have included them here, however, partly because of their importance in finding optimal values of parameters, and partly because they rely on the parameter space having a connected structure (for example, ordinality) even if the function is being minimized is not continuous.

8.3.5 Multivariate Parameter Optimization

We now move on to the much more difficult problem we are usually faced with in practice, namely, finding the minimum of a scalar score function S of a multivariate parameter vector θ in d-dimensions. Many of the methods used in the multivariate case are analogous to the scalar case. On the other hand, d may be quite large for our models, so that the multidimensional optimization problem may be significantly more complex to solve than its univariate cousin. It is possible, for example, that local minima may

NOTES

be much more prevalent in high-dimensional spaces than in lower-dimensional spaces. Moreover, a problem similar (in fact, formally equivalent) to the combinatorial explosion that we saw in the discussion of search also manifests itself in multidimensional optimization; this is the curse of dimensionality that we have already encountered.

8.3.6 Constrained Optimization

Many optimization problems involve constraints on the parameters. Common examples include problems in which the parameters are probabilities (which are constrained to be positive and to sum to 1), and models that include the variance as a parameter (which must be positive). Constraints often occur in the form of inequalities, requiring that a parameter θ satisfy $c_1 \le \theta \le c_2$, for example, with c_1 and c_2 being constants, but more complex constraints are expressed as functions: $g(\theta_1,..., \theta_d) \le 0$ for example. Occasionally, constraints have the form of equalities. In general, the region of parameter vectors that satisfy the constraints is termed the feasible region.

Problems that have linear constraints and convex score functions can be solved by methods of mathematical programming. For example, linear programming methods have been used in supervised classification problems, and quadratic programming is used in support vector machines. Problems in which the score functions and constraints are nonlinear are more challenging.

Sometimes constrained problems can be converted into unconstrained problems. For example, if the feasible region is restricted to positive values of the parameters $(\theta_1,..., \theta_d)$, we could, instead, optimize over $(f_1,...,f_d)$, where $\theta_i = \phi_i^2$, i = 1,..., d. Other (rather more complicated) transformations can remove constraints of the form $c_1 \le \theta \le c_2$.

A basic strategy for removing equality constraints is through Lagrange multipliers. A necessary condition for θ to be a local minimum of the score function $S = S(\theta)$ subject to constraints $h_j(\theta) = 0$, j = 1,..., m, is that it satisfies $\nabla S(\theta) + \sum_i \lambda_j \nabla h_j(\theta) = 0$, for some scalars, λ_i. These equations and the constraints yield a system of (d + m) simultaneous (nonlinear) equations, that can be solved by standard methods (often by using a least squares routine to minimize the sum of squares of the left hand sides of the (d + m) equations). These ideas are extend to inequality constraints in the Kuhn-Tucker conditions.

Unconstrained optimization methods can be modified to yield constrained methods. For example, penalties can be added to the score function so that the parameter estimates are repelled if they should approach boundaries of the feasible region during the optimization process.

8.4 Optimization with Missing Data: The EM Algorithm

In this section we consider the special but important problem of maximizing a likelihood score function when some of the data are missing, that is, there are variables in our data set whose values are unobserved for some of the cases. It turns out that a large number of problems in practice can effectively be modeled as missing data problems. For example, measurements on medical patients where for each patient only a subset of test results are available, or application form data where the responses to some questions depends on the answers to others.

More generally, any model involving a hidden variable (i.e., a variable that cannot be directly observed) can be modeled as a missing data problem, in which the values of this variable are unknown for all n objects or individuals. Clustering is a specific example; in effect we assume the existence of a discrete-valued hidden cluster variable C taking values $\{c_1,..., c_k\}$ and the goal is to estimate the values of C (that is, the cluster labels) for each observation $\mathbf{x}(i)$, $1 \leq i \leq n$.

The Expectation-Maximization (EM) algorithm is a rather remarkable algorithm for solving such missing data problems in a likelihood context. Specifically, let $D = \{\mathbf{x}(1),..., \mathbf{x}(n)\}$ be a set of n observed data vectors. Let $H = \{z(1),..., z(n)\}$ represent a set of n values of a hidden variable Z, in one-to-one correspondence with the observed data points D; that is, z(i) is associated with data point $\mathbf{x}(i)$. We can assume Z to be discrete (this is not necessary, but is simply convenient for our description of the algorithm), in which case we can think of the unknown z(i) values as class (or cluster) labels for the data, that are hidden.

We can write the log-likelihood of the observed data as

$$l(\theta) = \log p(D|\theta) = \log \sum_H p(D, H|\theta)$$

where the term on the right indicates that the observed likelihood can be expressed as the likelihood of both the observed and hidden data, summed over the hidden data values, assuming a probabilistic model in the form p(D, H|θ) that is parametrized by a set of unknown parameters θ. Note that our optimization problem here is doubly complicated by the fact that both the parameters θ and the hidden data H are unknown.

Let Q(H) be any probability distribution on the missing data H. We can then write the log-likelihood in the following fashion:

$$\begin{aligned} l(\theta) &= \log \sum_H p(D, H|\theta) \\ &= \log \sum_H Q(H) \frac{p(D, H|\theta)}{Q(H)} \\ &\geq \sum_H Q(H) \log \frac{p(D, H|\theta)}{Q(H)} \\ &= \sum_H Q(H) \log p(D, H|\theta) + \sum_H Q(H) \log \frac{1}{Q(H)} \\ &= F(Q, \theta) \end{aligned}$$

where the inequality is a result of the concavity of the log function (known as Jensen's inequality).

The function F(Q, θ) is a lower bound on the function we wish to maximize (the likelihood l(θ)). The EM algorithm alternates between maximizing F with respect to the distribution Q with the parameters θ fixed, and then maximizing F with respect to the parameters θ with the distribution Q = p(H) fixed. Specifically:

$$\text{E-step:} \qquad Q^{k+1} = \arg\max_Q F(Q^k, \theta^k)$$

$$\text{M-step:} \qquad \theta^{k+1} = \arg\max_\theta F(Q^{k+1}, \theta^k)$$

It is straightforward to show that the maximum in the E-step is achieved when Q^{k+1} = p(H|D, θ^k), a term that can often be calculated explicitly in a relatively straightforward fashion for many models. Furthermore, for this value of Q the bound becomes tight, i.e., the inequality becomes an equality above and l(θ^k) = F(Q, θ^k).

The maximization in the M-step reduces to maximizing the first term in F (since the second term does not depend on θ), and can be written as

$$\theta^{k+1} = \arg\max_{\theta} \sum_{H} p(H|D, \theta^k) \log p(D, H|\theta^k).$$

This expression can also fortunately often be solved in closed form.

Clearly the E and M steps as defined cannot decrease $l(\theta)$ at each step: at the beginning of the M-step we have that $l(\theta^k) = F(Q^{k+1}, \theta^k)$ by definition, and the M-step further adjusts θ to maximize this F.

The EM steps have a simple intuitive interpretation. In the E-step we estimate the distribution on the hidden variables Q, conditioned on a particular setting of the parameter vector θ^k. Then, keeping the Q function fixed, in the M-step we choose a new set of parameters θ^{k+1} so as to maximize the expected log-likelihood of observed data (with expectation defined with respect to Q = p(H)). In turn, we can now find a new Q distribution given the new parameters θ^{k+1}, then another application of the M-step to get θ^{k+2}, and so forth in an iterative manner. As sketched above, each such application of the E and M steps is guaranteed not to decrease the log-likelihood of the observed data, and under fairly general conditions this in turn implies that the parameters θ will converge to at least a local maximum of the log-likelihood function.

To specify an actual algorithm we need to pick an initial starting point (for example, start with either an initial randomly chosen Q or θ) and a convergence detection method (for example, detect when any of Q, θ, or $l(\theta)$ do not change appreciably from one iteration to the next). The EM algorithm is essentially similar to a form of local hill-climbing in multivariate parameter space (as discussed in earlier sections of this chapter) where the direction and distance of each step is implicitly (and automatically) specified by the E and M steps. Thus, just as with hill-climbing, the method will be sensitive to initial conditions, so that different choices of initial conditions can lead to different local maxima. Because of this, in practice it is usually wise to run EM from different initial conditions (and then choose the highest likelihood solution) to decrease the probability of finally settling on a relatively poor local maximum. The EM algorithm can converge relatively slowly to the final parameter values, and for example, it can be combined with more traditional optimization techniques (such as Newton -Raphson) to speed up convergence in the later iterations. Nonetheless, the standard EM algorithm is widely used given the broad generality of the framework and the relative ease with which an EM algorithm can be specified for many different problems.

The computational complexity of the EM algorithm is dictated by both the number of iterations required for convergence and the complexity of each of the E and M steps. In practice it is often found that EM can converges relatively slowly as it approaches a solution, although the actual rate of convergence can depend on a variety of different factors. Nonetheless, for simple models at least, the algorithm can often converge to the general vicinity of the solution after only a few (say 5 or 10) iterations. The complexity of the E and M steps at each iteration depends on the nature of the model being fit to the data (that is, the likelihood function $p(D, H|\theta)$). For many of the simpler models (such as the mixture models discussed below) the E and M steps need only take time linear in n, i.e., each data point need only be visited once during each iteration.

8.5 Online and Single-Scan Algorithms

All of the optimization methods we have discussed so far implicitly assume that the data are all resident in main memory and, thus, that each data point can be easily accessed multiple times during the course of the search. For very large data sets we may be interested in optimization and search algorithms that see each data point only once at most. Such algorithms may be referred to as online or single-scan and clearly are much more desirable than "multiple -pass" algorithms when we are faced with a massive data set that resides in secondary memory (or further away).

In general, it is usually possible to modify the search algorithms above directly to deal with data points one at a time. For example, consider simple gradient descent methods for parameter optimization. As discussed earlier, for the "offline" (or batch) version of the algorithm, one finds the gradient $g(\theta)$ in parameter space, evaluates it at the current location θ^k, and takes a step proportional to distance θ in that direction. Now moving in the direction of the gradient $g(\theta)$ is only a heuristic, and it may not necessarily be the optimal direction. In practice, we may do just as well (at least, in the long run) if we move in a direction

NOTES

approximating that of the gradient. This idea is used in practice in an online approximation to the gradient that uses the current best estimate based both on the current location and the current and (perhaps) recent data points. The online estimates can be viewed as stochastic (or "noisy") estimates of the full gradient estimate that would be produced by the batch algorithm looking at all of the data points. There exists a general theory in statistics for this type of search technique, known as stochastic approximation, which is beyond the scope of this text but that is relevant to online parameter estimation. Indeed, in using gradient descent to find weight parameters for neural networks (for example) stochastic online search has been found to be useful in practice. The stochastic (data-driven) nature of the search is even thought to sometimes improve the quality of the solutions found by allowing the search algorithm to escape from local minima in a manner somewhat reminiscent of simulated annealing.

More generally, the more sophisticated search methods (such as multivariate methods based on the Hessian matrix) can also be implemented in an online manner by appropriately defining online estimators for the required search directions and step-sizes.

8.6 Stochastic Search and Optimization Techniques

The methods we have presented thus far on model search and parameter optimization rely heavily on the notion of taking local greedy steps near the current state. The main disadvantage is the inherent myopia of this approach. The quality of the solution that is found is largely a function of the starting point. This means that, at least with a single starting position, there is the danger that the minimum (or maximum) one finds may be a nonglobal local optimum. Because of this, methods have been developed that adopt a more global view by allowing large steps away from the current state in a nondeterministic (stochastic) manner. Each of the methods below is applicable to either the parameter optimization or model search problem, but for simplicity we will just focus here on model search in a state-space.

- **Genetic Search:** Genetic algorithms are a general set of heuristic search techniques based on ideas from evolutionary biology. The essential idea is to represent states (models in our case) as chromosomes (often encoded as binary strings) and to "evolve" a population of such chromosomes by selectively pairing chromosomes to create new offspring. Chromosomes (states) are paired based on their "fitness" (their score function) to encourage the fitter chromosomes to survive from one generation to the next (only a limited number of chromosomes are allowed to survive from one generation to the next). There are many variations on this general theme, but the key ideas in genetic search are:

- Maintenance of a set of candidate states (chromosomes) rather than just a single state, allowing the search algorithm to explore different parts of the state space simultaneously

- Creating new states to explore based on combinations of existing states, allowing in effect the algorithm to "jump" to different parts of the state-space (in contrast to the local improvement search techniques we discussed earlier)

Genetic search can be viewed as a specific type of heuristic, so it may work well on some problems and less well on others. It is not always clear that it provides better performance on specific problems than a simpler method such as local iterative improvement with random restarts. A practical drawback of the approach is the fact that there are usually many algorithm parameters (such as the number of chromosomes, specification of how chromosomes are combined, and so on) that must be specified and it may not be clear what the ideal settings are for these parameters for any given problem.

- **Simulated Annealing:** Just as genetic search is motivated by ideas from evolutionary biology, the approach in simulated annealing is motivated by ideas from physics. The essential idea is to not to restrict the search algorithm to moves in state-space that decrease the score function (for a score function we are trying to minimize), but to also allow (with some probability) moves that can increase the score function. In principle, this allows a search algorithm to escape from a local minimum. The probability of such non-decreasing moves is set to be quite high early in the process and gradually decreased as the search progresses. The decrease in this probability is analogous to the process of gradually decreasing the temperature in the physical process of annealing a metal with the goal of obtaining a low-energy state in the metal (hence the name of the method).

For the search algorithm, higher temperatures correspond to a greater probability of large moves in the parameter space, while lower temperatures correspond to greater probability of only small moves that decrease the function being taken. Ultimately, the temperature schedule reduces the temperature to zero, so that the algorithm by then only moves to states that decrease the score

NOTES

function. Thus, at this stage of the search, the algorithm will inevitably converge to a point at which no further decrease is possible. The hope is that the earlier (more random) moves have led the algorithm to the deepest "basin" in the score function surface. In fact, one of the appeals of the approach is that it can be mathematically proved that (under fairly general conditions) this will happen if one is using the appropriate temperature schedule. In practice, however, there is usually no way to specify the optimal temperature schedule (and the precise details of how to select the possible non-decreasing moves) for any specific problem. Thus, the practical application of simulated annealing reduces to (yet another) heuristic search method with its own set of algorithm parameters that are often chosen in an ad hoc manner.

We note in passing that the idea of stochastic search is quite general, where the next set of parameters or model is chosen stochastically based on a probability distribution on the quality of neighboring states conditioned on the current state. By exploring state-space in a stochastic fashion, a search algorithm can in principle spend more time (on average) in the higher quality states and build up a model on the distribution of the quality (or score) function across the state-space. This general approach has become very popular in Bayesian statistics, with techniques such as Monte Carlo Markov Chain (MCMC) being widely used. Such methods can be viewed as generalizations of the basic simulated annealing idea, and again, the key ideas originated in physics. The focus in MCMC is to find the distribution of scores in parameter or state-space, weighted by the probability of those parameters or models given the data, rather than just finding the location of the single global minimum (or maximum).

It is difficult to make general statements about the practical utility of methods such as simulated annealing and genetic algorithms when compared to a simpler approach such as iterative local improvement with random restarts, particularly if we want to take into account the amount of time taken by each method. It is important when comparing different search methods to compare not only the quality of the final solution but also the computational resources expended to find that solution. After all, if time is unlimited, we can always use exhaustive enumeration of all models to find the global optimum. It is fair to say that since stochastic search techniques typically involve considerable extra computation an d overhead (compared to simpler alternatives) that they tend to be used in practice on specialized problems involving relatively small data sets, and are often not practical from a computational viewpoint for very large data sets.

Summary

We discussed the principles of how such structures (in the form of models and patterns) can be scored in terms of how well they match the observed data. This chapter focuses on the computational methods used for model and pattern-fitting in data mining algorithms; that is, it focuses on the procedures for searching and optimizing over parameters and structures guided by the available data and our score functions. The importance of effective search and optimization is often underestimated in the data mining, statistical and machine learning algorithm literatures, but successful applications in practice depend critically on such methods.

We have explained what is meant, in the context of data mining, by the terms model and pattern. A model is a high-level description, summarizing a large collection of data and describing its important features. Often a model is global in the sense that it applies to all points in the measurement space. In contrast, a pattern is a local description, applying to some subset of the measurement space, perhaps showing how just a few data points behave or characterizing some persistent but unusual structure within the data. Examples would be a mode (peak) in a density function or a small set of outliers in a scatter plot.

Reference

1. Hand D, Mannila H. Smith P: Principles of Data Mining (PHI).

2. Pujari A: Data Mining Techniques, University Press(orient Longman.

UNIT – IV

9. Descriptive Modeling

Structure

Objective

After going through this lesson, you should be able to:

- Discuss about a describing data by probability distributions and densities
- Discuss background on cluster analysis;
- Discuss about partition-based clustering algorithms;
- Discuss about a hierarchical clustering;

This lesson is concerned with descriptive models, presenting outlines of several algorithms for finding descriptive models that are import ant in data mining contexts. We have already noted that data mining is usually concerned with building empirical models that are not based on some underlying theory about the mechanism through which the data arose, but that are simply a description of the observed data. The fundamental objective is to produce insight and understanding about the structure of the data, and to enable us to see its important features. Beyond this, of course, we hope to discover unsuspected structure as well as structure that is interesting and valuable in some sense. A good model can also be thought of as generative in the sense that data generated according to the model will have the same characteristics as the real data from which the model was produced. If such synthetically generated data have features not possessed by the original data, or do not possess features of the original data (such as, for example, correlations between variables), then the model is a poor one: it is failing to summarize the data adequately.

There are, in fact, many different types of model, each related to the others in various ways (special cases, generalizations, and different ways of looking at the same structure, and so on). We cannot hope to examine all possible models types in detail in a single chapter. Instead we will look at just some of the more important types, focusing on methods for density estimation and cluster analysis in particular. The reader is alerted to the facts that are other descriptive techniques in the literature (techniques such as structural equation modeling or factor analysis for example) that we do not discuss here. One point is worth making at the start. Since we are concerned here with global models, with structures that are representative of a mass of objects in some sense, then we do not need to worry about failing to detect just a handful of objects possessing some property; that is, in this chapter we are not concerned with patterns.

9.2 Describing Data by Probability Distributions and Densities

9.2.1 Introduction

For data that are drawn from a larger population of values, or data that can be regarded as being drawn from such a larger population (for example, because the measurements have associated

NOTES

measurement error), describing data in terms of their underlying distribution or density function is a fundamental descriptive strategy. Adopting our usual notation of a p-dimensional data matrix, with variables $X_1, ..., X_p$, our goal is to model the joint distribution or density $f(X_1, ..., X_p)$ as first encountered. For convenience, we will refer to "densities" in this discussion, but the ideas apply to discrete as well as to continuous X variables.

The joint density in a certain sense provides us with complete information about the variables $X_1, ..., X_p$. Given the joint density, we can answer any question about the relationships among any subset of variable; for example, are X_3 and X_7 independent? Thus, we can answer questions about the conditional density of some variables given others; for example, what is the probability distribution of X_3 given the value of X_7, $f(x_3 \mid x_7)$?.

There are many practical situations in which knowing the joint density is useful and desirable. For example, we may be interested in the modes of the density (for real- valued Xs). Say we are looking at the variables income and credit-card spending for a data set of n customers at a particular bank. For large n, in a scatter plot we will just see a mass of points, many overlaid on top of each other. If instead we estimate the joint density f(income, spending) (where we have yet to describe how this would be done), we get a density function of the two dimensions that could be plotted as a contour map or as a three-dimensional display with the density function being plotted in the third dimension. The estimated joint density would in principle impart useful information about the underlying structure and patterns present in the data. For example, the presence of peaks (modes) in the density function could indicate the presence of subgroups of customers. Conversely, gaps, holes, or valleys might indicate regions where (for one reason or another) this particular bank had no customers. And the overall shape of the density would provide an indication of how income and spending are related, for this population of customers.

A quite different example is given by the problem of generating approximate answers to queries for large databases (also known as query selectivity estimation). The task is the following: given a query (that is, a condition that the observations must satisfy), estimate the fraction of rows that satisfy this condition (the selectivity of the query). Such estimates are needed in query optimization in database systems, and a single query optimization task might need hundreds of such estimates. If we have a good approximation for the joint distribution of the data in the database, we can use it to obtain approximate selectivity in a computationally efficient manner.

Thus, the joint density is fundamental and we will need to find ways to estimate and conveniently summarize it (or its main features).

9.2.2 Score Functions for Estimating Probability Distributions and Densities

As we have noted in earlier chapters, the most common score function for estimating the parameters of probability functions is the likelihood (or, equivalently by virtue of the monotonicity of the log transform, the log-likelihood). As a reminder, if the probability function of random variables **X** is $f(\mathbf{x}; \theta)$; where θ are the parameters that need to be estimated, then the log-likelihood is log $f(D|\theta)$ where D = {**x**(1), ..., **x**(n)} is the observed data. Making the common assumption that that the separate rows of the data matrix have arisen independently, this becomes

$$S_L(\theta) = -\sum_{i=1}^{n} \log f(\mathbf{x}(i); \theta).$$

If f has a simple functional form (for example, if it has the form of the single univariate distributions outlined in the appendix) then this score function can usually be minimized explicitly, producing a closed form estimator for the parameters θ. However, if f is more complex, iterative optimization methods may be required.

Despite its importance, the likelihood may not always be an adequate or appropriate measure for comparing models. In particular, when models of different complexity (for example, Normal densities with covariance structures parameterized in terms of different numbers of parameters) are compared then difficulties may arise. For example, with a nested series of models in which higher-level models include lower-level ones as special cases; the more flexible higher level models will always have a greater likelihood. This will come as no surprise. The likelihood score function is a measure of how well the model fits the data, and more flexible models necessarily fit the data no worse (and usually better) than a nested less flexible model. This means that likelihood will be appropriate in situations in which we are using it as a score function to summarize a complete body of data (since then our aim is simply closeness of fit between the simplifying description and the raw data) but not if we are using it to select a

NOTES

single model (from a set of candidate model structures) to apply it to a sample of data from a larger population (with the implicit aim being to generalize beyond the data actually observed). In the latter case, we can solve the problem by modifying the likelihood to take the complexity of the model into account. For example, the BIC (Bayesian Information Criterion) score function was defined as:

$$S_{BIC}(M_k) = 2S_L(\hat{\theta}_k; M_k) + d_k \log n, \quad 1 \le k \le K,$$

where d_k is the number of parameters in model M_k and $S_L(\hat{\theta}_k; M_k)$ is the minimizing value of the negative log -likelihood (achieved at $\hat{\theta}_k$).

Alternatively, we can calculate the score using an independent sample of data, producing an "out -of-sample" evaluation. Thus the validation log-likelihood (or "holdout log-likelihood") is defined as

$$S_{vl}(M_k) = \sum_{\mathbf{x} \in D_v} \log f_{M_k}(\mathbf{x}|\hat{\theta}), 1 \le k \le K,$$

where the points **x** are from the validation data set D_v, the parameters $\hat{\theta}_k$ were estimated (for example, via maximum likelihood) on the disjoint training data $D_t = D \setminus D_v$, and there are K models under consideration.

9.2.3 Parametric Density Models

There are two general classes of density function model structures: parametric and nonparametric. Parametric models assume a particular functional form (usually relatively simple) for the density function, such as a uniform distribution, a Normal distribution, an exponential distribution, a Poisson distribution, and so on. These distribution functions are often motivated by underlying causal models of generic data-generating mechanisms. Choice of what might be an appropriate density function should be based on knowledge of the variable being measured (for example, the knowledge that a variable such as income can only be positive should be reflected in the choice of the distribution adopted to model it). Parametric models can often be characterized by a relatively small number of parameters. For example, the p-dimensional Normal distribution is defined as

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)},$$

where ∑ is the p × p covariance matrix of the X variables, |∑| is the determinant of this matrix, and μ is the p-dimensional vector mean of the X s. The parameters of the model are the mean vector and the covariance matrix (thus, p + p(p + 1)/2 parameters in all). The multivariate Normal (or Gaussian) distribution is particularly important in data analysis. For example, because of the central limit theorem, under fairly broad assumptions the mean of N independent random variables (each from any distribution) tends to have a Normal distribution. Although the result is asymptotic in nature, even for relatively small values of N (e.g., N = 10) the sample mean will typically be quite Normal. Thus, if a measurement can be thought of as being made up of the sum of multiple relatively independent causes, the Normal model is often a reasonable model to adopt.

The functional form of the multivariate Normal model in above equation is less formidable than it looks. The exponent, $(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)$, is a scalar value (a quadratic form) known as the Mahalanobis distance between the data point **x** and the mean μ, denoted as $r^2_{\Sigma}(x, \mathbf{\mu})$. This is a generalization of standard Euclidean distance that takes into account (through the covariance matrix ∑) correlations in p-space when distance is calculated. The denominator is simply a normalizing constant (call it C) to ensure that the function integrates to 1 (that is, to ensure it is a true probability density function).

Thus, we can write our Normal model in significantly simplified form as

$$f(\mathbf{x}) = \frac{1}{C} e^{-r^2_{\Sigma}(\mathbf{x},\mu)/2}.$$

If we were to plot (say for p = 2) all of the points **x** that have the same fixed values of $r^2_{\Sigma}(x, \mathbf{\mu})$, (or equivalently, all of the points **x** that like on iso-density contours $f(\mathbf{x})$ = c for some constant c), we would find that they trace out an ellipse in 2-space (more generally, a hyperellipsoid in p-space), where the ellipse is centered at μ. That is, the contours describing the multivariate Normal distribution are ellipsoidal, with height falling exponentially from the center as a function of $r^2_{\Sigma}(x, \mathbf{\mu})$. Figure 9.1 provides a simple illustration in two dimensions. The eccentricity and orientation of the elliptical contours is determined by the form of S. If S is a multiple of the identity matrix (all variables have the same variance and are uncorrelated) then the contours are circles. If S is a diagonal matrix, but with different variance terms on the diagonals, then the ax es of the elliptical contours are parallel to the variable axes and the contours are elongated along the variable axes with greater variance. Finally, if some of the variables are highly correlated, the (hyper) elliptical contours will tend to be elongated along vectors defined as linear combinations of these variables. In figure 9.1, for example, the two variables X_1 and X_2 are highly correlated, and the data are spread out along the line defined by the linear combination $X_1 + X_2$.

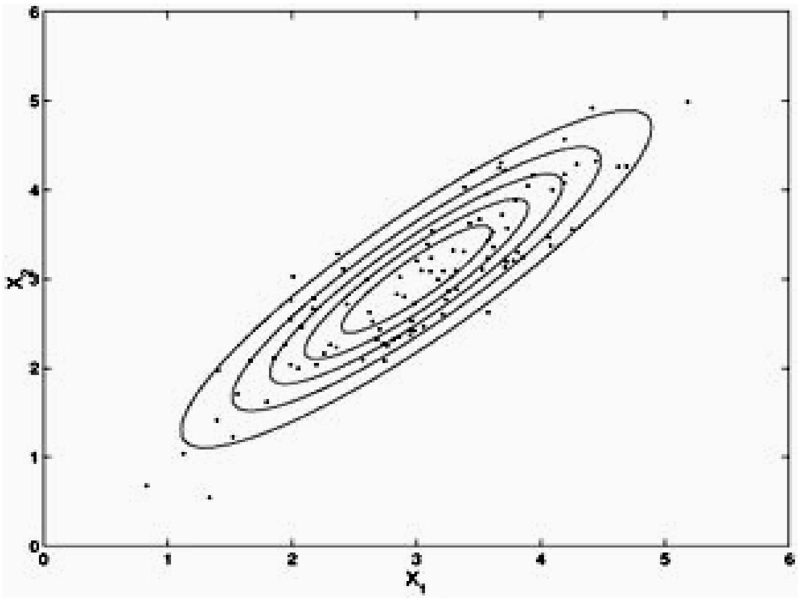


Figure 9.1: Illustration of the Density Contours for a Two - Dimensional Normal Density Function, With Mean [3, 3] and Covariance Matrix $\sum = \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix}$. Also shown are 100 Data Points simulated from this Density.

For high-dimensional data (large p) the number of parameters in the Normal model will be dominated by the $O(p^2)$ covariance terms in the covariance matrix. In practice we may not want to model all of these covariance terms explicitly, since for large p and finite n (the number of data points available) we may not get very reliable estimates of many of the covariance terms. We could, for example, instead assume that the variables are independent, which is equivalent in the Normal case to assuming that the covariance matrix has a diagonal structure (and, hence, has only p parameters). (Note that if we assume that $\sum$ is diagonal it is easy to show that the p-dimensional multivariate Normal density factors into a product of p univariate Normal distributions, a necessary and sufficient condition for independence of the p variables.) An even more extreme assumption would be to assume that $\sum = \sigma^2 I$ where I is the identity matrix that is, that the data has the same variance for all p variables as well as being independent.

Independence is a highly restrictive assumption. A less restrictive assumption would be that the covariance matrix had a block diagonal structure: we assume that there are groups of variables (the "blocks") that are dependent, but that variables are independent across the groups. In general, all sorts of assumptions may be possible, and it is important, in practice, to test the assumptions. In this regard, the multivariate Normal distribution has the attractive property that two variables are conditionally independent, given the other variables, if and only if the corresponding element of the inverse of the covariance matrix is zero. This means that the inverse covariance matrix $\sum^{-1}$ reveals the pattern of relationships between the variables. (Or, at least, it does in principle: in fact, of course, it will be necessary to decide whether a small value in the inverse covariance matrix is sufficiently small to be regarded as zero.) It also means that we can hypothesize a graphical model in which there are no edges linking the nodes corresponding to variables that have a small value in this inverse matrix.

It is important to test the assumptions made in a model. Specific statistical goodness-of- fit tests are often available, but even simple eyeballing can be revealing. The simple histogram, or one of its more sophisticated cousins outlined, can immediately reveal constraints on permissible ranges (for example, the non -negativity of income noted above), lack of symmetry, and so on. If the assumptions are not justified, then analysis of some transformation of the raw scores may be appropriate. Unfortunately, there are no hard -and-fast rules about whether or not an assumption is justified. Slight departures may well be unimportant but it will depend on the problem. This is part of the art of data mining. In many situations in which the distributional assumptions break down we can obtain perfectly legitimate estimates of parameters, but statistical tests are invalid. For example, we can physically fit a regression model using the least squares score function, whether or not the errors are Normally distributed, but hypothesis tests on the estimated

NOTES

parameters may well not be accurate. This might matter during the model building process in helping to decide whether or not to include a variable but it may not matter for the final model. If the final model is good for its purpose (for example, predictive accuracy in regression) that is sufficient justification for it to be adopted.

Fitting a p-dimensional Normal model is quite easy. Maximum likelihood (or indeed Bayesian) estimation of each of the means and the covariance terms can be defined in closed form, and takes only O(n) steps for each parameter, so $O(np^2)$ in total. Other well-known parametric models (such as those defined in the appendix) also usually possess closed-form parameter solutions that can be calculated by a single pass through the data.

The Normal model structure is a relatively simple and constrained model. It is unimodal and symmetric about the axes of the ellipse. It is parametrized completely in terms of its mean vector and covariance matrix. However, it follows from this that nonlinear relationships cannot be captured, nor can any form of multimodality or grouping. The reader should also note that although the Normal model is probably the most widely-used parametric model in practice, there are many other density functions with different "shapes" that are very useful for certain applications (e.g., the exponential model, the log-normal, the Poisson, the Gamma: the interested reader is referred to the appendix). The multivariate t-distribution is similar in form to the multivariate Normal but allows for longer tails, and is found useful in practical problems where more data can often occur in the tails than a Normal model would predict.

9.2.4 Mixture Distributions and Densities

This can be viewed as the next natural step in complexity in our discussion of density modeling: namely, the generalization from parametric distributions to weighted linear combinations of such functions, providing a general framework for generating more complex density and distribution models as combinations of simpler ones. Mixture models are quite useful in practice for modeling data when we are not sure what specific parametric form is appropriate (later in this chapter we will see how such mixture models can also be used for the task of clustering).

It is quite common in practice that a data set is heterogeneous in the sense that it represents multiple different subpopulations or groups, rather than one single homogeneous group. Heterogeneity is particularly prevalent in very large data sets, where the data may represent different underlying phenomena that have been collected to form one large data set. This is a histogram of the number of weeks owners of a particular credit card used that card to make supermarket purchases in 1996. As we pointed out there, the histogram appears to be bimodal, with a large and obvious mode to the left and a smaller, but nevertheless possibly important mode to the right. An initial stab at a model for such data might be that it follows a Poisson distribution (despite being bounded above by 52), but this would not have a sufficiently heavy tail and would fail to pick up the right -hand mode. Likewise, a binomial model would also fail to follow the right -hand mode. Something more sophisticated and flexible is needed. An obvious suggestion here is that the empirical distribution should be modeled by a theoretical distribution that has two components. Perhaps there are two kinds of people: those who are unlikely to use their credit card in a supermarket and those who do so most weeks. The first set of people could be modeled by a Poisson distribution with a small probability. The second set could be modeled by a reversed Poisson distribution with its mode around 45 or 46 weeks (the position of the mode would be a parameter to be estimated in fitting the model to the data). This leads us to an overall distribution of the form

$$f(x) = \pi \frac{(\lambda_1)^x e^{-\lambda_1}}{x!} + (1-\pi)\frac{(\lambda_2)^{52-x} e^{-\lambda_2}}{(52-x)!},$$

where x is the value of the random variable X taking values between 0 and 52 (indicating how many weeks a year a person uses their card in a supermarket), and $\lambda_1 > 0$, $\lambda_2 > 0$ are parameters of the two component Poisson models. Here p is the probability that a person belongs to the first group, and, given this, the expression $\frac{\lambda_1^x e^{-\lambda_1}}{x!}$ gives the probability that this person will use their card x times in the year. Likewise, 1 - π is the probability that this person belong to the second group and $\frac{\lambda_1^{52-x} e^{-\lambda_2}}{(52-x)!}$ is the conditional probability that such a person will use their card x times in the year.

One way to think about this sort of model is as a two-stage generative process for a particular individual. In the first step there is a probability π (and 1 - π) that the individual comes from one

NOTES

group or the other. In the second step, an observation x is generated for that person according to the component distribution he or she was assigned to in the first step.

The general form of a mixture distribution (for multivariate **x**) is

$$f(\mathbf{x}) = \sum_{k=1}^{K} \pi_k f_k(\mathbf{x}; \theta_k),$$

where π_k is the probability that an observation will come from the k th component (the so- called k th mixing proportion or weight), K is the number of components, $f_k(x; \theta_k)$ is the distribution of the k th component, and θ_k is the vector of parameters describing the k th component (in the Poisson mixture example above, each θ_k consisted of a single parameter λ_k). In most applications the component distributions f_k have the same form, but there are situations where this is not the case. The most widely used form of mixture distribution has Normal components. Note that the mixing proportions p_k must lie between 0 and 1 and sum to 1.

Some examples of the many practical situations in which mixture distributions might be expected on theoretical grounds are the length distribution of fish (since they hatch at a specific time of the year), failure data (where there may be different causes of failure, and each cause results in a distribution of failure times), time to death, and the distribution of characteristics of heterogeneous populations of people (e.g., heights of males and females).

9.2.5 The EM Algorithm for Mixture Models

Unlike the simple parametric models discussed earlier in this chapter, there is generally no direct closed-form technique for maximizing the likelihood score function when the underlying model is a mixture model, given a data set D = {**x**(1), ..., **x**(n)}. This is easy to see by writing out the log-likelihood for a mixture model we get a sum of terms such as $\log(\sum_k \pi_k f_k(\mathbf{x};\theta_k))$, leading to a nonlinear optimization problem (unlike, for example, the closed form solutions for the multivariate Normal model).

Over the years, many different methods have been applied in estimating the parameters of mixture distributions given a particular mixture form. One of the more widely used modern methods in this context is the EM approach. The mixture model can be regarded as a distribution in which the class labels are missing. If we knew

NOTES

these labels, we could get closed-form estimates for the parameters of each component by partitioning the data points into their respective groups. However, since we do not know the origin of each data point, we must simultaneously try to learn which component a data point originated from and the parameters of these components. This "chicken-and-egg" problem is neatly solved by the EM algorithm; it starts with some guesses at the parameter values for each component, then calculates the probability that each data point came from one of the K components (this is known as the E-step), calculates new parameters for each component given these probabilistic memberships (this is the M- step, and can typically be carried out in closed form), recalculates the probabilistic memberships, and continues on in this manner until the likelihood converges. The complexity of the EM algorithm depends on the complexity of the E and M steps at each iteration. For multivariate normal mixtures with K components the computation will be dominated by the calculation of the K covariance matrices during the M-step at each iteration. In p dimensions, with K clusters, there are $O(Kp^2)$ covariance parameters to be estimated, and each of these requires summing over n data points and membership weights, leading to a $O(Kp^2 n)$ time-complexity per step. For univariate mixtures (such as the Poisson above) we get O(Kn). The space-complexity is typically O(Kn) to store the K membership probability vectors for each of the n data points **x**(i). However, for large n, we often need not store the n × K membership probability matrix explicitly, since we may be able to calculate the parameter estimates during each M-step incrementally via a single pass through the n data points.

EM often provides a large increase in likelihood over the first few iterations and then can slowly converge to its final value; however the likelihood function as a function of iterations need not be concave. For example, figure 9.2 illustrates the convergence of the log-likelihood as a function of the EM iteration number, for a problem involving fitting Gaussian mixtures to a two-dimensional medical data set. For many data sets and models we can often find a reasonable solution in only 5 to 20 iterations of the algorithm. Each solution provided by EM is of course a function of where one started the search (since it is a local search algorithm), and thus, multiple restarts from randomly chosen starting points are a good idea to try to avoid poor local maxima. Note that as either (or both) K and p increase; the number of local maxima of the likelihood can increase greatly as the dimensionality of the parameter space scales accordingly.

NOTES

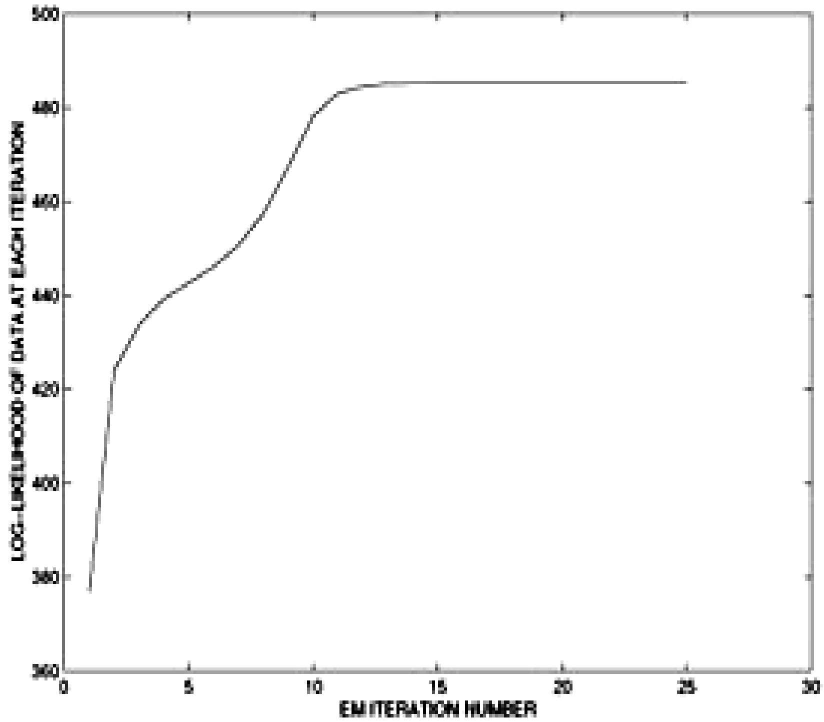


Figure 9.2: The log -likelihood of the red-blood cell data under a two -component normal mixture model as a function of Iteration number.

Sometimes caution has to be exercised with maximum likelihood estimates of mixture distributions. For example, in a normal mixture, if we put the mean of one component equal to one of the sample points and let its standard deviation tend to zero, the likelihood will increase without limit. The maximum likelihood solution in this case is likely to be of limited value. There are various ways around this. The largest finite value of the likelihood might be chosen to give the estimated parameter values. Alternatively, if the standard deviations are constrained to be equal, the problem does not arise. A more general solution is to set up the problem in a Bayesian context, with priors on the parameters, and maximize the MAP score function (for example) instead of the likelihood. Here the priors provide a framework for "biasing" the score function (the MAP score function) away from problematic regions in parameter space in a principled manner. Note that the EM algorithm generalizes easily from the case of maximizing likelihood to maximizing MAP (for example, we replace the M-step with an MAP-step, and so forth).

Another problem that can arise is due to lack of identifiability. A family of mixture distributions is said to be identifiable if and only if the fact that two members of the family are equal,

$$\sum_{k=1}^{c} \pi_k f(x;\theta_k) = \sum_{j=1}^{c'} \pi'_j f(x;\theta'_j),$$

implies that c = c' , and that for all k there is some j such that $\pi_k = \pi'_j$ and $\theta_k = \theta'_j$. If a family is not identifiable, then two different members of it may be indistinguishable, which can lead to problems in estimation.

Non-identifiability is more of a problem with discrete distributions than continuous ones because, with m categories, only m - 1 independent equation can be set up. For example, in the case of a mixture of several Bernoulli components, there is effectively only a single piece of information available in the data, namely, the proportion of 1s that occur in the data. Thus, there is no way of estimating the proportions that are separately due to each component Bernoulli, or the parameters of those components.

9.2.6 Nonparametric Density Estimation

A more general model structure for local densities is to define the density at any point **x** as being proportional to a weighted sum of all points in the training data set, where the weights are defined by an appropriately chosen kernel function. For the one-dimensional case we have

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} w_i, \quad w_i = K\left(\frac{x - x(i)}{h}\right),$$

where f(x) is the kernel density estimate at a query point x, K(t) is the kernel function (for example, K(t) = 1 - |t|, t = 1; K(t) = 0 otherwise) and h is the bandwidth of the kernel. Intuitively, the density at x is proportional to the sum of weights evaluated at x, which in turn depend on the proximity of the n points in the training data to x. As with nonparametric regression, the model is not defined explicitly, but is determined implicitly by the data and the kernel function. The approach is "memory-based" in the sense that all of the data points are retained in the model; that is, no summarization occurs. For very large data sets of course this may be impractical from a computational and storage viewpoint.

In one dimension, the kernel function K is usually chosen as a smooth unimodal function (such as a Normal or triangular distribution) that integrates to 1; the precise shape is typically not critical. As in regression, the bandwidth h plays the role of determining how smooth the model is. If h is relatively large, then the kernel is relatively wide so that many points receive significant weight in the sum and the estimate of the density is very smooth. If h is relatively small, the kernel estimate is determined by the small number of points that are close to x, and the estimate of the density is more sensitive locally to the data (more "spiky" in appearance). Estimating a good value of h in practice can be somewhat problematic. There is no single objective methodology for finding the bandwidth h that has wide acceptance. Techniques based on cross-validation can be useful but are typically computationally complex and not always reliable. Simple "eyeballing" of the resulting density along specific dimensions is always recommended to check whether or not the chosen values for h appear reasonable.

Under appropriate assumptions these kernel models are flexible enough to approximate any smooth density function, if h is chosen appropriately, which adds to their appeal. However, this approximation result holds in the limit as we get an infinite number of data points, making it somewhat less relevant for the finite data sets we see in practice. Nonetheless, kernel models can be very valuable for low-dimensional problems as a way to determine structure in the data (such as local peaks or gaps) in a manner that might not otherwise be visible.

Density estimation with kernel models becomes much more difficult as p increases. To begin with, we now need to define a p-dimensional kernel function. A popular choice is to define the p-dimensional kernel as a product of one-dimensional kernels, each with its own bandwidth, which keeps the number of parameters (the bandwidths $h_1, ..., h_p$ for each dimension) linear in the number of dimensions. A less obvious problem is the fact that in high dimensions it is natural for points to be farther away from each other than we might expect intuitively (the "curse of dimensionality" again). In fact, if we want to keep our approximation error constant as p increases, the number of data points we need grows exponentially with p. This is rather unfortunate and means in practice that kernel models are really practical only for relatively low-dimensional problems.

NOTES

Kernel methods are often complex to implement for large data sets. Unless the kernel function K(t) has compact support (that is, unless it is zero outside some finite range on t) then calculating the kernel estimate $f(x)$ at some point x potentially involves summing over contributions from all n data points in the database. In practice of course since most of these contributions will be negligible (that is, will be in the tails of the kernel) there are various ways to speed up this calculation. Nonetheless, this "memory -based" representation can be a relatively complex one to store and compute with (it can be O(n) to compute the density at just one query data point).

9.2.7 Joint Distributions for Categorical Data

First there is the problem of how to estimate such a large number of probabilities. As an example, let $\{p_1 \ldots p_{m^p}\}$ represent a list of all the joint probability terms in the unknown distribution we are trying to estimate from a data set with n p-dimensional observations. Hence, we can think of m^p different "cells," $\{c_1 \ldots c_{m^p}\}$ each containing n_i observations, $1 \le i \le m^p$. The expected number of data points in celli, given a random sample from $p(\mathbf{x})$ of size n, can be written as $E_{p(\mathbf{x})}[n_i] = np_i$. Assuming (for example) that $p(\mathbf{x})$ is approximately uniform (that is, $p_i \approx 1/m^p$) we get that

$$E_{p(\mathbf{x})}[n_i] \approx \frac{n}{m^p}.$$

Thus, for example, if $n < 0.5m^p$, the expected number of data points falling in any given cell is closer to 0 than to 1. Furthermore, if we use straightforward frequency counts as our method for estimating probabilities, we will estimate $\hat{p}_i = 0$ for each empty cell, whether or not $p_i = 0$ in truth. Note that if $p(\mathbf{x})$ is nonuniform the problem is actually worse since there will be more cells with smaller p_i (that is, less chance of any data falling in them). The fundamental problem here is the m^p exponential growth in the number of cells. With p = 20 binary variables (m = 2) we get $m^p \approx 10^6$. By doubling the number of variables to p = 40 we now get $m^p \approx 10^{12}$. Say that we had n data points for the case of p = 20 and that we wanted to add some new variables to the analysis while still keeping the expected number of data points per cell to be constant (that is, the same as it was with n data points). If we added extra 20 variables to the problem we would need to increase the data set from n to n' = 10^6 n, an increase by a factor of a million.

A second practical problem is that even if we can reliably estimate a full joint distribution from data, it is exponential in both space and time to work with directly. A full joint distribution will have a O(m^p) memory requirement; for example, O (10^{12}) real-valued probabilities would need to be stored for a full distribution on 40 binary variables. Furthermore, many computations using this distribution will also scale exponentially. Let the variables be {X_1, ..., X_p}, each taking m values. If we wanted to determine the marginal distribution on any single variable X_j (say), we could calculate it as

$$p(x_j) = \sum_{X_1,\ldots,X_{j-1},X_{j+1},\ldots,X_p} p(X_1,\ldots,X_{j-1},x_j,X_{j+1}\ldots,X_p),$$

that is, by summing over all the other variables in the distribution. The sum on the right involves O(m^{p-1}) summations for example, O (10^{39}) summations when p = 40 and m = 2. Clearly this sort of exercise is intractable except for relatively small values of m and p.

The practical consequence is that we can only reliably estimate and work with full joint distributions for relatively low-dimensional problems. Although our examples were for categorical data, essentially the same problems also arise of course for ordered or real valued data.

9.3 Background on Cluster Analysis

We now move beyond probability density and distribution models to focus on the related descriptive data mining task of cluster analysis that is, decomposing or partitioning a (usually multivariate) data set into groups so that the points in one group are similar to each other and are as different as possible from the points in other groups. Although the same techniques may often be applied, we should distinguish between two different objectives. In one, which we might call segmentation or dissection, the aim is simply to partition the data in a way that is convenient. "Convenient" here might refer to administrative convenience, practical convenience, or any other kind. For example, a manufacturer of shirts might want to choose jus t a few sizes and shapes so as to maximize coverage of the male population. He or she will have to choose those sizes in terms of collar size, chest size, arm length, and so on, so that no man has a shape too different from that of a well-fitting shirt. To do this, he or she will partition the population of men into a few groups in terms of the variables collar, chest, and arm length. Shirts of one size will then be made for each group.

In contrast to this, we might want to see whether a sample of data is composed of natural subclasses. For example, whiskies can be characterized in terms of color, nose, body, palate, and finish, and we might want to see whether they fall into distinct classes in terms of these variables. Here we are not partitioning the data for practical convenience, but rather are hoping to discover something about the nature of the sample or the population from which it arose to discover whether the overall population is, in fact, heterogeneous.

Technically, this second exercise is what cluster analysis seeks to do to see whether the data fall into distinct groups, with members within each group being similar to other members in that group but different from members of other groups. However, the term "cluster analysis" is often used in general to describe both segmentation and cluster analysis problems (and we shall also be a little lax in this regard). In each case the aim is to split the data into classes, so perhaps this is not too serious a misuse. It is resolved, as we shall see below, by the fact that there is a huge number of different algorithms for partitioning data in this way. The important thing is to match our method with our objective. This way, mistakes will not arise, whatever we call the activity.

It will be obvious from this that such methods (cluster and dissection techniques) hinge on the notion of distance. In order to decide whether a set of points can be split into subgroups, with members of a group being closer to other members of their group than to members of other groups, we need to say what we mean by "closer to." The notion of "distance," and different measures of it. Any of the measures described there, or indeed any other distance measure, can be used as the basis for a cluster or dissection analysis. As far as these techniques are concerned, the concept of distance is more fundamental than the coordinates of the points. In principle, to carry out a cluster analysis all we need to know is the set of interpoint distances, and not the values on any variables. However, some methods make use of "central points" of clusters, and so require that the raw coordinates be available.

NOTES

Cluster analysis has been the focus of a huge amount of research effort, going back for several decades, so that the literature is now vast. It is also scattered. Considerable portions of it exist in the statistical and machine learning literatures, but other many other publications on cluster analysis may be found elsewhere. One of the problems is that new methods are constantly being developed, sometimes without an awareness of what has already been developed. More seriously, a proper understanding of their properties and the way they behave with different kinds of data is available for very few of the methods. One of the reasons for this is that it is difficult to tell whether a cluster analysis has been successful. Contrast this with predictive modeling, in which we can take a test data set and see how accurately the value of the target variable is predicted in this set. For a clustering problem, unfortunately, there is no direct notion of generalization to a test data set, although, as we will see in our discussion of probabilistic clustering, it is possible in some situations to pose the question of whether or not the cluster structure discovered in the training data is genuinely present in the underlying population. Generally speaking, however, the validity of a clustering is often in the eye of the beholder; for example, if a cluster produces an interesting scientific insight, we can judge it to be useful. Quantifying this precisely is difficult, if not impossible, since the interpretation of how interesting a clustering is will inevitably be application-dependent and subjective to some degree.

As we shall see in the next few sections, different methods of cluster analysis are effective at detecting different kinds of clusters, and we should consider this when we choose a particular algorithm. That is, we should consider what we mean or intend to mean by a "cluster." In effect, different clustering algorithms will be biased toward finding different types of cluster structures (or "shapes") in the data, and it is not always easy to ascertain precisely what this bias is from the description of the clustering algorithm.

To illustrate, we might take a "cluster" as being a collection of points such that the maximum distance between all pairs of points in the cluster is as small as possible. Then each point will be similar to each other point in the cluster. An algorithm will be chosen that seeks to partition the data so as to minimize this maximum interpoint distance (more on this below). We would clearly expect such a method to produce compact, roughly spherical, clusters. On the other hand, we might take a "cluster" as being a collection of points such that each point is as close as possible to some other member of the cluster although not necessarily to all other members. Clusters discovered by this approach need not be compact or roughly spherical, but could have long (and not necessarily straight) sausage shapes. The first approach would simply fail to pick up such clusters. The first approach would be appropriate in a segmentation situation, while the second would be appropriate if the objects within each hypothesized group were measured at different stages of some evolutionary process. For example, in a cluster analysis of people suffering from some illness, to see whether there were different subtypes, we might want to allow for the possibility that the patients had been measured at different stages of the disease, so that they had different symptom patterns even though they belonged to the same subtype.

The important lesson to be learned from this is that we must match the method to the objectives. In particular, we must adopt a cluster analytic tool that is effective at detecting clusters that conform to the definition of what is meant by "cluster" in the problem at hand. It is perhaps worth adding that we should not be too rigid about it. Data mining, after all, is about discovering the unexpected, so we must not be too determined in imposing our preconceptions on the analysis. Perhaps a search for a different kind of cluster structure will throw up things we have not previously thought of.

Broadly speaking, we can identify three different general types of cluster analysis algorithms: those based on an attempt to find the optimal partition into a specified number of clusters, those based on a hierarchical attempt to discover cluster structure, and those based on a probabilistic model for the underlying clusters. We discuss each of these in turn in the next three sections.

9.4 Partition-Based Clustering Algorithms

In data mining algorithms can often be conveniently thought of in five parts: the task, the model, the score function, the search method, and the data management technique. In partition -based clustering the task is to partition a data set into k disjoint sets of points such that the points within each set are as homogeneous as possible, that is, given the set of n data points D = {$\mathbf{x}(1)$, ..., $\mathbf{x}(n)$}, our task is to find K clusters C = {C_1, ..., C_K} such that each data point $\mathbf{x}(i)$ is assigned to a unique cluster C_k. Homogeneity is captured by an appropriate score function (as discussed below), such as minimizing the distance between each point and the centroid of the cluster to which it is assigned. Partition-based

NOTES

clustering typically places more emphasis on the score function than on any formal notion of a model. Often the centroid or average of the points belonging to a cluster is considered to be a representative point for that cluster, and there is no explicit statement of what sort of shape of cluster is being sought. For cluster representations based on the notion of a single "center" for each cluster, however, the boundaries between clusters will be implicitly defined. For example, if a point **x** is assigned to a cluster according to which cluster center is closest in a Euclidean-distance sense, then we will get linear boundaries between the clusters in **x** space.

9.4.1 Score Functions for Partition-Based Clustering

A large number of different score functions can be used to measure the quality of clustering and a wide range of algorithms has been developed to search for an optimal (or at least a good) partition.

In order to define the clustering score function we need to have a notion of distance between input points. Denote by d(**x, y**) the distance between points **x, y** Є D, and assume for simplicity that the function d defines a metric on D. Most score functions for clustering stress two aspects: clusters should be compact, and clusters should be as far from each other as possible. A straightforward formulation of these intuitive notions is to look at within cluster variation wc (C) and between cluster variation bc(C) of a clustering C. The within cluster variation measures how compact or tight the clusters are, while the between cluster variation looks at the distances between different clusters.

9.4.2 Basic Algorithms for Partition-Based Clustering

In principle, at least, the problem is straightforward. We simply search through the space of possible assignments C of points to clusters to find the one that minimizes the score (or maximizes it, depending on the chosen score function).

The nature of the search problem can be thought of as a form of combinatorial optimization, since we are searching for the allocation of n objects into K classes that maximizes (or minimizes) our chosen score function. The number of possible allocations (different clusterings of the data) is approximately K^n. For example, there are some $2^{100} \approx 10^{10}$ possible allocations of 100

NOTES

objects into two classes. Thus, as we have seen with other data mining problems, direct exhaustive search methods are certainly not applicable unless we are dealing with tiny data sets. Nonetheless, for some clustering score functions, methods have been developed that permit exhaustive coverage of all possible clusterings without actually carrying out an exhaustive search. These include branch and bound methods, which eliminate potential clusterings on the grounds that they have poorer scores than alternatives already found, without actually evaluating the scores for the potential clusterings. Such methods, while extending the range over which exhaustive evaluation can be made, still break down for even moderately -sized data sets. For this reason, we do not examine them further here.

Unfortunately, neither do there exist closed-form solutions for any score function of interest; that is, there is usually no direct method for finding a specific clustering C that optimizes the score function. Thus, since closed form solutions do not exist and exhaustive search is infeasible, we must resort to some form of systematic search of the space of possible clusters. It is important to emphasize that given a particular score function, the problem of clustering has been reduced to an optimization problem, and thus there are a large variety of choices available in the optimization literature that are potentially applicable.

Iterative improvement algorithms based on local search are particularly popular for cluster analysis. The general idea is to start with a randomly chosen clustering of the points, then to reassign points so as to give the greatest increase (or decrease) in the score function, then to recalculate the updated cluster centers, to reassign points again, and so forth until there is no change in the score function or in the cluster memberships. This greedy approach has the virtue of being simple and guaranteeing at least a local maximum (minimum) of the score function. Of course it suffers the usual drawback of greedy search algorithms in that we do not know how good the clustering C that it converges to is relative to the best possible clustering of the data (the global optimum for the score function being used).

Here we describe one well-known example of this general approach, namely, the K-means algorithm. The number K of clusters is fixed before the algorithm is run (this is typical of many clustering algorithms). There are several variants of the K-means algorithm. The basic version begins by randomly picking K cluster centers, assigning each point to the cluster whose mean is closest

NOTES

in a Euclidean distance sense, then computing the mean vectors of the points assigned to each cluster, and using these as new centers in an iterative approach. As an algorithm, the method is as follows: assuming we have n data points D = {$\mathbf{x}_1$, ..., $\mathbf{x}_n$}, our task is to find K clusters {C_1...,C_K}:

for k = 1, ..., K let **r**(k) be a randomly chosen point from D;
while changes in clusters C_k happen **do**
    form clusters:
    for k = 1, ..., K **do**
        $C_k = \{\mathbf{x} \in D \mid d(\mathbf{r}_k, \mathbf{x}) = d(\mathbf{r}_j, \mathbf{x})$ for all j = 1, ..., K,j $\neq$ k
    };
    end;
compute new cluster centers:
    for k = 1, ..., K **do**
    $\mathbf{r}_k$ = the vector mean of the points in C_k
    end;
end;

The complexity of the K-means algorithm is O(KnI), where I is the number of iterations. Namely, given the current cluster centers $\mathbf{r}_k$, we can in one pass through the data compute all the K_n distances d($\mathbf{r}_k$, **x**) and for each **x** select the minimal one; then computing the new cluster centers can also be done in time O(n).

A variation of this algorithm is to examine each point in turn and update the cluster centers whenever a point is reassigned, repeatedly cycling through the points until the solution does not change. If the data set is very large, we can simply add in each data point, without the recycling. Further extensions (for example, the ISODATA algorithm) include splitting and/or merging clusters. Note that there are a large number of different partition -based clustering algorithms, many of which hinge around adding or removing one point at a time from a cluster. Efficient updating formula been developed in the context of evaluating the change incurred in a score function by moving one data point in or out of a cluster in particular, for all of the score functions involving **W** discussed in the last section.

The search in the K-means algorithm is restricted to a small part of the space of possible partitions. It is possible that a good cluster solution will be missed due to the algorithm converging to a local rather than global minimum of the score function. One way to alleviate (if not solve) this problem is to carry out multiple searches from different randomly chosen starting points for the cluster centers. We can even take this further and adopt a simulated annealing strategy to try to avoid getting trapped in local minima of the score function.

Since cluster analysis is essentially a problem of searching over a huge space of potential solutions to find whatever optimizes a specified score function, it is no surprise that various kinds of mathematical programming methods have been applied to this problem. These include linear programming, dynamic programming, and linear and nonlinear integer programming.

Clustering methods are often applied on large data sets. If the number of observations is so large that standard algorithms are not tractable, we can try to compress the data set by replacing groups of objects by succinct representations. For example, if 100 observations are very close to each other in a metric space, we can replace them with a weighted observation located at the centroid of those observations and having an additional feature (the radius of the group of points that is represented). It is relatively straightforward to modify some of the clustering algorithms to operate on such "condensed" representations.

9.5 Hierarchical Clustering

Whereas partition-based methods of cluster analysis begin with a specified number of clusters and search through possible allocations of points to clusters to find an allocation that optimizes some clustering score function, hierarchical methods gradually merge points or divide super clusters. In fact, on this basis we can identify two distinct types of hierarchical methods: the agglomerative (which merge) and the divisive (which divide). The agglomerative are the more important and widely used of the two. Note that hierarchical methods can be viewed as a specific (and particularly straightforward) way to reduce the size of the search.

A notable feature of hierarchical clustering is that it is difficult to separate the model from the score function and the search method used to determine the best clustering. Because of this, in this section we will focus on clustering algorithms directly. We can consider the final hierarchy to be a model, as a hierarchical mapping of data points to clusters; however, the nature of this model (that is, the cluster "shape") is implicit in the algorithm rather than being explicitly represented. Similarly for the score function, there is no notion of an explicit global score function. Instead,

NOTES

various local scores are calculated for pairs of leaves in the tree (that is, pairs of clusters for a particular hierarchical clustering of the data) to determine which pair of clusters are the best candidates for agglomeration (merging) or dividing (splitting). Note that as with the global score functions used for partition-based clustering, different local score functions can lead to very different final clusterings of the data.

Hierarchical methods of cluster analysis permit a convenient graphical display, in which the entire sequence of merging (or splitting) of clusters is shown. Because of its tree-like nature, such a display is called a dendrogram. We illustrate in an example below.

Cluster analysis is particularly useful when there are more than two variables: if there are only two, then we can eyeball a scatterplot to look for structure. However, to illustrate the ideas on a data set where we can see what is going on, we will apply a hierarchical method to some two dimensional data. Figure 9.3 shows a scatterplot of the two dimensional data. The vertical axis is the time between eruptions and the horizontal axis is the length of the following eruption, both measured in minutes. The points are given numbers in this plot merely so that we can relate them to the dendrogram in this exposition, and have no other substantive significance.

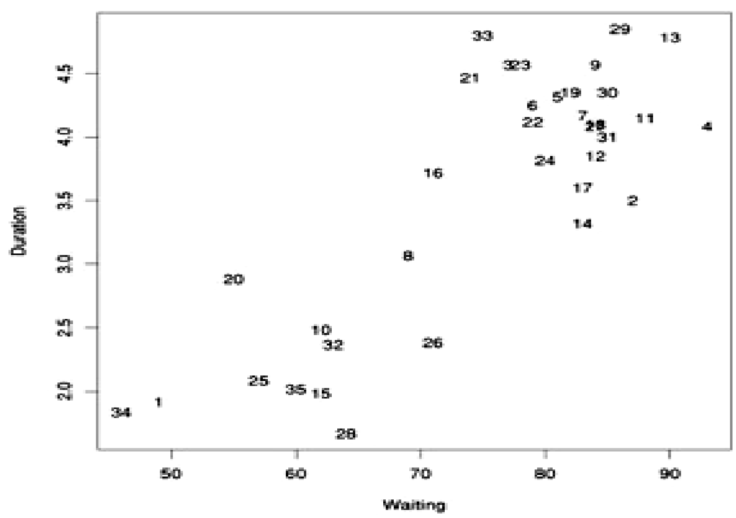


Figure 9.3: Duration of eruptions versus waiting time between eruptions (in minutes) for the old faithful geyser in Yellowstone Park.

NOTES

As an example, figure 9.4 shows the dendrogram that results from agglomerative merging the two clusters that leads to the smallest increase in within-cluster sum of squares. The height of the crossbars in the dendrogram (where branches merge) shows values of this score function. Thus, initially, the smallest increase is obtained by merging points 18 and 27, and from figure 9.3 we can see that these are indeed very close (in fact, the closest). Note that closeness from a visual perspective is distorted because of the fact that the x-scale is in fact compressed on the page relative to the y-scale. The next merger comes from merging points 6 and 22. After a few more mergers of individual pairs of neighboring points, point 12 is merged with the cluster consisting of the two points 18 and 27, this being the merger that leads to least increase in the clustering criterion. This procedure continues until the final merger, which is of two large clusters of points. This structure is evident from the dendrogram. (It need not always be like this. Sometimes the final merger is of a large cluster with one single outlying point as we shall see below.) The hierarchical structure displayed in the dendrogram also makes it clear that we could terminate the process at other points. This would be equivalent to making a horizontal cut through the dendrogram at some other level, and would yield a different number of clusters.

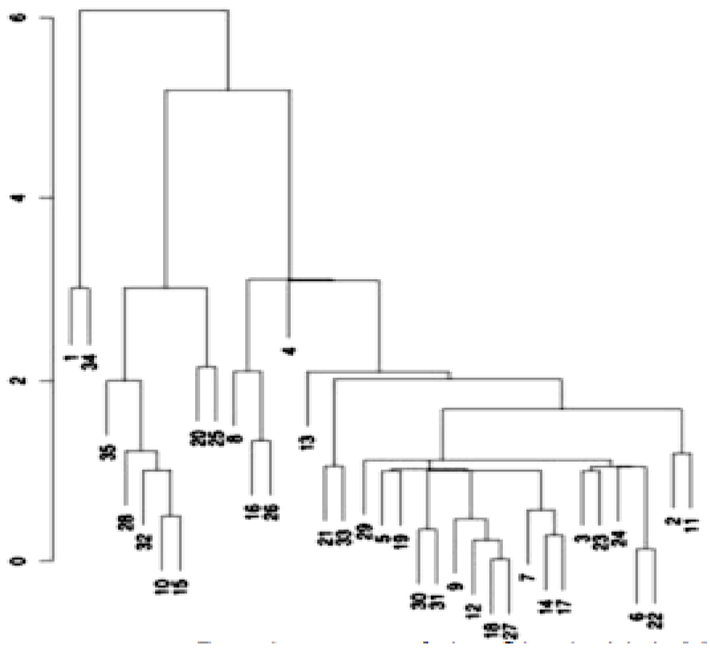


Figure 9.4: Dendrogram resulting from clustering of data in figure 9.3 using the criterion of merging clusters that leads to the smallest increase in the total sum of squared errors.

9.5.1 Agglomerative Methods

Agglomerative methods are based on measures of distance between clusters. Essentially, given an initial clustering, they merge those two clusters that are nearest, to form a reduced number of clusters. This is repeated, each time merging the two closest clusters, until just one cluster, of all the data points, exists. Usually the starting point for the process is the initial clustering in which each cluster consists of a single data point, so that the procedure begins with the n points to be clustered.

Assume we are given n data points D = {**x**(1), ..., **x**(n)}, and a function $D(C_i, C_j)$ for measuring the distance between two clusters C_i and C_j. Then an agglomerative algorithm for clustering can be described as follows:

for i = 1, ..., n let $C_i = \{\mathbf{x}(i)\}$;
while there is more than one cluster left **do**
    let C_i and C_j be the clusters minimizing the distance $D(C_k, C_h)$ between any two clusters;
    $C_i = C_i \cup C_j$;
    remove cluster C_j;
end;

What is the time complexity of this method? In the beginning there are n clusters, and in the end 1; thus there are n iterations of the main loop. In iteration i we have to find the closest pair of clusters among n - i + 1 clusters. We will see shortly that there are a variety of methods for defining the intercluster distance $D(C_i, C_j)$. All of them, however, require in the first iteration that we locate the closest pair of objects. This takes $O(n^2)$ time, unless we have special knowledge about the distance between objects and so, in most cases, the algorithm requires $O(n^2)$ time, and frequently much more. Note also that the space complexity of the method is also $O(n^2)$, since all pairwise distances between objects must be available at the start of the algorithm. Thus, the method is typically not feasible for large values of n. Furthermore, interpreting a large dendrogram can be quite difficult (just as interpreting a large classification tree can be difficult).

Note that in agglomerative clustering we need distances between individual data objects to begin the clustering, and during clustering we need to be able to compute distances between groups of data points (that is, distances between clusters). Thus, one advantage of this approach (over partition -based clustering, for example) is the fact that we do not need to have a vector representation for each object as long as we can compute distances between objects or between sets of objects. Thus, for example, agglomerative clustering provides a natural framework for clustering objects that are not easily summarized as vector measurements. A good example would be clustering of protein sequences where there exist several well-defined notions of distance such as the edit-distance between two sequences (that is, a measure of how many basic edit operations are required to transform one sequence into another).

In terms of the general case of distances between sets of objects (that is, clusters) many measures of distance have been proposed. If the objects are vectors then any of the global

However, local pairwise distance measures (that is, between pairs of clusters) are especially suited to hierarchical methods since they can be computed directly from pairwise distances of the members of each clus ter. One of the earliest and most important of these is the nearest neighbor or single link method. This defines the distance between two clusters as the distance between the two closest points, one from each cluster;

$$(\mathcal{D}_{sl}(C_i, C_j) = \min_{\mathbf{x},\mathbf{y}}\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\},$$

where d(**x, y**) is the distance between objects **x** and **y**. The single link method is susceptible (which may be a good or bad thing, depending upon our objectives) to the phenomenon of "chaining," in which long strings of points are assigned to the same cluster (contrast this with the production of compact spherical clusters). This means that the single link method is of limited value for segmentation. It also means that the method is sensitive to small perturbations of the data and to outlying points (which, again, may be good or bad, depending upon what we are trying to do). The single link method also has the property (for which it is unique no other measure of distance between clusters possesses it) that if two pairs of clusters are equidistant it does not matter which is merged first. The overall result will be the same, regardless of the order of merger.

The dendrogram from the single link method applied to the data in figure 9.3 is shown in figure 9.5. Although on this particular data set

NOTES

the results of single link clustering and that of figure 9.4 are quite similar, the two methods can in general produce quite different results.

Figure 9.5: Dendrogram of the single link method applied to the data in figure 9.3.

At the other extreme from single link, furthest neighbor, or complete link , takes as the distance between two clusters the distance between the two most distant points, one from each cluster:

$$\mathcal{D}_{fl}(C_i, C_j) = \max_{\mathbf{x},\mathbf{y}}\{d(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\},$$

where d(**x, y**) is again the distance between objects **x** and **y**. For vector objects this imposes a tendency for the groups to be of equal size in terms of the volume of space occupied (and not in terms of numbers of points), making this measure particularly appropriate for segmentation problems.

Other important measures, intermediate between single link and complete link, include (for vector objects) the centroid measure (the distance between two clusters is the distance between their centroids), the group average measure (the distance between two clusters is the average of all the distances between pairs of points, one from each cluster), and Ward's measure for vector data (the distance between two clusters is the difference between the total within cluster sum of squares for the two clusters separately, and the within cluster sum of squares resulting from merging the two clusters discussed above). Each such measure has slightly different properties and other variants also exist; for example, the median measure for vector data ignores the size of clusters, taking the "center" of a combination of two clusters to be the midpoint of the line joining the centers of the two components. Since we are seeking the novel in data mining, it may well be worthwhile to experiment with several measures, in case we throw up something unusual and interesting.

9.5.2 Divisive Methods

Just as stepwise methods of variable selection can start with no variables and gradually add variables according to which lead to most improvement (analogous to agglomerative cluster analysis methods), so they can also start with all the variables and gradually remove those whose removal leads to least deterioration in the model. This second approach is analogous to divisive methods of cluster analysis. Divisive methods begin with a single cluster composed of all of the data points, and seek to split this into components. These further components are then split, and the process is taken as far as necessary. Ultimately, of course, the process will end with a partition in which each cluster consists of a single point.

Monothetic divisive methods split clusters using one variable at a time. This is a convenient (though restrictive) way to limit the number of possible partitions that must be examined. It has the attraction that the result is easily described by the dendrogram the split at each node is defined in terms of just a single variable. The term association analysis is sometimes uses to describe monothetic divisive procedures applied to multivari ate binary data.

Polythetic divisive methods make splits on the basis of all of the variables together. Any intercluster distance measure can be used. The difficulty comes in deciding how to choose potential allocations to clusters that is, how to restrict the search through the space of possible partitions. In one approach, objects are examined one at a time, and that one is selected for transfer from a main cluster to a subcluster that leads to the greatest improvement in the clustering score.

NOTES

In general, divisive methods are more computationally intensive and tend to be less widely used than agglomerative methods.

9.6 Probabilistic Model-Based Clustering using Mixture Models

This is often referred to as probabilistic model-based clustering since there is an assumed probability model for each component cluster. In this framework it is assumed that the data come from a multivariate finite mixture model of the general form

$$f(x) = \sum_{k=1}^{K} \pi_k f_k(x; \theta_k)$$

where f_k are the component distributions. Roughly speaking, the general procedure is as follows: given a data set D = {**x**(1), ..., **x**(n)}, determine how many clusters K we want to fit to the data, choose parametric models for each of these K clusters (for example, multivariate Normal distributions are a common choice), and then use the EM algorithm to determine the component parameters θ_k and component probabilities π_k from the data. (We can of course also try to determine a good value of K from the data, we will return to this question later in this section.) Typically the likelihood of the data (given the mixture model) is used as the score function, although other criteria (such as the so-called classification likelihood) can also be used. Once a mixture decomposition has been found, the data can then be assigned to clusters for example, by assigning each point to the cluster from which it is most likely to have come.

The advantages come at a certain cost. The main "cost" is the assumption of a parametric model for each component; for many problems it may be difficult a priori to know what distributional forms to assume. Thus, model-based probabilistic clustering is really only useful when we have reason to believe that the distributional forms are appropriate. For our red blood cell data, we can see by visual inspection that the normal assumptions are quite reasonable. Furthermore, since the two measurements consist of estimated means from large samples of blood cells, basic statistical theory also suggests that a normal distribution is likely to be quite appropriate.

The other main disadvantage of the probabilistic approach is the complexity of the associated estimation algorithm. Consider the

NOTES

difference between EM and K-means. We can think of K-means as a stepwise approximation to the EM algorithm applied to a mixture model with Normal mixture components (where the covariance matrices for each cluster are all assumed to be the identity matrix). However, rather than waiting until convergence is complete before assigning the points to the clusters; the K-means algorithm reassigns them at each step.

The probabilistic clustering, consider the problem of finding the best value for K from the data. Note that as K (the number of clusters) is increased, the value of the likelihood at its maximum cannot decrease as a function of K. Thus, likelihood alone cannot tell us directly about which of the models, as a function of K, is closest to the true data generating process. Moreover, the usual approach of hypothesis testing (for example, testing the hypothesis of one component versus two, two versus three, and so forth) does not work for technical reasons related to the mixture likelihood. However, a variety of other ingenious schemes have been developed based to a large extent on approximations of theoretical analyses. We can identify three general classes of techniques in relatively widespread use:

Penalized Likelihood: Subtract a term from the maximizing value of the likelihood. The BIC (Bayesian (Bayesian Information Criterion) is widely used. Here

$$S_{BIC}(M_K) = 2S_L(\hat{\theta}_K; M_K) + d_K \log n$$

where $S_L(\theta_K ; M_K)$ is the minimizing value of the negative log - likelihood and d_K is the number of parameters, both for a mixture model with K components. This is evaluated from K = 1 up to some K_{max} and the minimum taken as the most likely value of K. The original derivation of BIC was based on asymptotic arguments in a different (regression) context, arguments that do not strictly hold for mixture modeling. Nonetheless, the technique has been found to work quite well in practice and has the merit of being relatively cheap to compute relative to the other methods listed below. In figure 9.6 the negative of the BIC score function is plotted for the red blood cell data and points to K = 2 as the best model (recall that there is independent medical knowledge that the data belong to two groups here, so this result is quite satisfying). There are a variety of other proposals for penalty terms, but BIC appears to be the most widely used in the clustering context.

NOTES

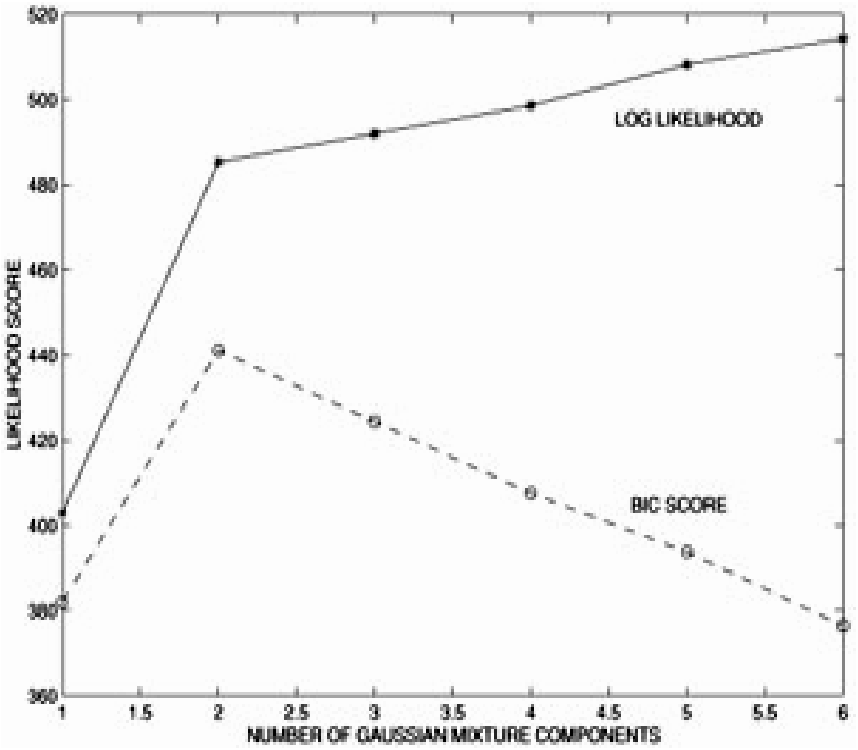


Figure 9.6: Log-Likelihood and BIC Score as a Function of the Number of Normal Components Fitted to the Red Blood Cell Data of Figure 9.11.

Resampling Techniques: We can use either bootstrap methods or cross- validated likelihood using resampling ideas as another approach to generate "honest" estimates of which K value is best. These techniques have the drawback of requiring significantly more computation than BIC for example, ten times more for the application of ten -fold cross-validation. However, they do provide a more direct assessment of the quality of the models, avoiding the need for the assumptions associated with methods such as BIC.

Bayesian Approximations: The fully Bayesian solution to the problem is to estimate a distribution $p(K|D)$, that is, the probability of each K value given the data, where all uncertainty about the parameters is integrated out in the usual fashion. In practice, of course, this integration is intractable (recall that we are integrating in a d_K -dimensional space) so various approximations are sought. Both analytic approximations (for example, the Laplace approximation about the mode of the posterior distribution) and sampling techniques (such as Markov chain Monte Carlo) are used. For large data sets with many parameters in the model, sampling techniques may be computationally impractical, so analytic approximation methods tend to be more widely used. For example, the AUTOCLASS algorithm of Cheese man and Stutz (1996) for clustering with mixture models uses a specific analytic approximation of the posterior distribution for model selection. The BIC penalty-based score function can also be viewed as an approximation to the full Bayesian approach.

In a sense, the formal probabilistic modeling implicit in mixture decomposition is more general than cluster analysis. Cluster analysis aims to produce merely a partition of the available data, whereas mixture decomposition produces a description of the distribution underlying the data (that this distribution is composed of a number of components). Once these component probability distributions have been identified, points in the data set can be assigned to clusters on the basis of the component that is most likely to have generated them. We can also look at another way: the aim of cluster analysis is to divide the data into naturally occurring regions in which the points are closely or densely clustered, so that there are relatively sparse regions between the clusters. From a probability density perspective, this will correspond to regions of high density separated by valleys of low density, so that the probability density function is fundamentally multimodal. However, mixture distributions, even though they are composed of several components, can well be unimodal.

Consider the case of a two-component univariate normal mixture. Clearly, if the means are equal, then this will be unimodal. In fact, a sufficient condition for the mixture to be unimodal (for all values of the mixing proportions) when the means are different is $| \mu_1 - \mu_2 | = 2 \min(\sigma_1, \sigma_2)$. Furthermore, for every choice of values of the means and standard deviations in a two-component normal mixture there exist values of the mixing proportions for which the mixture is unimodal. This means that if the means are close enough there will be just one cluster, even though there are two components. We can still use the mixture decomposition to induce a clustering, by assigning each data point to the cluster from which it is most likely to have come, but this is unlikely to be a useful clustering.

NOTES

10. Predictive Modeling for Classification

Structure

Objective

After going through this lesson, you should be able to:

- Discuss about a classification modeling;
- Discuss the perceptron and linear discriminants;
- Discuss different predictive models;

10.1 A Brief Overview of Predictive Modeling

Predictive modeling can be thought of as learning a mapping from an input set of vector measurements **x** to a scalar output y (we can learn mappings to vector outputs, but the scalar case is much more common in practice). In predictive modeling the training data D_{train} consists of pairs of measurements, each consisting of a vector **x**(i) with a corresponding "target" value y(i), $1 \le i \le n$. Thus the goal of predictive modeling is to estimate (from the training data) a mapping or a function $y = f(\mathbf{x}; \theta)$ that can predict a value y given an input vector of measured values **x** and a set of estimated parameters θ for the model f. f is the functional form of the model structure, the θs are the unknown parameters within f whose values we will determine by minimizing a suitable score function on the data, and the process of searching for the best θ values is the basis for the actual data mining algorithm. We thus need to choose three things: a particular model structure (or a family of model structures), a score function, and an optimization strategy for finding the best parameters and model within the model family.

In data mining problems, since we typically know very little about the functional form of $f(\mathbf{x}; \theta)$ ahead of time, there may be attractions in adopting fairly flexible functional forms or models for f. On the other hand, simpler models have the advantage of often being more stable and more interpretable, as well as often providing the functional components for more complex model structures. For predictive modeling, the score function is usually relatively straightforward to define, typically a function of the difference between the prediction of the model $\hat{y}(i) = f(x(i); \theta)$ and the true value y(i) that is,

$$S(\theta) = \sum_{D_{train}} d\Big(y(i), \hat{y}(i)\Big) = \sum_{D_{train}} d\Big(y(i), f(\mathbf{x}(i); \theta)\Big)$$

where the sum is taken over the tuples (**x**(i), y(i)) in the training data set D_{train} and the function d defines a scalar distance such as squared error for real -valued y or an indicator function for categorical y. The actual heart of the data mining algorithm then involves minimizing S as a function of θ; the details of this are determined both by the nature of the distance function and by the functional form of $f(\mathbf{x}; \theta)$ that jointly determine how S depends on θ.

To compare predictive models we need to estimate their performance on "out-of-sample data" data that have not been used in constructing the models (or else, as discussed earlier, the performance estimates are likely to be biased). In this case we can redefine the score function S(?) so that it is estimated on a validation data set, or via cross- validation, or using a penalized score function, rather than on the training data directly).

10.2 Introduction to Classification Modeling

Here we briefly review some of the basic concepts. In classification we wish to learn a mapping from a vector of measurements **x** to a categorical variable Y. The variable to be predicted is typically called the class variable (for obvious reasons), and for convenience of notation we will use the variable C, taking values in the set $\{c_1, ..., c_m\}$ to denote this class variable for the rest of this chapter (instead of using Y). The observed or measured variables $X_1, ..., X_p$ are variously referred to as the features, attributes, explanatory variables, input variables, and so on the generic term input variable will be used throughout this chapter. We will refer to **x** as a p-dimensional vector (that is, we take it to be comprised of p variables), where each component can be real-valued, ordinal, categorical, and so forth. $x_j(i)$ is the jth component of the ith input vector, where 1 = i = n, 1 = j = p. In our introductory discussion we will implicitly assume that we are using the so-called "0–1" loss function, where a correct prediction incurs a loss of 0 and an incorrect class prediction incurs a loss of 1 irrespective of the true class and the predicted class.

We will begin by discussing two different but related general views of classification: the decision boundary (or discriminative) viewpoint, and the probabilistic viewpoint.

10.2.1 Discriminative Classification and Decision Boundaries

In the discriminative framework a classification model $f(\mathbf{x}; \theta)$ takes as input the measurements in the vector **x** and produces as output a symbol from the set $\{c_1, ..., c_m\}$. Consider the nature of the mapping function f for a simple problem with just two real- valued input variables X_1 and X_2. The mapping in effect produces a piecewise constant surface over the (X_1, X_2) plane; that is, only in certain regions does the surface take the value c_1. The union of all such regions where a c_1 is predicted is known as the decision region for class c_1; that is, if an input $\mathbf{x}(i)$ falls in this region its class will be predicted as c_1 (and the complement of this region is the decision region for all other classes). Knowing where these decision regions are located in the (X_1, X_2) plane is equivalent to knowing where the decision boundaries or decision surfaces are

NOTES

between the regions. Thus we can think of the problem of learning a classification function f as being equivalent to learning decision boundaries between the classes. In this context, we can begin to think of the mathematical forms we can use to describe decision boundaries, for example, straight lines or planes (linear boundaries), curved boundaries such as low- order polynomials, and other more exotic functions.

In most real classification problems the classes are not perfectly separable in the **X** space. That is, it is possible for members of more than one class to occur at some (perhaps all) values of **X** though the probability that members of each class occur at any given value **x** will be different. (It is the fact that these probabilities differ that permits us to make a classification. Broadly speaking, we assign a point **x** to the most probable class at **x**.) The fact that the classes "overlap" leads to another way of looking at classification problems. Instead of focusing on decision surfaces, we can seek a function $f(\mathbf{x};\ \theta)$ that maximizes some measure of separation between the classes. Such functions are termed discriminant functions. Indeed, the earliest formal approach to classification, Fisher's linear discriminant analysis method, was based on precisely this idea: it sought that linear combination of the variables in **x** that maximally discriminated between the (two) classes.

10.2.2 Probabilistic Models for Classification

Let $p(c_k)$ be the probability that a randomly chosen object or individual i comes from class c_k. Then $\sum_k p(c_k) = 1$, assuming that the classes are mutually exclusive and exhaustive. This may not always be the case for example, if a person had more than one disease (classes are not mutually exclusive) we might model the problem as set of multiple two -class classification problems ("disease 1 or not," "disease 2 or not," and so on). Or there might be a disease that is not in our classification model (the set of classes is not exhaustive), in which case we could add an extra class c_k+1 to the model to account for "all other diseases." Despite these potential practical complications, unless stated otherwise we will use the mutually exclusive and exhaustive assumption throughout this chapter since it is widely applicable in practice and provides the essential basis for probabilistic classification.

Imagine that there are two classes, males and females, and that $p(c_k)$, k = 1, 2, represents the probability that at conception a person receives the appropriate chromosomes to develop as male or female. The $p(c_k)$ are thus the probabilities that individual i belongs to class c_k if we have no other information (no measurements **x**(i)) at all. The $p(c_k)$ are sometime referred to as the class "prior probabilities," since they represent the probabilities of class membership before observing the vector **x**. Note that estimating the $p(c_k)$ from data is often relatively easy: if a random sample of the entire population has been drawn, the maximum likelihood estimate of $p(c_k)$ is just the frequency with which c_k occurs in the training data set. Of course, if other sampling schemes have been adopted, things may be more complicated. For example, in some medical situations it is common to sample equal numbers from each class deliberately, so that the priors have to be estimated by some other means.

Objects or individuals belonging to class k are assumed to have measurement vectors **x** distributed according to some distribution or density function $p(\mathbf{x}|c_k, \theta_k)$ where the θ_k are unknown parameters governing the characteristics of class c_k. For example, for multivariate real -valued data, the assumed model structure for the **x** for each class might be multivariate Normal, and the parameters θ_k would represent the mean (location) and variance (scale) characteristics for each class. the means are far enough apart, and the variances small enough, we can hope that the classes are relatively well separated in the input space, permitting classification with very low misclassification (or error) rate. The general problem arises when neither the functional form nor the parameters of the distributions of the **x**s are known a priori.

Once the $p(\mathbf{x}|c_k, \theta_k)$ distributions have been estimated, we can apply Bayes theorem to yield the posterior probabilities

$$p(c_k|\mathbf{x}) = \frac{p(\mathbf{x}|c_k, \theta)p(c_k)}{\sum_{l=1}^{m} p(\mathbf{x}|c_l, \theta_l)p(c_l)}, \quad 1 \le k \le m.$$

The posterior probabilities $p(c_k|\mathbf{x}, \theta_k)$ implicitly carve up the input space **x** into m decision regions with corresponding decision boundaries. For example, with two classes (m = 2) the decision boundaries will be located along the contours where $p(c_1|\mathbf{x}, \theta_1) = p(c_2|\mathbf{x}, \theta_2)$. Note that if we knew the true posterior class probabilities (instead of having to estimate them), we could make optimal predictions given a measurement vector **x**. For example, for the case in which all errors incur equal cost we should predict the class value c_k that has the hig hest posterior probability $p(c_k|\mathbf{x})$ (is most likely given the data) for any given **x** value. Note that this scheme is optimal in the sense that no other prediction method can do better (with the given variables **x**) it does not mean that it makes no errors. Ind eed, in most real problems the optimal classification scheme will have a nonzero error rate, arising from the overlap of the distributions p(**x**|c_k, θ_k). This overlap means that the maximum class probability p(c_k|**x**) < 1, so that there is a non-zero probability 1-p(c_k|**x**) of data arising from the other (less likely) classes at **x**, even though the optimal decision at **x** is to choose c_k. Extending this argument over the whole space, and averaging with respect to **x** (or summing over discrete-valued variables), the Bayes Error Rate is defined as

$$p_B^* = \int (1 - \max_k p(c_k|\mathbf{x}))p(\mathbf{x})d\mathbf{x}.$$

This is the minimum possible error rate. No other classifier can achieve a lower expected error rate on unseen new data. In practical terms, the Bayes error is a lower -bound on the best possible classifier for the problem.

Now consider a situation in which **x** is bivariate, and in which the members of one class are entirely surrounded by members of the other class. Here neither of the two X variables alone will lead to classification rules with zero error rate, but (provided an appropriate model was used) a rule based on both variables together could have zero error rate. Analogous situations, though seldom quite so extreme, often occur in practice: new variables add information, so that we can reduce the Bayes error rate by adding extra variables. While the Bayes error rate can only stay the same or decrease if we add more variables to the model, in fact we do not know the optimal classifier or the Bayes error rate. We have to estimate a classification rule from a finite set of training data. If the number of variables for a fixed number of training points is increased, the training data are representing the underlying distributions less and less accurately. The Bayes error rate may be decreasing, but we have a poorer approximation to it. At some point, as the number of variables increases, the paucity of our approximation overwhelms the reduction in Bayes error rate, and the rules begin to deteriorate.

The solution is to choose our variables with care; we need variables that, when taken together, separate the classes well. Finding appropriate variables (or a small number of features combinations of variables) is the key to effective classification. This is perhaps especially marked for complex and potentially very high

NOTES

dimensional data such as images, where it is generally acknowledged that finding the appropriate features can have a much greater impact on classification accuracy than the variability that may arise by choosing different classification models. One data-driven approach in this context is to use a score function such as cross-validated error rate to guide a search through combinations of features of course, for some classifiers this may be very computationally intensive, since the classifier may need to be retrained for each subset examined and the total number of such subsets is combinatorial in p (the number of variables).

10.2.3 Building Real Classifiers

While this framework provides insight from a theoretical viewpoint, it does not provide a prescriptive framework for classification modeling. That is, it does not tell us specifically how to construct classifiers unless we happen to know precisely the functional form of $p(\mathbf{x}|c_k)$ (which is rare in practice). We can list three fundamental approaches:

1. The discriminative approach:

Here we try to model the decision boundaries directly that is, a direct mapping from inputs **x** to one of m class label $c_1 \ldots c_m$. No direct attempt is made to model either the class-conditional or posterior class probabilities. Examples of this approach include perceptions and the more general support vector machines.

2. The regression approach:

The posterior class probabilities $p(c_k|\mathbf{x})$ are modeled explicitly, and for prediction the maximum of these probabilities (possibly weighted by a cost function) is chosen. The most widely used technique in this category is known as logistic regression. Note that decision trees can be considered under either the discriminative approach (if the tree only provides the predicted class at each leaf) or the regression approach (if in addition the tree provides the posterior class probability distribution at each leaf).

NOTES

3. The class-conditional approach:

Here, the class-conditional distributions p(**x**|c_k, θ_k) are modeled explicitly, and along with estimates of p(c_k) are inverted via Bayes rule to arrive at p(c_k|**x**) for each class c_k, a maximum is picked (possibly weighted by costs), and so forth, as in the regression approach. We can refer to this as a "generative" model in the sense that we are specifying (via p(**x**|c_k, θ_k)) precisely how the data are generated for each class. Classifiers using this approach are also sometimes referred to as "Bayesian" classifiers because of the use of Bayes theorem, but they are not necessarily Bayesian in the formal sense of Bayesian parameter estimation. In practice the parameter estimates used in equation p(c_k|**x**), $\hat{\theta}_k$, are often estimated via maximum likelihood for each class c_k, and "plugged in" to p(**x**|c_k, θ_k). There are Bayesian alternatives that average over θ_k. Furthermore, the functional form of p(**x**|c_k, θ_k) can be quite general any of parametric (for example, Normal), semi-parametric (for example, finite mixtures), or non-parametric (for example, kernels) can be used to estimate p(**x**|c_k, θ_k). In addition, in principle, different model structures can be used for each class c_k (for example, class c_1 could be modeled as a Normal density, class c_2 could be modeled as a mixture of exponentials, and class c_3 could be modeled via a kernel density estimate).

Note that both the discriminative and regression approaches focus on the differences between the classes (or, more formally, the focus is on the probabilities of class membership conditional on the values of **x**), whereas the class-conditional/generative approach focuses on the distributions of **x** for the classes. Methods that focus directly on the probabilities of class membership are sometimes referred to as diagnostic methods, while methods that focus on the distribution of the **x** values are termed sampling methods. Of course, all of the methods are related. The class-conditional/generative approach is related to the regression approach in that the former ultimately produces posterior class probabilities, but calculates them in a very specific manner (that is, via Bayes rule), whereas the regression approach is unconstrained in terms of how the posterior probabilities are modeled. Similarly, both the regression and class- conditional/generative approaches implicitly contain decision boundaries; that is, in "decision mode" they map inputs **x** to one of m classes; however, each does so within a probabilistic framework, while the "true" discriminative classifier is not constrained to do so.

NOTES

10.3 The Perceptron

One of the earliest examples of an automatic computer-based classification rule was the perceptron. The perceptron is an example of a discriminative rule, in that it focuses directly on learning the decision boundary surface. The perceptron model was originally motivated as a very simple artificial neural network model for the "accumulate and fire" threshold behavior of real neurons in our brain.

In its simplest form, the perceptron model (for two classes) is just a linear combination of the measurements in **x**. Thus, define h (**x**) = $\sum w_j x_j$, where the w_j, $1 \leq j \leq p$ are the weights (parameters) of the model. One usually adds an additional input with constant value 1 to allow for an additional trainable offset term in the operation of the model. Classification is achieved by comparing h (**x**) with a threshold, which we shall here take to be zero for simplicity. If all class 1 points have h (**x**) > 0 and all class 2 points have h (**x**) < 0, we have perfect separation between the classes. We can try to achieve this by seeking a set of weights such that the above conditions are satisfied for all the points in the training set. This means that the score function is the number of misclassification errors on the training data for a given set of weights $w_1, \ldots, w_{p+1}$. Things are simplified if we transform the measurements of our class 2 points, replacing all the x_j by $-x_j$. Now we simply need a set of weights for which h (**x**) > 0 for all the training set points.

The weights w_j are estimated by examining the training points sequentially. We start with an initial set of weights and classify the first training set point. If this is correctly classified, the weights remain unaltered. If it is incorrectly classified, so that h (**x**) < 0, the weights are updated, so that h (**x**) is increased. This is easily achieved by adding a multiple of the misclassified vector to the weights. That is, the updating rule is $\mathbf{w} = \mathbf{w} + \lambda \mathbf{x}_j$. Here λ is a small constant. This is repeated for all the data points, cycling through the training set several times if necessary. It is possible to prove that if the two classes are perfectly separable by a linear decision surface, then this algorithm will eventually fin d a separating surface, provided a sufficiently small value of λ is chosen. The updating algorithm is reminiscent of the gradient descent techniques, although it is actually not calculating a gradient here but instead is gradually reducing the error rate score function.

Of course, other algorithms are possible, and others are indeed more attractive if the two classes are not perfectly linearly separable as is often the case. In such cases, the misclassification error rate is rather difficult to deal with analytically (since it is not a smooth function of the weights), and the squared error score function is often used instead:

$$S(\mathbf{w}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{p+1} w_j x_j(i) - y(i)\right)^2.$$

Since this is a quadratic error function it has a single global minimum as a function of the weight vector **w** and is relatively straightforward to minimize. Numerous variations of the basic perceptron idea ex ist, including (for example) extensions to handle more than two classes. The appeal of the perceptron model is that it is simple to understand and analyze. However, its applicability in practice is limited by the fact that its decision boundaries are linear (that is, hyperplanes in the input space **X**) and real world classification problems may require more complex decision surfaces for low error rate classification.

10.4 Linear Discriminants

The linear discriminant approach to classification can be conside red a "cousin" of the perceptron model within the general family of linear classifiers. It is based on the simple but useful concept of searching for the linear combination of the variables that best separates the classes. Again, it can be regarded an exam ple of a discriminative approach, since it does not explicitly estimate either the posterior probabilities of class membership or the class-conditional distributions. Fisher presents one of the earliest treatments of linear discriminant analysis (for the two -class case). Let **C** be the pooled sample covariance matrix defined as

$$\hat{\mathbf{C}} = \frac{1}{n_1 + n_2}\left(n_1\hat{\mathbf{C}}_1 + n_2\hat{\mathbf{C}}_2\right),$$

where n_i is the number of training data points per class, and $\hat{\mathbf{C}}_i$ are the p × p sample (estimated) covariance matrices for each class, 1 ≤ i ≤ 2. To capture the notion of separability along any p-dimensional vector **w** , Fisher defined a scalar score function as follows:

NOTES

$$S(\mathbf{w}) = \frac{\mathbf{w}^T \hat{\mu}_1 - \mathbf{w}^T \hat{\mu}_2}{\mathbf{w}^T \hat{\mathbf{C}} \mathbf{w}},$$

where $\hat{\mu}_1$ and $\hat{\mu}_2$ are the p × 1 mean vectors for **x** for data from class 1 and class 2 respectively. The top term is the difference in projected means for each class, which we wish to maximize. The denominator is the estimated pooled variance of the projected data along direction **w** and takes into account the fact that the different variables x_j can have both different individual variances and covariance with each other.

Given the score function S(**w**), the problem is to determine the direction **w** that maximizes this expression. In fact, there is a closed form solution for the maximizing **w**, given by:

$$\hat{\mathbf{w}}_{lda} = \hat{\mathbf{C}}^{-1}(\hat{\mu}_1 - \hat{\mu}_2).$$

A new point is classified by projecting it onto the maximally separating direction, and classifying **x** to class 1 if

$$\mathbf{w}_{lda}^T \left(\mathbf{x} - \frac{1}{2}(\hat{\mu}_1 - \hat{\mu}_2)\right) > \log \frac{p(c_1)}{p(c_2)},$$

where $p(c_1)$ and $p(c_2)$ are the respective class probabilities.

In the special case in which the distributions within each class have a multivariate Normal distribution with a common covariance matrix, this method yields the optimal classification rule as in equation $p(c_k|\mathbf{x})$ (and, indeed, it is optimal whenever the two classes have ellipsoidal distributions with equal quadratic forms). Note, however, that since $\mathbf{w}_{lda}$ was determined without assuming Normality, the linear discriminant methodology can often provide a useful classifier even when Normality does not hold. Note also that if we approach the linear discriminant analysis method from the perspective of assumed forms for the underlying distributions, the method might be more appropriately viewed as being based on the class-conditional distribution approach, rather than on the discriminative approach.

A variety of extensions to Fisher's original linear discriminant model have been developed. Canonical discriminant functions generate m - 1 different decision boundaries (assuming m - 1 < p) to handle the case where the number of classes m > 2. Quadratic discriminant functions lead to quadratic decision boundaries in the input space when the assumption that the covariance matrices are equal is relaxed. Regularized discriminant analysis shrinks the quadratic method toward a simpler form.

Determining the linear discriminant model has computational complexity $O(mp^2n)$. Here we are assuming that $n >> \{p, m\}$ so that the main cost is in estimating the class covariance matrices C_i, $1 \leq i \leq m$. All of these matrices can be found with at most two linear scans of the database (one to get the means and one to generate the $O(p^2)$ covariance matrix terms). Thus the method scales well to large numbers of observations, but is not particularly reliable for large numbers of variables, as the dependence (in terms of the number of parameters to be estimated) on p, the number of variables, is quadratic.

10.5 Tree Models

The basic principle of tree models is to partition (in a recursive manner) the space spanned by the input variables to maximize a score of class purity meaning (roughly, depending on the particular score chosen) that the majority of points in each cell of the partition belong to one class. Thus, for example, with three input variables, x, y, and z, one might split x, so that the input space is divided into two cells. Each of these cells is then itself split into two, perhaps again at some threshold on x or perhaps at some threshold on y or z. This process is repeated as many times as necessary (see below), with each branch point defining a node of a tree. To predict the class value for a new case with known values of input variables, we work down the tree, at each node choosing the appropriate branch by comparing the new case with the threshold value of the variable for that node.

Tree models have been around for a very long time, although formal methods of building them are a relatively recent innovation. Before the development of such methods they were constructed on the basis of prior human understanding of the underlying processes and phenomena generating the data. They have many attractive properties. They are easy to understand and explain. They can handle mixed variables (continuous and discrete, for example) with ease since, in their simplest form, trees partition the space using binary tests (thresholds on real variables and subset membership tests on categorical variables). They can predict the class value for a new case very quickly. They are also very flexible, so that they can provide a powerful predictive tool. However, their essentially sequential nature, which is reflected in the way they are

NOTES

constructed, can sometimes lead to suboptimal partitions of the space of input variables.

The basic strategy for building tree models is simplicity itself: we simply recursively split the cells of the space of input variables. To split a given cell (equivalently, to choose the variable and threshold on which to split the node) we simply search over each possible threshold for each variable to find the threshold split that leads to the greatest improvement in a specified score function. The score is assessed on the basis of the training data set elements. If the aim is to predict to which one of two classes an object belongs, we choose the variable and threshold that leads to the greatest average improvement to the local score (averaged across the two child nodes). Splitting a node cannot lead to a deterioration in the score function on the training data. For classification it turns out that using classification error directly is not a useful score function for selecting variables to split on. Other more indirect measures such as entropy have been found to be much more useful. Note that, for ordered variables, a binary split simply corresponds to a single threshold on the variable values. For nominal variables, a split corresponds to partitioning the variable values into two subsets of values.

In principle, this splitting procedure can be continued until each leaf node contains a single training data point or, in the case when some training data points have identical vectors of input variables (which can happen if the input variables are categorical) continuing until each leaf node contains only training data points with identical input variable values. However, this can lead to severe over fitting. Better trees (in the sense that they lead to better predictions on new data drawn from the same distributions) can typically be obtained by not going to such an extreme (that is, by constructing smaller, more parsimonious trees).

Early work sought to achieve this by stopping the growing process before the extreme had been reached. However, this approach suffers from a consequence of the sequential nature of the procedure. It is possible that the best improvement that can be made at the next step is only very small, so that growth stops, while the step after this could lead to substantial improvement in performance. The "poor" step might be necessary to set things up so that the next step can make a substantial improvement. There is nothing specific to trees about this, of course. It is a general disadvantage of sequential methods: precisely the same applies to the stepwise regression search algorithms which is why more

NOTES

sophisticated methods involving stepping forward and backward have been developed. Similar algorithms have evolved for tree methods.

Nowadays a common strategy is to build a large tree to continue splitting until some termination criterion has been reached in each leaf (for example the points in a node all belong to one class or all have the same **x** vector) and then to prune it back. That is, at each step the two leaf nodes are merged that lead to least reduction in predictive performance on the training set. Alternatively, measures such as minimum description length or cross-validation are used to trade off goodness of fit to the training data against model complexity.

Two other strategies for avoiding the problem of over fitting the training set are also fairly widely used. The first is to average the predictions obtained by the leaves and the nodes leading to the leaves. The second, which has attracted much attention recently, is to base predictions on the averages of several trees, each one constructed by slightly perturbing the data in some way. Such model averaging methods are, in fact, generally suitable for all predictive modeling situations. Model averaging works particularly well with tree models since trees have relatively high variance in the following sense: a tree can be relatively sensitive to small changes in the training data since a slight perturbation in the data could lead to a different root node being chosen and a completely different tree structure being fit. Averaging over multiple perturbations of the data set (e.g., averaging over trees built on bootstrap samples from the training data) tends to counteract this effect by reducing variance.

The most common class value among the training data points at a given leaf node (the majority class) is typically declared as the predicted label for any data points that arrive at this leaf. In effect the region in the input space defined by the branch leading to this node is assigned the label of the most likely class in the region. Sometimes useful information is contained in the overall probability distribution of the classes in the training data at a given leaf. Note that for any particular class, the tree model produces probabilities that are in effect piecewise-constant in the input space, so small changes in the value of an input variable could send a data point down different branches (into a different leaf or region) with dramatically different class probabilities.

NOTES

When seeking the next best split while building a large tree prior to pruning, the algorithm searches through all variables and all possible splits on those variables. For real -valued variables the number of possible positions for splits is typically taken to be n' - 1 (that is, one less than the number of data points n' at each node), each possible position being located halfway between two data points (putting them halfway between is not necessarily optimal, but has the virtue of simplicity). The computational complexity of finding the best splits among p real -valued variables will typically scale as O(pn' log n') if it is carried out in a direct manner. The n' log n' term results from having to sort the variable values at the node in order to calculate the score function: for any threshold we need to know how many points are above and below that threshold. For many score functions we can show that the optimal threshold for ordered variables must be located between two values of the variable that have different class labels. This fact can be used to speed up the search, particularly for large numbers of data points. In addition, various bookkeeping efficiencies can be taken advantage of to avoid resorting as we proceed from node to node. For categorical valued variables, some form of combinatorial search must be conducted to find the best subset of variable values for defining a split.

From a database viewpoint, tree growing can be an expensive procedure. If the number of data points at a node exceeds the capacity of main memory, then the function must operate with a cache of data in main memory and the rest in secondary memory. A brute force implementation will result in linear scans of the database for each node in the tree, resulting in a potentially very slow algorithm. Thus, when we use tree algorithms with data that exceeds the capacity of main memory, we typically either use clever tree algorithms whose data management strategy is tailored to try to minimize secondary memory access, or we resort to working with a random sample that can fit in main memory.

One disadvantage of the basic form of tree is that it is monothetic: each node is split on just one variable. Sometimes, in real problems, the class variable changes most rapidly with a combination of input variables. For example, in a classification problem involving two input variables, it might be that one class is characterized by having low values on both variables while the other has high values on both variables. The decision surface for such a problem would lie diagonally in the input variable space. Standard methods would try to achieve this by multiple splits, ending up with a staircase like approximation to this diagonal decision surface. Figure 10.1 provides a simple illustration of this effect. The optimum, of course, would be achieved by using a threshold defined on a linear combination of the input variables and some extensions to tree methods do just this, permitting linear combinations of the raw input variables to be included in the set of possible variables to be split. Of course, this complicates the search process required for building the tree.

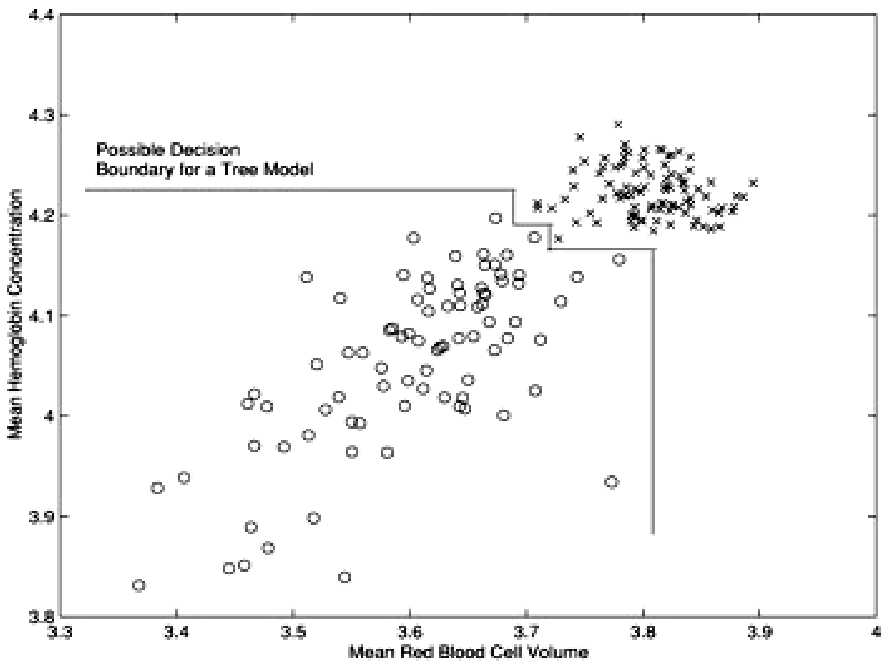


Figure 10.1: Decision Boundary for a Decision Tree for the Red Blood Cell Data from Chapter 9, Composed of "Axis-Parallel" Linear Segments.

10.6 Nearest Neighbor Methods

At their basic level, nearest neighbor methods are very straightforward: to classify a new object, with input vector **y**, we simply examine the k closest training data set points to **y** and assign the object to the class that has the majority of points among these k. Close is defined here in terms of the p - dimensional input space. Thus we are seeking those objects in the training data that are most similar to the new object, in terms of the input variables, and then classifying the new object into the most heavily represented class among these most similar objects.

NOTES

In theoretical terms, we are taking a small volume of the space of variables, centered at **x**, and with radius the distance to the k th nearest neighbor. Then the maximum likelihood estimators of the probability that a point in this small volume belongs to each class are given by the proportion of training points in this volume that belong to each class. The k - nearest neighbor method assigns a new point to the class that has the largest estimated probability. Nearest neighbor methods are essentially in the class of what we have termed "regression" methods they directly estimate the posterior probabilities of class membership.

Of course, this simple outline leaves a lot unsaid. In particular, we must choose a value for k and a metric through which to define close. The most basic form takes k = 1, but this makes a rather unstable classifier (high variance, sensitive to the data), and the predictions can often be made more consistent by increasing k (reduces the variance, but may increase the bias of the method since there is more averaging). However, increasing k means that the training data points now being included are not necessarily very close to the object to be classified. This means that the "small volume" may not be small at all. Since the estimates are estimates of the average probability of belonging to each class in this volume, this may deviate substantially from the value at any particular point within the volume and this deviation is likely to be larger as the volume is larger. The dimensionality p of course plays an important role here: for a fixed number of data points n we increase p (add variables) the data become more and more sparse. This means that the predicted probability may be biased from the true probability at the point in question.

We are back at the ubiquitous issue of the bias/variance trade-off, where increasing k reduces variance but may increase bias. There is theoretical work on the best choice of k, but since this will depend on the particular structure of the data set, as well as other general issues, the best strategy for choosing k seems to be a data-adaptive one: try various values, plotting the performance criterion (the misclassification rate, for example) against k , to find the best. In following this approach, the evaluation must be carried out on a data set independent of the training data (or else the usual problem of overoptimistic results ensues). However, for smaller data sets it would be unwise to reduce the size of the training data set too much by splitting off too large a test set, since the best value of k clearly depends on the number of points in the training data set. A leaving -one-out cross-validated score function is often a useful strategy to follow, particularly for small data sets.

NOTES

Many applications of nearest neighbor methods adopt a Euclidean metric: if **y** is the input vector for the point to be classified, and **x** is the input vector for a training set point, then the Euclidean distance between them is $\sum_j (x_j - y_j)^2$. The problem with this is that it does not provide an explicit measure of the relative importance of the different input variables. We could seek to overcome this by using $\sum_j w_j(x_j - y_j)^2$, where the w_j are weights. This seems more complicated than the Euclidean metric, but the appearance that the Euclidean metric does not require a choice of weights is illusory.

This is easily seen simply by changing the units of measurement of one of the variables before calculating the Euclidean metric. (An exception to this is when all variables are measured in the same unitsas, for example, with situations where the same variable is measured on several different occasions so-called repeated measures data.)

In the two-class case, an optimal metric would be one defined in terms of the contours of probability of belonging to class c_1 that is, $P(c_1|\mathbf{x})$. Training data points on the same contour as **y** have the same probability of belonging to class c_1 as does a point at **y**, so no bias is introduced by including them in the k nearest neighbors. This is true no matter how far from **y** they are, provided they are on the contour. In contrast, points close to **y** but not on the contour of $P(c_1|\mathbf{x})$ through **y** will have different probabilities of belonging to class c_1, so including them among the k will tend to introduce bias. Of course, we do not know the positions of the contours. If we did, we would not need to undertake the exercise at all. This means that, in practice, we estimate approximate contours and base the metrics on these. Both global approaches (for example estimating the classes by multivariate Normal distributions) and local approaches (for example iterative application of nearest neighbor methods) have been used for finding approximate contours.

Nearest neighbor methods are closely related to the kernel methods for density estimation. The basic kernel method defines a cell by a fixed bandwidth and calculates the proportion of points within this cell that belong to each class. This means that the denominator in the proportion is a random variable. The basic nearest neighbor method fixes the proportion (at k /n) and lets the "bandwidth" be a random variable. More sophisticated extensions of both methods (for example, smoothly decaying kernel functions, differential weights on the nearest neighbor points according to their distance from **x**, or choice of bandwidth that varies according to **x**) often lead to methods that are barely distinguishable in practice.

The nearest neighbor method has several attractive properties. It is easy to program and no optimization or training is required. Its classification accuracy can be very good on some problems, comparing favorably with alternative more exotic methods such as neural networks. It permits easy application of the reject option, in which a decision is deferred if we are not sufficiently confident about the predicted class. Extension to multiple classes is straightforward (though the best choice of metric is not so clear here). Handling missing values (in the vector for the object to be classified) is simplicity itself: we simply work in the subspace of those variables that are present.

From a theoretical perspective, the nearest neighbor method is a valuable tool: as the design sample size increases, so the bias of the estimated probability will decrease, for fixed k . If we can contrive to increase k at a suitable rate (so that the variance of the estimates also decreases), the misclassification rate of a nearest neighbor rule will converge to a value related to the Bayes error rate. For example, the asymptotic nearest neighbor misclassification rate (the rate as the number of data points n goes to 8) is bounded above by twice the Bayes error rate.

High-dimensional applications cause problems for all methods. Essentially such problems have to be overcome by adopting a classification rule that is not so flexible that it over fits the data, given the large opportunity for over fitting provided by the many variables. Parametric models of superficially restricted form (such as linear methods) often do well in such circumstances. Nearest neighbor methods often do not do well.

With large numbers of variables (and not correspondingly large numbers of training data cases) the nearest k points are often quite far in real terms. This means that fairly gross smoothing is induced, smoothing that is not related to the classification objectives. The consequence is that nearest neighbor methods can perform poorly in problems with many variables.

In addition, theoretical analyses suggest potential problems for nearest neighbor methods in high dimensions. Under some distributional conditions the ratio of the distance to the closest point and the distance to the most distant point, from any particular **x** point, approaches 1 as the number of dimensions grows. Thus the concept of the nearest neighbor becomes more or less meaningless. However, the distributional assumptions needed for this result are relatively strong, and other more realistic assumptions imply that the notion of nearest neighbor is indeed well defined.

A potential drawback of nearest neighbor methods is that they do not build a model, relying instead on retaining all of the training data set points (for this reason, they are sometimes called "lazy" methods). If the training data set is large, searching through them to find the k nearest can be a time-consuming process. Specifically it can take O(np) per query data point if performed in brute force manner, visiting each of the n training data points and performing p operations to calculate the distance to each. From a memory viewpoint, the method requires us to store the full training data set of size np. Both the time and storage requirements make the direct approach impractical for applications involving very large values of n and/or real-time classification (for example, real-time recommendation of a product to a visitor at a Web site using a nearest neighbor algorithm to find similar individuals from a database with millions of customers). A variety of methods have been developed for accelerating the search and reducing the memory demands of the basic approach. For example, branch and bound methods can be applied: if it is already known that at least k points lie within a distance d of the point to be classified, then a training set point is not worth considering if it lies within a distance d of a point already known to be further than 2d from the point to be classified. This involves preprocessing the training data set. Other preprocessing methods discard certain training data points. For example, condensed nearest neighbor and reduced nearest neighbor methods selectively discard design set points so that those remaining still correctly classify all other training data points. The edited nearest neighbor method discards isolated points from one class that are in dense regions of another class, smoothing out the empirical decision surface in this manner. The gains in speed and memory from these methods depend in general on a variety of factors: the values of n and p, the nature of the particular data set at hand, the particular technique used, and trade-offs between time and memory.

An alternative method for scaling up nearest neighbor methods for large data sets in high dimensions is to use clustering to obtain a grouping of the data. The data points are stored on disk according

NOTES

to their membership in clusters. When finding the nearest point for input point **y**, the clusters nearest to **y** are located and search confined to those clusters. With high probability, under fairly broad assumptions, this method can produce the true nearest neighbor.

10.7 Logistic Discriminant Analysis

For the two-class case, one of the most widely used basic methods of classification based on the regression perspective is logistic discriminant analysis. Given a data point **x**, the estimated probability that it belongs to class c_1 is

$$p(c_1|\mathbf{x}) = \frac{1}{1+\exp(\beta'\mathbf{x})}.$$

Since the probabilities of belonging to the two classes sum to one, by subtraction, the probability of belonging to class 2 is

$$p(c_2|\mathbf{x}) = \frac{\exp(\beta'\mathbf{x})}{1+\exp(\beta'\mathbf{x})}.$$

By inverting this relationship, it is easy to see that the logarithm of the odds ratio is a linear function of the x_j. That is,

$$\log \frac{p(c_2|\mathbf{x})}{p(c_1|\mathbf{x})} = \beta'\mathbf{x}.$$

This approach to modeling the posterior probabilities has several attractive properties. For example, if the distributions are multivariate normal with equal covariance matrices, it is the optimal solution. Furthermore, it is also optimal with discrete **x** variables if the distributions can be modeled by log -linear models with the same interaction terms. These two optimality properties can combine, to yield an attractive model for mixed variables (that is, discrete and continuous) types.

Fisher's linear discriminant analysis method is also optimal for the case of multivariate normal classes with equal covariance matrices. If the data are known to be sampled from such distributions, then Fisher's method is more efficient. This is because it makes explicit use of this information, by modeling the covariance matrix, whereas the logistic method sidesteps this. On the other hand, the more general validity of the logistic method (no real data is ever exactly multivariate normally distributed) means that this is generally preferred to linear discriminant analysis nowadays. The word nowadays here arises because of the algorithms required to compute the parameters of the two models. The mathematical simplicity of the linear discriminant analysis model means that an explicit solution can be found. This is not the case for logistic discriminant analysis, and an iterative estimation procedure must be adopted. The most common such algorithm is a maximum likelihood approach, based on using the likelihood as the score function.

10.8 The Naive Bayes Model

In principle, methods based on the class-conditional distributions in which the variables are all categorical are straightforward: we simply estimate the probabilities that an object from each class will fall in each cell of the discrete variables (each possible discrete value of the vector variable **X**), and then use Bayes theorem to produce a classification. In practice, however, this is often very difficult to implement because of the sheer number of probabilities that must be estimated $O(k^p)$ for p k-valued variables. For example, with p = 30 and binary variables (k = 2) we would need to estimate on the order of $2^{30} \approx 10^9$ probabilities. Assuming (as a rule of thumb) that we should have at least 10 data points for every parameter we estimate (where here the parameters in our model are the probabilities specifying the joint distribution), we would need on the order of 10^{10} data points to accurately estimate the required joint distribution. For m classes (m > 2) we would need m times this number. As p grows the situation clearly becomes impractical.

We can always simplify any joint distribution by making appropriate independence assumptions, essentially approximating a full table of k^p probabilities by products of much smaller tables. At an extreme, we can assume that all the variables are conditionally independent , given the classes that is, that

$$p(\mathbf{x}|c_k) = p(x_1, \ldots, x_p|c_k) = \prod_{j=1}^{p} p(x_j|c_k), \quad 1 \leq k \leq m$$

This is sometimes referred to as the Naive Bayes or first-order Bayes assumption. The approximation allows us to approximate the full conditional distribution requiring $O(k^p)$ probabilities with a product of univariate distributions, requiring in total O(kp) probabilities per class. Thus the conditional independence model is linear in the number of variables

p rather than being exponential. To use the model for classification we simply use the product form for the class-conditional distributions, yielding the Naive Bayes classifier. The reduction in the number of parameters by using the Naive Bayes model above comes at a cost: we are making a very strong independence assumption. In some cases the conditional independence assumption may be quite reasonable. For example, if the x_j are medical symptoms, and the c_k are different diseases, then it may (perhaps) be reasonable to assume that given that a person has disease c_k, the probability of any one symptom depends only on the disease c_k and not on the occurrence of any other symptom. In other words, we are modeling how symptoms appear, given each disease, as having no interactions (note that this does not mean that we are assuming marginal (unconditional) independence). In many practical cases this conditional independence assumption may not be very realistic. For example, let x_1 and x_2 be measures of annual income and savings total respectively for a group of people, and let c_k represent their creditworthiness, this being divided into two classes: good and bad. Even within each class we might expect to observe dependence between x_1 and x_2, because it is likely that people who earn more also save more. Assuming that two variables are independent means, in effect, that we will treat them as providing two distinct pieces of information, which is clearly not the case in this example.

Although the independence assumption may not be a realistic model of the probabilities involved, it may still permit relatively accurate classification performance. There are various reasons for this, including: the fact that relatively few parameters are estimated implies that the variance of the estimates will be small; although the resulting probability estimates may be biased, since we are not interested in their absolute values but only in their ranked order, this may not matter; often a variable selection process has already been undertaken, in which one of each pair of highly correlated variables has been discarded; the decision surface from the naive Bayes classifier may coincide with that of the optimal classifier.

Apart from the fact that its performance is often surprisingly good, there is another reason for the popularity of this particularly simple form of classifier. Using Bayes theorem, our estimate of the

NOTES

probability that a point with measurement vector **x** will belong to the kth class is

$$\begin{aligned} p(c_k|\mathbf{x}) &\propto p(\mathbf{x}|c_k)p(c_k) \\ &= p(c_k)\prod_{j=1}^{p} p(x_j|c_k) \quad 1 \le k \le m \end{aligned}$$

by conditional independence. Now let us take the log -odds ratio and assume that we have just two classes c_1 and c_2. After some straightforward manipulation we get

$$\log\frac{p(c_1|\mathbf{x})}{p(c_2|\mathbf{x})} = \log\frac{p(c_1)}{p(c_2)} + \sum \log\frac{p(x_j|c_1)}{p(x_j|c_2)}.$$

Thus the log odds that a case belongs to class c_1 are given by a simple sum of contributions from the priors and separate contributions from each of the variables. This additive form can be quite useful for explanation purposes since each term $\frac{\log\left(p(x_j|c_1)\right)}{(x_j|c_2)}$, can be viewed as contributing a positive or negative additive contribution to whether c_1 is c_2 is more likely.

The naive Bayes model can easily be generalized in many different directions. If our measurements x_j are real-valued we can still make the conditional independence assumption, where now we have products of estimated univariate densities, instead of distributions. For any real-valued x_j we can estimate $f(x_j|c_k)$ using any of our favorite density estimation techniques for example, parametric models such as a Normal density, more flexible models such as a mixture, or a non-parametric estimate such as a kernel density function. Combinations of real-valued and discrete variables can be handled simply by products of distributions and densities in equation $p(c_k|x)$above.

Despite the simplicity of the form of equations above, the decision surfaces can be quite complicated and are certainly not constrained to be linear (e.g., the multivariate Normal naive Bayes model produces quadratic boundaries in general), in contrast to the linear surfaces produced by simple weighted sums of raw variables (such as those of the perceptron and Fisher's linear discriminant). The simplicity, parsimony, and interpretability of the naive Bayes model have led to its widespread popularity, particularly in the machine learning literature.

NOTES

We can generalize the model equally well by including some but not all dependencies beyond first-order. One can imagine searching for higher order dependencies to allow for selected "significant" pair wise dependencies in the model (such as $p(x_j, x_k|c_k)$, and then triples, and so forth). In doing so we are in fact building a general graphical model for the conditional distribution $p(\mathbf{x}|c_k)$. However, the conventional wisdom in practice is that such additions to the model often provide only limited improvements in classification performance on many data sets, once again underscoring the difference between building accurate density estimators and building good classifiers.

Finally we comment on the computational complexity of the naive Bayes classifier. Since we are just using (in effect) additive models based on simple functions of univariate densities, the complexity scales roughly as pm times the complexity of the estimation for each of the individual univariate class-dependent densities or distributions. For discrete valued variables, the sufficient statistics are simple counts of the number of data points in each bin, so we can construct a naive Bayes classifier with just a single pass through the data. A single scan is also sufficient for parametric univariate density models of real- valued variables (we just need to collect the sufficient statistics, such as the mean and the variance for Normal distributions). For more complex density models, such as mixture models, we may need multiple scans to build the model because of the iterative nature of fitting such density functions.

10.9 Other Methods

A huge number of predictive classification methods have been developed in recent years. Many of these have been powerful and flexible methods, in response to the exciting possibilities offered by modern computing power. We have outlined some of these, showing how they are related. Many other methods also exist, but in just one chapter of one book it is not feasible to do justice to all of them. Furthermore, development and invention have not ended. Exciting work continues even as we write. Examples of methods that we have not had space to cover are:

- Mixture models and radial basis function approaches approximate each class- conditional distribution by a mixture of simpler distributions (for example, multivariate Normal distributions). Even the use of just a few component distributions can lead to a function that is surprisingly effective in modeling the class-conditional distributions.

- Feed-forward neural networks are a generalization of perceptrons. Sometimes they are called multi-layer perceptrons. The first later generates h_1 linear terms, each a weighted combination of the p inputs (in effect, h_1 perceptrons). The h_1 terms are then non-linearly transformed (the logistic function is a popular choice) and the process repeated through multiple layers. The nonlinearity of the transformations permits highly flexible decision surface shapes, so that such models can be very effective for some classification problems. However, their fundamental nonlinearity means that estimation is not straightforward and iterative techniques (such as hill-climbing) must be used. The computational complexity of the estimation process means that such methods may not be particularly useful with large data sets.

- Projection pursuit methods can be viewed as a "cousin" of neural networks. They can be shown, mathematically, to be just as powerful, but they have the advantage that the estimation is more straightforward. They again consist of linear combinations of nonline ar transformations of linear combinations of the raw variables. However, whereas neural networks fix the transformations, in projection pursuit they are data-driven.

- Just as neural networks emerged from early work on the perceptron, so also did support vec tor machines. The early perceptron work assumed that the classes were perfectly separable, and then sought a suitable separating hyperplane. The best generalization performance was obtained when the hyperplane was as far as possible from all of the data points. Support vector machines generalize this to more complex surfaces by extending the measurement space, so that it includes transformations (combinations) of the raw variables. A linear decision surface that perfectly separates the data in this enhanced space is equivalent to a nonlinear decision surface that perfectly separates the data in the original raw measurement space. A distinct feature of this approach is the use of a unique score function, namely the "margin," which attempts to optimize the loc action of the linear decision boundary between the two classes in a manner that is likely to lead to the best possible generalization performance. Practical experience with such methods is rapidly improving, but estimation can be slow since it involves solving a complicated optimization problem that can require $O(n^2)$ storage and $O(n^3)$ time to solve.

Frequently in classification a very flexible model is fitted, and after that it is smoothed in some way to avoid over fitting (or the two processes occur simultaneously), and thus a suitable compromise between bias and variance is obtained. This is manifest in pruning of trees, in weight decay techniques for fitting neural networks, in regularization in discriminant analysis, in the "flatness" of support vector machines, and so on. A rather different strategy, that has proven highly effective in predictive modeling, is to estimate several (or many) models and to average their predictions, as with averaging multiple tree classifiers. This approach clearly has conceptual similarities to the Bayesian model averaging approach, which explicitly regards the parameters of a model (or the model itself) as being uncertain and then averages over this uncertainty when making a prediction. Whereas model averaging has its natural origins in statistics, the similar approach of majority voting among classifiers has its natural origins in machine learning. Yet other ways of combining classifiers are also possible; for example, we can regard the output of classifiers as inputs to a higher level classifier. In principle, any type of predictive classification model can be used at each stage. Of course, parameter estimation will generally not be easy.

A question that obviously arises with the model averaging strategy is: how to weight the different contributions to the average how much weight should each individual classifier be accorded? The simplest strategy is to use equal weights, but it seems obvious that there may be advantages to permitting the use of different weights (not least because equal weights are a special case of this more general model). Various strategies have been suggested for finding the weights, including letting them depend on the predictive performance of the individual model and on the relative complexity of the model. The method of boosting can also be viewed as a model averaging method. Here a succession of models is built, each one being trained on a data set in which points misclassified by the previous model are given more weight. This has obvious similarities to the basic error correction strategy used in early perceptron algorithms. Recent research has provided empirical and theoretical evidence suggesting that boosting can be a highly effective data-driven strategy for building flexible predictive models.

10.10 Evaluating and Comparing Classifiers

This chapter has discussed predictive classification models models for predicting the likely class members hip of a new object, based on a series of measurements on that object. There are many different methods available, so a perfectly reasonable question is "which particular method we should use for a given problem?" Unfortunately, there is no general answer to this question. Choice must depend on features of the problem, the data, and the objectives. We can be aware of the properties of the different methods, and this can help us make a choice, but theoretical properties are not always an effective guide to practical performance (the effectiveness of the independence Bayes model illustrates this). Of course, differences in expected and observed performance serve as a stimulus for further theoretical work, leading to deeper understanding.

If practical results sometimes confound the state of current understanding, we must often resort to empirical comparison of performance to guide our choice of method. There has been a huge amount of work on the assessment and evaluation of classification rules. Much of this work has provided an initial test bed for enhanced understanding in other areas of model building. This section provides a brief introduction to assessing the performance of classification models.

We have so far referred to the error rate or misclassification rate of classification models the proportion of future objects that the rule is likely to incorrectly classify. We defined the Bayes error rate as the optimal error rate the error rate that would result if our model were based on the true distribution functions underlying the data. In practice, of course, these functional forms must be selected a priori (or the alternative discriminative or regression approaches used, and their parameters estimated), so that the model is likely to depart from the optimal. In this case, the model has a true or actual error rate (which can be no smaller than the Bayes error rate). The true error rate is sometimes called the conditional error rate, because it is conditioned on the given training data set.

We will need ways to estimate this true error rate. One obvious way to do this is to reclassify the training data and see what proportion was misclassified. This is the apparent or resubstitution error rate. Unfortunately, this is likely to underestimate the future proportion misclassified. This is because the predictive model has been built so that it does well, in some sense, on the training data.

NOTES

(It would be perverse, to say the least, deliberately to choose a model that did poorly on the training data!) Since the training data is merely a sample from the distributions in question, it will not perfectly reflect these distributions. This means that our model may well reflect part of the data- specific aspects of the training data. Thus, if the training data are reclassified, a higher proportion will be correctly classified than would be the case for future data points.

We have already discussed this phenomenon in different contexts. Many ways have been proposed to overcome it. One straightforward possibility is to estimate future error rate by calculating the proportion misclassified in a new sample a test set. This is perfectly fine apart from the fact that, if a test set is available, we might more fruitfully use it to make a larger training data set. This will permit a more accurate predictive classification model to be constructed. It seems wasteful to ignore part of the data deliberately when we construct the model, unless of course n is very large and we are confident that training on (say) one million data points (keeping another million for testing) is just about as good as training on the full two million.

When our data size is more moderate, various cross-validation approaches have been suggested, in which some small portion (say, one tenth) of the data is left out when the rule is constructed, and then the rule is evaluated on the part that was left out. This can be repeated, with different parts of the data being omitted. Important methods based on this principle are:

- the **leaving-one -out** method, in which only one point is left out at each stage, but each point in turn is left out, so that we end up with a test set of size equal to that of the entir e training set, but where each single point test set is independent of the model it is tested on. Other methods use larger fractions of the data for the test sets (for example, one tenth of the entire data set) but these are more biased than the leaving-one-out method as estimates of the future performance of the model based on the entire data set.

- **bootstrap** methods, of which there are several. These model the relationship between the unknown true distributions and the sample by the relationship between the sample and a subsample of the same size drawn, with replacement, from the sample. In one method, this relationship is used to correct the bias of the resubstitution error rate. Some highly sophisticated variants of bootstrap methods have been developed and they are the most effective methods known to date. Jackknife methods are also based on leaving one training set element out at a time (as in cross-validation), but are equivalent to an approximation to the bootstrap approach.

There are many other methods of error rate estimation. Error rate treats the misclassification of all objects as equally serious. However, this is often (some argue almost always) unrealistic. Often, certain kinds of misclassification are more serious than other kinds. For example, misdiagnosing a patient with a curable but otherwise lethal disease as suffering from some minor illness is more serious than the reverse. In this case, we may want to attach costs to the different kinds of misclassification. In place of simple error rate, then, we seek a model that will minimize overall loss.

These ideas generalize readily enough to the multiple-class case. Often it is useful to draw up a confusion matrix, a cross-classification of the predicted class against the true class. Each cell of such a matrix can be associated with the cost of making that particular kind of misclassification (or correct classification, in the case of the diagonal of the matrix) so that overall loss can be evaluated.

Unfortunately, costs are often difficult to determine. When this is the case, an alternative strategy is to integrate over all possible values of the ratio of one cost to the other (for the two-class case generalizations are possible for more than two classes). This approach leads to what is known as the Gini co-efficient of performance. This measure is equivalent to the test statistic used in the Mann -Whitney -Wilcoxon statistical test for comparing two independent samples, and is also equivalent to the area under a Receiver Operating Characteristic or ROC curve (a plot of the estimated proportion of class 1 objects correctly classified as class 1 against the estimated proportion of class 2 objects incorrectly classified as class 1). ROC curves and the areas under them are widely used in some areas of research. They are not without their interpretation problems, however.

Simple performance of classification models is but one aspect of the choice of a method. Another is how well the method matches the data. For example, some methods are better suited to discrete **x** variables, and others to continuous **x**, while others work with either type with equal facility. Missing values, of course, are a potential (and, indeed, ubiquitous) problem with any method. Some methods can handle incomplete data more readily than others. The independence Bayes method, for example, handles such data very easily, whereas Fisher's linear discriminant analysis approach does not. Things are further complicated by the fact that data may be missing for various reasons, and that the reasons can affect the validity of the model built on the incomplete data.

10.11 Feature Selection for Classification in High Dimensions

An important issue that often confronts data miners in practice is the problem of having too many variables. Simply put, not all variables that are measured are likely to be necessary for accurate discrimination and including them in the classification model may in fact lead to a worse model than if they were removed. Consider the simple example of building a system to discriminate between images of male and female faces (a task that humans perform effortlessly and relatively accurately but that is quite challenging for an image classification algorithm). The colors of a person's eyes, hair, or skin are hardly likely to be useful in this discriminative context. These are variables that are easy to measure (and indeed are general characteristics of a person's appearance) but carry little information as to the class identity in this particular case.

In most data mining problems it is not so obvious which variables are (or are not) relevant. For example, relating a person's demographic characteristics to online purchasing behavior may be quite subtle and may not necessarily follow the traditional patterns (consider a hypothetical group of high-income PhD -educated consumers who spend a lot of money on comic books if they exist, a comic-book retailer would like to know!). In data mining we are particularly interested in letting the data speak, which in the context of variable selection means using data-adaptive methods for variable

selection (while noting as usual that should useful prior knowledge be available to inform us about which variables are clearly irrelevant to the task, then by all means we should use this information).

Where we outlined some general strategies that we briefly review here:

- **Variable Selection:** The idea here is to select a subset p' of the original p variables. Of course we don't know in advance what value of p' will work well or which variables should be included, so there is a combinatorially large search space of variable subsets that could be considered. Thus most approaches rely on some form of heuristic search through the space of variable subsets, often using a greedy approach to add or delete variables one at a time. There are two general approaches here: the first uses a classification algorithm that automatically performs variable selection as part of the definition of the basic model, the classification tree model being the best-known example. The second approach is to use the classifier as a "black box" and to have an external loop (or "wrapper") that systematically adds and subtracts variables to the current subset, each subset being evaluated on the basis of how well the classification model performs.

- **Variable Transformations:** The idea here is to transform the original measurements by some linear or nonlinear function via a preprocessing step, typically resulting in a much smaller set of derived variables, and then to build the classifier on this transformed set.

Summary

Descriptive models as described in simply summarize data in convenient ways or in ways that we hope will lead to increased understanding of the way things work. In contrast, predictive models have the specific aim of allowing us to predict the unknown value of a variable of interest given known values of other variables. Examples include providing a diagnosis for a medical patient on the basis of a set of test results, estimating the probability that customers will buy product A given a list of other products they have purchased, or predicting the value of the Dow Jones index six months from now, given current and past values of the index.

We described in detail predictive models in which the variable to be predicted (the response variable) was a nominal variable that is, it could take one of only a finite (and typically small) number of values and these values had no numerical significance, so that they were simply class identifiers. In this chapter we turn to predictive models in which the response variable does have numerical significance. Examples are the amount a retail store might earn from a given customer over a ten-year period, the rate of fuel consumption of a

NOTES

given type of car under normal conditions, the number of people who might access a particular Web site in a given month, and so on. The variables to be used as input for prediction will be called predictor variables and the variable to be predicted is the response variable. Other authors sometimes use the terms dependent or target for the response variable, and independent, explanatory, or regressor for the predictor variables. Other names used in the classification context were mentioned. Note that the predictor variables can be numerical, but they need not be. Our aim, then, is to use a sample of objects, for which both the response variable and the predictor variables are known, to construct a model that will allow prediction of the numerical value of the response variable for a new case for which only the predictor variables are known.. In fact, as we will see later in this chapter, we can also treat prediction of nominal variables (that is, classification) within this general framework of regression.

Reference

1. Hand D, Mannila H. Smith P: Principles of Data Mining (PHI).

2. Pujari A: Data Mining Techniques, University Press (orient Longman.

UNIT – V

11. Predictive Modeling for Regression

Structure

Objective

After going through this lesson, you should be able to:

- Discuss about a linear models and least squares fitting;
- Discuss generalized linear models;
- Discuss artificial neural networks;

11.1 Introduction

In this lesson we turn to predictive models in which the response variable does have numerical significance. Examples are the amount a retail store might earn from a given customer over a ten-year period, the rate of fuel consumption of a given type of car under normal conditions, the number of people who might access a particular Web site in a given month, and so on. The variables to

NOTES

be used as input for prediction will be called predictor variables and the variable to be predicted is the response variable. Other authors sometimes use the terms dependent or target for the response variable, and independent, explanatory, or regressor for the predictor variables. Note that the predictor variables can be numerical, but they need not be. Our aim, then, is to use a sample of objects, for which both the response variable and the predictor variables are known, to construct a model that will allow prediction of the numerical value of the response variable for a new case for which only the predictor variables are known.

Accuracy of prediction is one of the most important properties of such models, so various measures of accuracy have been devised. These measures may also be used for choosing between alternative models, and for choosing the values of parameters in models. In the terminology introduced earlier, these measures are score functions, by which different models may be compared.

Predictive accuracy is a critical aspect of models, but it is not the only aspect. For example, we might use the model to shed insight into which of the predictor variables are most important. We might even insist that some variables be included in the model, because we know they should be there on substantive grounds, even though they lead to only small predictive improvement. Contrariwise, we might omit variables that we feel would enhance our predictive performance. (An example of this situation arises in credit scoring, in which, in many countries, it is illegal to include sex or race as a predictor variable.) We might be interested in whether predictor variables interact, in the sense that the effect that one has on the response variable depends on the values taken by others. For obvious reasons, we might be interested in whether good prediction can be achieved by a simple model. Sometimes we might even be willing to sacrifice some predictive accuracy in exchange for substantially reduced model complexity. Though predictive accuracy is perhaps the most important component of the performance of a predictive model, this has to be tempered by the context in which the model is to be applied.

11.2 Linear Models and Least Squares Fitting

The idea of linear models they are linear in the parameters. The simplest such model yields predicted values, y, of the response variable y, that are also a linear combination of the predictor variables x_j:

$$\hat{y} = a_o + \sum_{i=1}^{p} a_j x_j$$

In fact, of course, we will not normally be able to predict the response variable perfectly (life is seldom so simple) and a common aim is to predict the mean value that y takes at each vector of the predictor variables so y, is our predicted estimate of the mean value at **x** = (x_1, ..., x_p). Models of this form are known as linear regression models. In the simplest case of a single predictor variable (simple regression), we have a regression line in the space spanned by the response and predictor variables. More generally (multiple regression) we have a regression plane. Such models are the oldest, most important, and single most widely used form of predictive model. One reason for this is their evident simplicity; a simple weighted sum is very easy both to compute and to understand. Another compelling reason is that they often perform very well even in circumstances in which we know enough to be confident that the true relationship between the predictor and response variables cannot be linear. This is not altogether surprising: when we expand continuous mathematical functions in a Taylor series we often find that the lowest order terms the linear terms are the most important, so that the best simple approximation is obtained by using a linear model.

It is extremely rare that the chosen model is exactly right. This is especially true in data mining situations, where our model is generally empirical rather than being based on an underlying theory. The model may not include all of the predictor variables that are needed for perfect prediction (many may not have been measured or even be measurable); it may not include certain functions of the predictor variables (maybe x_1^2 is needed as well as x_1, or maybe products of the predictor variables are needed because they interact in their effect on y); and, in any case, no measurement is perfect; the y variable will have errors associated with it so that each vector (x_1, ..., x_p) will be associated with a distribution of possible y values, as we have noted above.

All of this means that the actual y values in a sample will differ from the predicted values. The differences between observed and predicted values are called residuals, and we denote these by e:

$$y(i) = \hat{y}(i) + e(i) = c_0 + \sum_{j=1}^{p} a_j x_j(i) + e(i), \qquad 1 \le i \le n$$

NOTES

In matrix terms, if we denote the observed y measurements on the n objects in the training sample by the vector **y** and the p measurements of the predictor variables on the n objects by the n by p + 1 matrix **X** (an additional column of 1s are added to incorporate the intercept term a_0 in the model), we can express the relationship between the observed response and predictor measurements, in terms of our model, as

$$\mathbf{y} = \mathbf{X}\mathbf{a} + \mathbf{e},$$

where **y** is an n × 1 matrix of response values, **a** = (a_0, ..., a_p) represents the (p+1) × 1 vector of parameter values, and the n × 1 vector e = (e(1), ..., e(n)) contains the residuals. Clearly we want to choose the parameters in our model (the values in the p + 1 vector **a**) so as to yield predictions that are as accurate as possible. Put another way, we must find estimates for the a_j that minimize the e discrepancies in some way. To do this, we combine the elements of **e** in such a way as to yield a single numerical measure that we can minimize. Various ways of combining the e(i) have been proposed, but by far the most popular method is to sum their squares that is, the sum of squared errors score function. Thus we seek the values for the parameter vector **a** that minimizes

$$\sum_{i=1}^{n} e(i)^2 = \sum_{i=1}^{n} \left(y(i) - \sum_{j=0}^{p} a_j x_j(i) \right)^2$$

In this expression, y(i) is the observed y value for the ith training sample point and

$$(x_0(i), x_1(i), ..., x_p(i)) = (1, x_1(i), ..., x_p(i))$$

is the vector of predictor variables for this point. For obvious reasons, this method is known as the least squares method. For simplicity, we will denote the parameter vector that minimizes this by (a_0, ..., a_p). (It would be more correct, of course, if we used some notation to indicate that it is an estimate, such as ($\hat{a}_0$, ..., $\hat{a}_p$), but our notation has the merit of simplicity.) In matrix terms, the values of the parameters that minimize equation $\sum_{i=1}^{n} e(i)^2$ can be shown to be

$$\mathbf{a} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}.$$

In linear regression in general, the parameters are called regression coefficients.

11.2.1 Computational Issues in Fitting the Model

Solving equation **a** directly requires that the matrix $\mathbf{X}^T\mathbf{X}$ be invertible. Problems will arise if the sample size n is small (rare in data mining situations) or if there are linear dependencies between the measured values of the predictor variables (not so rare). In the latter case, modern software packages normally issue warnings, and appropriate action can be taken, such as dropping some of the predictor variables.

A rather more subtle problem arises when the measured values of the predictor variables are not exactly linearly dependent, but are almost so. Now the matrix can be inverted, but the solution will be unstable. This means that slight alterations to the observed **X** values would lead to substantial differences in the estimated values of **a**. Different measurement errors or a slightly different training sample would have led to different parameter estimates. This problem is termed multicollinearity. The instability in the estimated parameters is a problem if these values are the focus of interest for example, if we want to know which of the variables is most important in the model. However, it will not normally be a problem as far as predictive accuracy is concerned: although substantially different **a** vectors may be produced by slight variations of the data, all of these vectors will lead to similar predictions for most $\mathbf{x}_k$ vectors.

Solving equation **a** is usually carried out by numerical linear algebra techniques for equation solving (such as the LU decomposition or the singular value decomposition (SVD)), which tend to have better numerical stability than that achieved by inverting the matrix $\mathbf{X}^T\ \mathbf{X}$ directly. The underlying computational complexity is typically the same no matter which particular technique is used, namely, $O(p^2n + p^3)$. The p^2 n term comes from the n multiplications required to calculate each element in the p × p matrix $\mathbf{C} = \mathbf{X}^T\ \mathbf{X}$. The p^3 term comes from then solving $\mathbf{Ca} = \mathbf{X}^T\ \mathbf{y}$ for **a**.

In lesson 6 we remarked that the additive nature of the regression model could be retained while permitting more flexible model forms by including transformations of the raw x_j as well as the raw variables themselves. Figure 11.1 shows a plot of data collected in an experiment in which a subject performed a physical task at a gradually increasing level of difficulty. The vertical axis shows a measure on the gases expired from the lungs while the horizontal axis shows the oxygen uptake. The nonlinearity of the relationship between these two variables is quite clear from the plot. A straight line $y = a_0 + a_1x$ provides a poor fit as is shown in the figure. The predicted values from this model would be accurate only for x (oxygen uptake) values just above 1000 and just below 4000. (Despite this, the model is not grossly inaccurate the point made earlier about models linear in x providing reasonable approximations is clearly true.) However, the model $y = a_0 + a_1x + a_2x^2$ gives the fitted line shown in figure 11.2. This model is still linear in the parameters, so that these can be easily estimated using the same standard matrix manipulation shown above in equation **a**. It is clear that the predictions obtained from this model are about as good as they can be. The remaining inaccuracy in the model is the irreducible measurement error associated with the variance of y about its mean at each value of x.

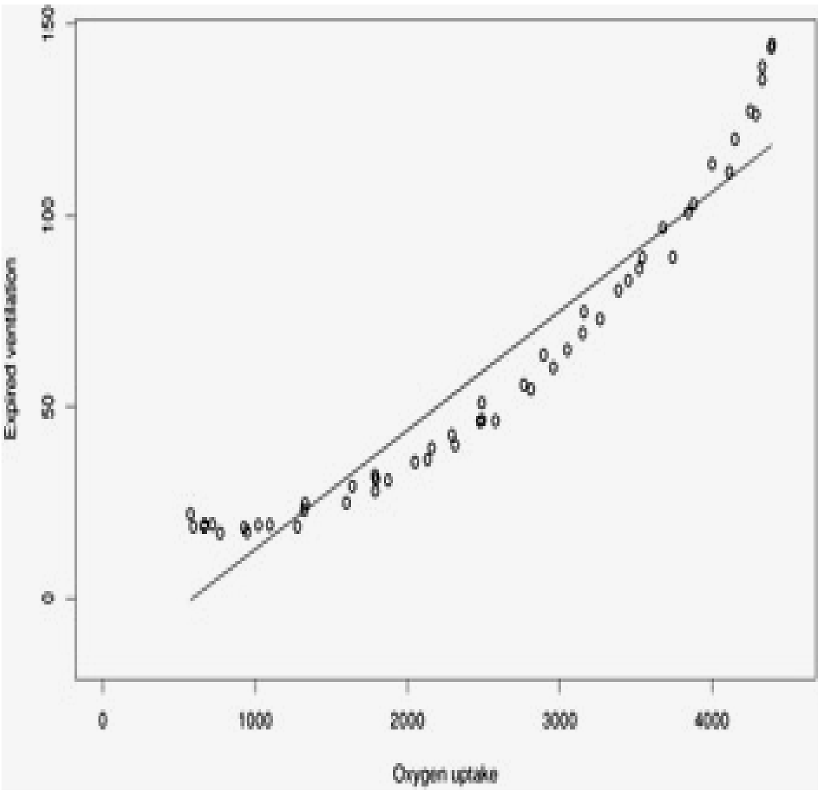


Figure 11.1: Expired ventilation plotted against oxygen uptake in a series of trials, with fitted straight line.

NOTES

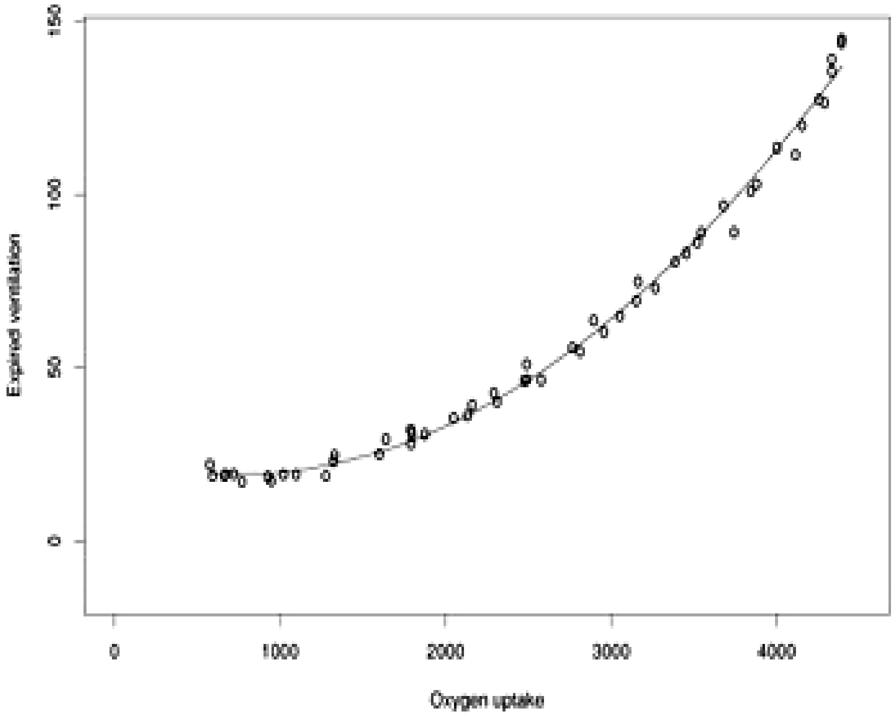


Figure 11.2: The data from figure 11.1 with a model that includes a term in x^2.

11.2.2 A Probabilistic Interpretation of Linear Regression

This informal data analytic route allows us to fit a regression model to any data set involving a response variable and a set of predictor variables, and to obtain a vector of estimated regression coefficients. If our aim were merely to produce a convenient summary of the training data (as, very occasionally, it is) we could stop there. However, this chapter is concerned with predictive models. Our aim is to go beyond the training data to predict y values for other "out -of-sample" objects. Goodness of fit to the given data is all very well, but we are really interested in fit to future data that arise from the same process, so that our future predictions are as accurate as possible. In order to explore this, we need to embed the model-building process in a more formal inferential context. To do this, we suppose that each observed value $y(i)$ is produced as a sum of weighted predictor variables $a^T \mathbf{x}(i)$ and a random term (i) that follows a $N(0, s^2)$ distribution independent of other values. (Note that implicit in this is the assumption that the variances of the random terms are all the same s^2 is the same for all possible values of the vector of predictor variables. We will discuss this assumption further below.) The $n \times 1$

random vector **Y** thus takes the form **Y** = **X**α +є. The observed n × 1 **y** vector is a realization from this distribution. The components of the n × 1 vector are often called errors. Note that they are different from the residuals, **e**. An "error" is a random realization from a given distribution, whereas a residual is a difference between a fitted model and an observed y value. Note also that is different from **a**. a is represents the underlying and unknown truth, whereas **a** gives the values used in a model of the truth.

It turns out that within this framework the least squares estimate **a** is also the maximum likelihood estimate of a. Furthermore, the covariance matrix of the estimate **a** obtained above is $(\mathbf{X}^T \mathbf{X})^{-1}\sigma^2$, where this covariance matrix expresses the uncertainty in our parameter estimates **a**. In the case of a single predictor variable, this gives

$$\left(1 + \frac{n\bar{x}^2}{\sum_i \left(x(i) - \bar{x}\right)^2}\right) \frac{\sigma^2}{n}$$

for the variance of the intercept term and

$$\frac{\sigma^2}{\sum_i \left(x(i) - \bar{x}\right)^2}$$

for the variance of the slope. Here $\hat{\mathbf{x}}$ is the sample mean of the single predictor variable. The diagonal elements of the covariance matrix for **a** above give the variances of the regression coefficients which can be used to test whether the individual regression coefficients are significantly different from zero. If v_j is the jth diagonal element of $(\mathbf{X}^T \mathbf{X})^{-1}\sigma^2$, then the ratio $\frac{a_j}{\sqrt{v_j}}$ can be compared with a t(n - p - 1) distribution to see whether the regression coefficient is zero. However, as we discuss below, this test makes sense only in the context of the other variables included in the model, and alternative methods, also discussed below, are available for more elaborate model-building exercises. If **x** is the vector of predictor variables for a new object, with predicted y value y, then the variance of y is $\mathbf{x}^T(\mathbf{X}^T \mathbf{X})^{-1}\mathbf{x}\sigma^2$. With one predictor variable, this reduces to $\frac{1}{n} + \frac{(x-\bar{x})^2}{\Sigma(x(i) - \bar{x})^2}$. Note that this variance is greater the further **x** is from the mean of the training sample. That is, the least accurate predictions, in terms of variance, are those in the tails of the distribution of predictor variables. Note also that confidence intervals based on this variance are confidence values for the predicted value of y.

We may also be interested in (what are somewhat confusingly called) prediction intervals, telling us a range of plausible values for the observed y at a given value of x, not a range of plausible values for the predicted value. Prediction intervals must include the uncertainty arising from our prediction and also that arising from the variability of y about our predicted value. This means that the variance above is increased by an extra term s^2, yielding $\frac{1}{n} + \frac{(x-\bar{x})^2}{\sum (x(i) - \bar{x})^2}$

11.2.3 Interpreting the Fitted Model

The coefficients in a multiple regression model can be interpreted as follows: if the jth predictor variable, x_j, is increased by one unit, while all the other predictor variables are kept fixed, then the response variable y will increase by a_j. The regression coefficients thus tell us the conditional effect of each predictor variable, conditional on keeping the other predictor variables constant. This is an important aspect of the interpretation. In particular, the size of the regression coefficient associated with the jth variable will depend on what other variables are in the model. This is clearly especially important if we are constructing models in a sequential manner: add another variable and the coefficients of those already in the model will change. (There is an exception to this. If the predictor variables are orthogonal, then the estimated regression coefficients are unaffected by the presence or absence of others in the model. However, this situation is most common in designed experiments, and is rare in the kinds of secondary data analyses encountered in data mining.) The sizes of the regression coefficients tell us the relative importance of the variables, in the sense that we can compare the effects of unit changes. Note also that the size of the effects depends on the chosen units of measurement for the predictor variables. If we measure x_1 in kilometers instead of millimeters, then its associated regression coefficient will be multiplied by a million. This can make comparisons between variables difficult, so people often work with standardized variables measuring each predictor variable relative to its standard deviation. We used the sum of squared errors between the predictions and the observed y values as a criterion through which to cho ose the values of the parameters in the model. This is the residual sum of squares or the sum of squared residuals, $\sum e(i)^2 = \sum \left(y(i) - \hat{y}(i)\right)^2$. In a sense, the worst model would be obtained if we simply predicted all of the y values by the value $\bar{y}$ the mean of the sample of y values that is constant relative to the x values (thus effectively ignoring the inputs to the model and always guessing the output to be the mean of y). The total sum of squares is defined as the sum of squared errors for this worst model $\sum\left(y(i) - \bar{y}(i)\right)^2$. The difference between the residual sum of squares from a model and the total sum of squares is the sum of squares that can be attributed to the regression for that model it is the regression sum of squares. This is the sum of squared differences of the predicted values, $\hat{y}(i)$, from the overall mean, $\sum(\hat{y}(i) - \bar{y}(i))^2$. The symbol R^2 is often used for the "multiple correlation coefficient," the ratio of the regression sum of squares to total sum of squares:

$$R^2 = \frac{\sum(\hat{y}(i) - \bar{y})^2}{\sum(y(i) - \bar{y})^2}.$$

A value near 1 tells us that the model explains most of the y variation in the data. The number of independent components contributing to each sum of squares is called the number of degrees of freedom for that sum of squares. The degrees of freedom for the total sum of squares is n - 1 (one less than the sample size, since the components are all calculated relative to the mean). The degrees of freedom for the residual sum of squares is n - 1 - p (although there are n terms in the summation, p + 1 regression coefficients are calculated). The degrees of freedom for the regression sum of squa res are p, the difference between the total and residual degrees of freedom. These sums of squares and their associated degrees of freedom are usefully put together in an analysis of variance table, as in table 11.1, summarizing the decomposition of the totals into components. The meaning of the final column is described below.

Table 11.1: The Analysis of Variance Decomposition Table for a Regression

Source of variation	Sum of squares	Degree of freedom	Mean square

Regression residual total	$\sum(\hat{y}(i)-\bar{y}(i))^2$ $\sum(y(i)-\hat{y}(i))^2$ $\sum(y(i)-\bar{y}(i))^2$	P n-p-1 n-1	$\sum(\hat{y}(i)-\bar{y}(i))^2$ /p $\sum(y(i)-\hat{y}(i))^2$ /(n-p-1

11.2.4 Inference and Generalization

We have already noted that our real aim in building predictive models is one of inference: we want to make statements (predictions) about objects for which we do not know the y values. This means that goodness of fit to the training data is not our real objective. In particular, for example, merely because we have obtained nonzero estimated regression coefficients, this does not necessarily mean that the variables are related: it could be merely that our model has captured chance idiosyncrasies of the training sample. This is particularly relevant in the context of data mining where many models may be explored and fit to the data in a relatively automated fashion. As discussed earlier, we need some way to test the model, to see how easily the observed data could have arisen by chance, even if there was no structure in the population the data were collected from. In this case, we need to test whether the population regression coefficients are really zero. (Of course, this is not the only test we might be interested in, but it is the one most often required.) It can be shown that if the values of a_j are actually all zero (and still making the assumption that the $\epsilon(i)$ are independently distributed as N(0, σ^2)),

$$\frac{\sum(\hat{y}(i)-\bar{y})^2/p}{\sum(y(i)-\bar{y})^2/(n-p-1)}$$

has an F(p, n - p - 1) distribution. This is just the ratio of the two mean squares given in table 11.1. The test is carried out by comparing the value of this ratio with the upper critical level of the F(p, n - p - 1) distribution. If the ratio exceeds this value the test is significant and we would conclude that there is a linear relationship between the y and nx_j variables (or that a very unlikely event has occurred). If the ratio is less than the critical value we have no evidence to reject the null hypothesis that the population regression coefficients are all zero.

11.2.5 Model Search and Model Building

We have described an overall test to see whether the regression coefficients in a given model are all zero. However, we are more often involved in a situation of searching over model space or model building in which we examine a sequence of models to find one that is "best" in some sense. In particular, we often need to examine the effect of adding a set of predictor variables to a set we have already included. Note that this includes the special case of adding just one extra variable, and that the idea is applied in reverse, it can also handle the situation of removing variables from a model.

In order to compare models we need a score function. Once again, the obvious one is the sum of squared errors between the predictions and the observed y values. Suppose we are comparing two models: a model with p predictor variables (model M) and the largest model we are prepared to contemplate, with q variables (these will include all the untransformed predictor variables we think might be relevant, along with any transformations of them we think might be relevant), model M*. Each of these models will have an associated residual sum of squares, and the difference between them will tell us how much better the larger model fits the data than the smaller model. (Equivalently, we could calculate the difference between the regression sums of squares. Since the residual and regression sum of squares sum to the total sum of squares, which is the same for both models, the two calculations will yield the same result.) The degrees of freedom associated with the difference between the residual sums of squares for the two models is q - p, the extra number of regression coefficients computed in fitting the larger model, M*. The ratio between the difference of the residual sums of squares and the difference of degrees of freedom again gives us a mean square now a mean square for the difference between the two models. Comparison of this with the residual mean

square for model M* gives us an F-test of whether the difference between the models is real or not. Table 11.2 illustrates this extension. From this table, the ratio

$$\left[\frac{(SS(M^*) - SS(M))}{(q-p)}\right] / \left[\frac{(SS(T) - SS(M^*))}{(n-q-1)}\right]$$

is compared with the critical value of an F(q - p, n - q - 1) distribution.

Table 11.2: The Analysis of Variance Decomposition Table for Model Building.

Source of variation	Sum of squares	Degree of freedom	Mean square
Regression model 1	SS(M)	P	SS(M)/p
Regression full model	SS(M*)	q	SS(M*)/p
Difference	SS(M*) - SS(M)	q-p	$\frac{SS(M^*) - SS(M)}{q - p}$
Residual	SS(T) - SS(M*)	n-q-1	$\frac{SS(T) - SS(M^*)}{n - q - 1}$
Total	SS(T)	n-1	

This is fine if we have just a few models we want to compare, but data mining problems are such that often we need to rely on automatic model building processes. Such automatic methods are available in most modern data mining computer packages. There are various strategies that may be adopted. A basic form is a forward selection method, in which variables are added one at a time to an existing model. At each step that variable is chosen from the set of potential variables that leads to the greatest increase in predictive power (measured in terms of reduction of sum of squared residuals), provided the increase exceeds some specified threshold. Ideally, the addition would be made as long as the inc rease in predictive power was statistically significant, but in practice this is complicated to ensure: the variable selection process necessarily involves carrying out many tests, not all independent, so that computing correct significance values is a nontrivial process. The simple significance level based on table 11.2 does not apply when multiple dependent tests are made. (The implication of this is that if the significance level is being used to choose variables, then it is being used as a score function, and should not be given a probabilistic interpretation.)

We can, of course, in principle use any of the score functions discussed for model selection in regression, such as BIC, minimum description length, cross validation, or more Bayesian methods. These provide an alternative to the hypothesis- testing framework that measures the statistical significance of adding and deleting terms on a model-by-model basis. Penalized score functions such as BIC, and variations on cross-validation tailored specifically to regression, are commonly used in practice as score functions for model selection in regression.

NOTES

A strategy opposite to that of forward selection is backward elimination. We begin with the most complex model we might contemplate (the "largest model," M*, above) and progressively eliminate variables, selecting them on the basis that eliminating them leads to the least increase in sum of squared residuals (again, subject to some threshold). Other variants include combinations of forward selection and backward elimination. For example, we might add two variables, eliminate one, add two, and remove one, and so on. For data sets where the number of variables p is very large, it may be much more practical computationally to build the model in the forward direction than in the backward direction. Stepwise methods are attempts to restrict the search of the space of all possible sets of predictor variables, so that the search is manageable. But if the search is restricted, it is possible that some highly effective combination of variables may be overlooked. Very occasionally (if the set of potential predictor variables is small), we can examine all possible sets of variables (although, with p variables, there are (2^p - 1) possible subsets). The size of problems for which all possible subsets can be examined has been expanded by the use of strategies such as branch and bound, which rely on the monotonicity of the residual sum of squares criterion.

A couple of cautionary comments are worth making here. First, as we have noted, the coefficients of variables already in the model will generally change as new variables are added. A variable that is important for one model may become less so when the model is extended. Second, as we have discussed in earlier chapters, if too elaborate a search is carried out there is a high chance of over fitting the training set that is, of obtaining a model that provides a good fit to the training set (small residual sum of squares) but does not predict new data very well.

11.2.6 Diagnostics and Model Inspection

Although multiple regressions are a very powerful and widely used technique, some of the assumptions might be regarded as restrictive. The assumption that the variance of the y distribution is the same at each vector **x** is often inappropriate. (This assumption of equal variances is called homoscedasticity. The converse is heteroscedasticity.) For example, figure 11.3 shows the normal average January minimum temperature (in deg F) plotted against the latitude (deg N) for 56 cities in the United States. There is evidence that, for smaller latitudes, at least, the variance of the temperature increases with increasing latitude (although the mean temperature seems to decrease). We can still apply the standard least squares algorithm above to estimate parameters in this new situation (and the resulting estimates would still be unbiased if the model form were correct), but we could do better in the sense that it is possible to find estimators with smaller variance.

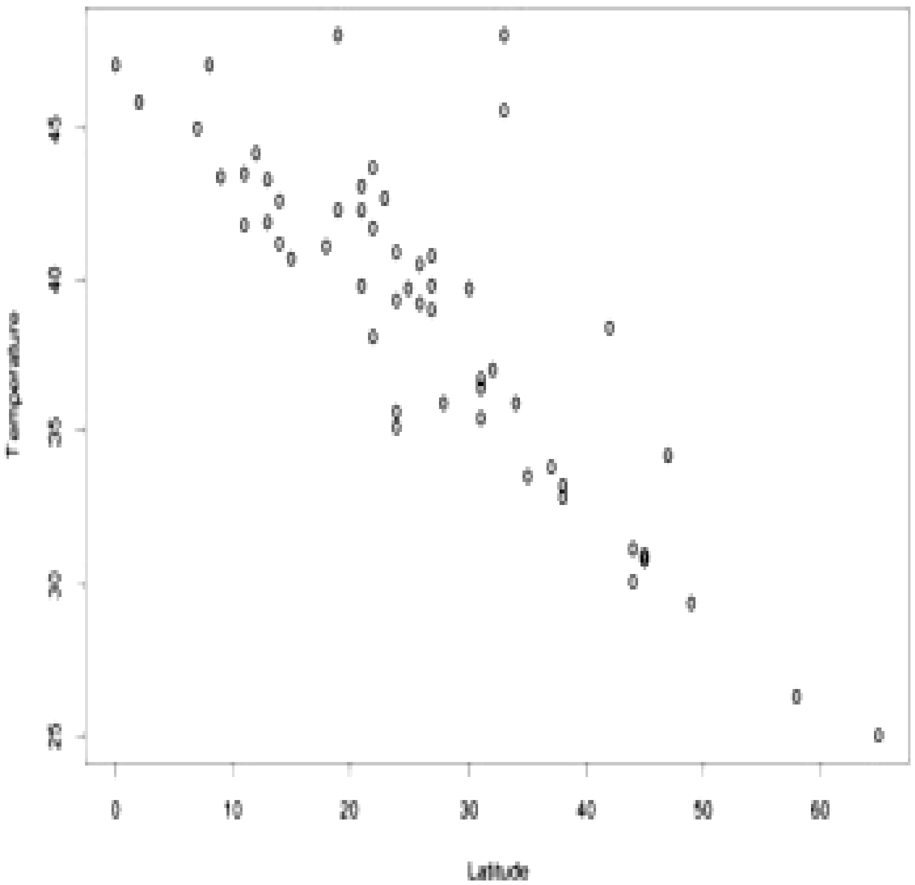


Figure 11.3: Temperature (degrees f) against latitude (degrees n) for 56 cities in the United States.

To do this we need to modify the basic method. Essentially, we need to arrange things so that those values of **x** associated with y values with larger variance are weighted less heavily in the model fitting process. This makes perfect sense it means that the estimator is more influenced by the more accurate values. Formally, this idea leads to a modification of the solution equation **a**. Suppose that the covariance matrix of the n×1 random vector є is the n × n matrix σ^2**V** (previously we took **V** = **I**). The case of unequal variances means that **V** is diagonal with terms that are not all equal. Now it is possible (see any standard text on linear algebra) to find a unique nonsingular matrix **P** such that $\mathbf{P}^T\mathbf{P}=\mathbf{V}$ We can use this to define a new random vector **f** = $\mathbf{P}^{-1}\epsilon$, and it is easy to show that the covariance matrix of **f** is $\sigma^2\mathbf{I}$. Using this idea, we form a new model by pre-multiplying the old one by $\mathbf{P}^{-1}$:

$$\mathbf{P}^{-1}\mathbf{Y} = \mathbf{P}^{-1}\mathbf{X}\alpha + \mathbf{P}^{-1}\epsilon$$

Or $$\mathbf{Z} = \mathbf{W}\beta + \mathbf{f},$$

now of the form required to apply the standard least squares algorithm. If we do this, and then convert the solution back into the original variables **Y**, we obtain:

$$\mathbf{a} = (\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X})\,\mathbf{X}\mathbf{V}^{-1}\mathbf{y},$$

a weighted least squares solution. The variance of this estimated parameter vector **a** is $(\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X})^{-1}\sigma^2$.

Unequal variances of the y distributions for different **x** vectors is one way in which the assumptions of basic multiple regression can break down. There are others. What we really need are ways to explore the quality of the model and tools that will enable us to detect where and why the model deviates from the assumptions. That is, we require diagnostic tools. In simple regression, where there is only one predictor variable, we can see the quality of the model from a plot of y against x (see figures 11.1, 11.2 and 11.3). More generally, however, when there is more than one predictor variable, such a simple plot is not possible, and more sophisticated methods are needed. In general, the key features for examining the quality of a regression model are the residuals, the components of the vector **e** = **y** - **y**. If there is a pattern to these, it tells us that the model is failing to explain the distribution of the data. Various plots involving the residuals are used, including plotting the residuals against the fitted values, plotting standardized residuals (obtained by dividing the residuals by their standard errors) against the fitted values, and plotting the standardized residuals against standard normal quintiles. (The latter are "normal probability plots." If the residuals are approximately normally distributed, the points in this plot should lie roughly on a straight line.) Of course, interpreting some of the diagnostic plots requires practice and experience.

One general cautionary comment, applies to all predictive models:

such models are valid only within the bounds of the data. It can be very risky to extrapolate beyond the data. A very simple example is given in figure 11.4. This shows a plot of the tensile strength of paper plotted against the percentage of hardwood in the pulp from which the paper was made. But suppose only those samples with pulp values between 1 and 9 had been measured. The figure shows that a straight line would provide quite a good fit to this subset of the data. For new samples of paper, with pulp values lying between 1 and 9, quite good prediction of the strength could legitimately be expected. But the figure also shows, strikingly clearly, that our model would produce predictions that were seriously amiss if we used it to predict the strength of paper with pulp values greater than 9. Only within the bounds of our data is the model trustworthy. These examples are particularly clear but they involve just a few data points and a single predictor variable. In data mining applications, with large data sets and many variables, things may not be so clear. Caution needs to be exercised when we make predictions.

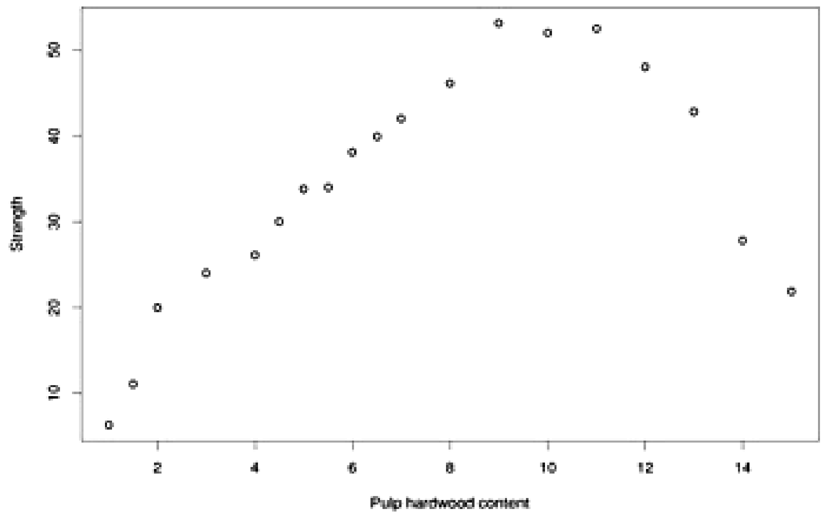


Figure 11.4: A plot of tensile strength of paper against the percentage of hardwood in the pulp.

11.3 Generalized Linear Models

Section 11.2 described the linear model, in which the response variable was decomposed into two parts: a weighted sum of the predictor variables and a random component: $Y(i) = \sum_j \alpha_j x_j(i) + \epsilon(i)$. For inferential purposes we also assumed that the $\epsilon(i)$ were independently distributed as $N(0, \sigma^2)$. We can write this another way, which permits convenient generalization, splitting the description of the model into three parts:

i. The Y (i) are independent random variables, with distribution N(μ(i), σ^2).

ii. The parameters enter the model in a linear way via the sum v(i) = ∑ $a_j x_j$(i).

iii. The v(i) and μ(i) are linked by v(i) = μ(i).

This permits two immediate generalizations, while retaining the advantages of the linear combination of the parameters. First, in (i) we can relax the requirement that the random variables follow a normal distribution. Second, we can generalize the link expressed in (iii), so that some other link function g(μ(i)) = v(i) relates the parameter of the distribution to the linear term v(i) = ∑$a_j x_j$(i). These extensions result in what are called generalized linear models. They are one of the most important advances in data analysis of the last two decades. As we shall see, such models can also be regarded as fundamental components of feed forward neural networks.

One of the most important kinds of generalized linear model for data mining is logistic regression. We have already encountered this in the form of logistic discrimination, but we describe it in rather more detail here, and use it as an illustration of the ideas underlying generalized linear models. In many situations the response variable is not continuous, as we have assumed above, but is a proportion: the number of flies from a given sample that die when exposed to an insecticide, the proportion of questions people get correct in a test, the proportion of oranges in a carton that are rotten. The extreme of this arises when the proportion is out of 1, that is, the observed response is binary: whether or not an individual insect dies, whether or not a person gets a particular one of the questions right, whether or not an individual orange is rotten. We are now dealing with a binary response variable, with the random variable taking values 0 or 1 corresponding to the two possible outcomes. We will assume that the probability that the ith individual yields the value 1 is p(i), and that the responses of different individuals are independent. This means that the response for the ith individual follows a Bernoulli distribution:

$$p\Big(Y(i) = y(i)\Big) = p(i)^{y(i)}\left(1 - p(i)\right)^{1-y(i)},$$

where here y(i) Є {0, 1}. For logistic regression, this is the generalization of (i) above: the Bernoulli distribution is replacing the normal distribution.

Our aim is to formulate a model for the probability that an object with predictor vector **x** will take value 1. That is, we want a model for the mean value of the response, the probability p(y = 1|**x**). We could use a linear model a weighted sum of the predictor variables. However, this would not be ideal. Most obviously, a linear model can take values less than 0 and greater than 1 (if the **x** values are extreme enough). This suggests that we need to modify the model to include a nonlinear aspect. We achieve this by transforming the probability, nonlinearly, so that it can be modeled by a linear combination. That is, we use a nonlinear link function in (iii). A suitable function (not the only possible one) is a logistic (or logit) link function, in which

$$g\left(p(y=1|\mathbf{x})\right) = \log \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})},$$

where g (p(y = 1|**x**)) is modeled as $\sum a_j x_j$. As p varies from 0 to 1, log(p/1 - p) clearly varies from $-\infty$ to ∞, matching the potential range of $g(p) = \sum a_j x_j(i)$. One of the advantages of the logistic link function over alternatives is that it permits convenient interpretation. For example:

- The ratio $\frac{\mathbf{p(y=1|x)}}{\mathbf{1-p(y=1|x)}}$ in the transformation is the familiar odds that a 1 will be observed and log $\frac{\mathbf{p(y=1|x)}}{\mathbf{1-p(y=1|x)}}$ is the log odds.
- Given a new vector of predictor variables **x** = (x_1, ..., x_p), the predicted probability of observing a 1 is derived from $\frac{\mathbf{\log(p(y=1|x))}}{\mathbf{1-p(y=1|x)}}$ The effect on this of changing the jth predictor variable by one unit is simply α_j. Thus the coefficients tell us the difference in log odds or, equivalently, the log odd ratio resulting from the two values. From this it is easy to see that e^{α}, is the factor by which the odds changes when the jth predictor variable changes by one unit.

Generalized linear models thus have three main features:

i. The Y (i), i = 1, ..., n, are independent random variables, with the same exponential family distribution (see below).

ii. The predictor variables are combined in a form v(i) = $\sum a_j x_j$ (i), called the linear predictor, where the a_js are estimates of the α_js.

iii. The mean μ(i) of the distribution for a given predictor vector is related to the linear combination in (ii) through the link function

$$g(\mu(i)) = v(i) = \sum a_j x_j(i)$$

.

The exponential family of distributions is an important family that includes the normal, the Poisson, the Bernoulli, and the binomial distributions. Members of this family can be expressed in the general form

$$f(y;\theta,\phi) = e^{\dagger}(((y^{\dagger}\theta - b(\theta))/(\alpha(\phi) + c(y,\emptyset)))$$

If ϕ is known, then θ is called the natural or canonical parameter. When, as is often the case, α(ϕ) = ϕ , ϕ is called the dispersion or scale parameter. A little algebra reveals that the mean of this distribution is given by b' (θ) and variance by α(ϕ)b''(θ). Note that the variance is related to the mean via b''(θ), and this, expressed in the form V(θ), is sometimes called the variance function. In the model as described in (i) to (iii) above, there are no restrictions on the link function. However (and this is where the exponential family comes in), things simplify if the link function is chosen to be the function expressing the canonical parameter for the distribution being used as a linear sum. For multiple regression this is simply the identity distribution and for logistic regression it is the logistic transformation presented above. For Poisson regression, in which the distribution in (i) is the Poisson distribution, the canonical link is the log link g(u) = log (u).

Prediction from a generalized linear model requires the inversion of the relationship g(μ(i)) =$\sum a_j x_j$(i). The algorithms in least squares estimation were very straightforward, essentially involving only matrix inversion. For generalized linear models, however, things are more complicated: the non -linearity means that an iterative scheme has to be adopted. We will not go into details of the mathematics here, but it is not difficult to show that the maximum likelihood solution is given by solving the equations

$$\sum [(x_1 j\,(y(i) - \mu(i))/(a_1 i\,(\phi)V(\mu(i)g^{\dagger\prime}(\mu(i)) = 0, \qquad j = 1,\ldots,p,]$$

where the i indices for α_i(f) and μ(i) are in recognition of the fact that these vary from data point to data point. Standard application of the Newton-Raphson method leads to iteration of the equations

$$\mathbf{a}^{(s)} = \mathbf{a}^{(s-1)} - \mathbf{M}_{s-1}^{-1}\mathbf{u}_{s-1},$$

where $\mathbf{a}^{(s)}$ represents the vector of values of (a_1, ..., a_p) at the sth iteration, $\mathbf{u}_{s-1}$ is the vector of first derivatives of the log likelihood, evaluated at $\mathbf{a}^{(s-1)}$, and $\mathbf{M}_{s-1}$ is the matrix of second derivatives of the log likelihood, again evaluated at $\mathbf{a}^{(s-1)}$.

11.4 Artificial Neural Networks

Artificial neural networks (ANNs) are one of a class of highly parameterized statistical models that have attracted considerable attention in recent years (other such models are outlined in later sections). In the present context, we will be concerned only with feed- forward neural networks or multilayer perceptrons. In this section, we can barely scratch the surface of this topic, and suitable further reading is suggested below. The fact that ANNs are highly parameterized makes them very flexible, so that they can accurately model relatively small irregularities in functions. On the other hand, as we have noted before, such flexibility means that there is a serious danger of over fitting. Indeed, early work was characterized by inflated claims when such networks were overfitted to training set s, with predictions of future performance being based on the training set performance. In recent years strategies have been developed for overcoming this problem, resulting in a very powerful class of predictive models.

To set ANNs in context, recall that the generalized linear models of the previous section formed a linear combination of the predictor variables, and transformed this via a nonlinear transformation. Feed forward ANNs adopt this as the basic element. However, instead of using just one such element, they use multiple layers of many such elements. The outputs from one layer the transformed linear combinations from each basic element serve as inputs to the next layer. In this next layer the inputs are combined in exactly the same way each element forms a weighted sum that is then non - linearly transformed. Mathematically, for a network with just one layer of transformations between the input variables x and the output y (one hidden layer), we have

$$y = \sum_k w_k^{(2)} f_k \left(\sum_j w_j^{(1)} x_j \right) .$$

NOTES

Here the w is the weights in the linear combinations and the f_Ks are the non -linear transformations. The nonlinearity of these transformations is essential, since otherwise the model reduces to a nested series of linear combinations of linear combinations which is simply a linear combination. The term network derives from a graphical representation of this structure in which the predictor variables and each weighted sum are nodes, with edges connecting the terms in the summation to the node.

There is no limit to the number of layers that can be used, though it can be proven that a single hidden layer (with enough nodes in that layer) is sufficient to model any continuous functions. Of course, the practicality of this will depend on the available data, and it might be convenient for other reasons (such as interpretability) to use more than one hidden layer. There are also generalizations, in which layers are skipped, with inputs to a node coming not only from the layer immediately preceding it but also from other preceding layers.

The earliest forms of ANN used threshold logic units as the nonlinear transformations: the output was 0 if the weighted sum of inputs was below some threshold and 1 other wise. However, there are mathematical advantages to be gained by adopting differentiable forms for these functions. In applications, the two most common forms are logistic $f(x) = e^X/(1 + e^X)$ and hyperbolic tangent $f(x) = \tanh(x)$ transformations of the weighted sums.

We saw, when we moved from simple linear models to generalized linear models, that estimating the parameters became more complicated. A further extra level of complication occurs when we move from generalized linear models to ANNs. This will probably not come as a surprise, given the number of parameters (these now being the weights in the linear combinations) in the model and the fundamental nonlinearity of the transformations. As a consequence of this, neural network models can be slow to train. This can limit their applicability in data mining problems involving large data sets. (But slow estimation and convergence is not all bad. There are stories within the ANN folklore relating how severe over fitting by a flexible model has been avoided by accident, simply because the estimation procedure was stopped early.) Various estimation algorithms have been proposed. A popular approach is to minimize the score function consisting of the sum of squared deviations (again!) between the output and predicted values by steepest descent on the weight parameters. This can be expressed as a sequence of steps in which the weights are

NOTES

updated, working from the output node(s) back to the input nodes. For this reason, the method is called back -propagation. Other criteria have also been used. When Y takes only two values the sum of squared deviations is rather unnatural (since, as we have seen, the sum of squared deviations arises as a score function naturally from the log -likelihood for normal distributions). A more natural score function, based on log-likelihood for Bernoulli data, is

$$\sum_i \left[y(i) \log \frac{\hat{y}(i)}{y(i)} - (1 - y(i)) \log \frac{(1 - \hat{y}(i))}{(1 - y(i))} \right] .$$

As it happens, in practical applications with reasonably sized data sets, the precise choice of criterion seems to make little difference. The vast amount of work on neural networks in recent years, which has been carried out by a diverse range of intellectual communities, has led to the rediscovery of many concepts and phenomena already well known and understood in other areas. It has also led to the introduction of unnecessary new terminology.

Nonetheless, research in this area has also led to several novel general forms of models that we have not discussed here. For example, radial basis function networks replace the typical logistic nonlinearity of feed forward net -works with a "bump" function (a radial basis function). An example would be a set of p-dimensional Gaussian bumps in **x** space, with specified widths. The output is approximated as a linear weighted combination of these bumps functions. Model training consists of estimating the locations, widths, and weights of the bumps, in a manner reminiscent of mixture models.

11.5 Other Highly Parameterized Models

The characterizing feature of neural networks is that they provide a very flexible model with which to approximate functions. Partly because of this power and flexibility, but probably also partly because of the appeal of their name with its implied promise, they have attracted a great deal of media attention. However, they are not the only class of flexible models. Others, in some cases with an approximating power equivalent to that of neural net -works, have also been developed. Some of these have advantages as far as interpretation and estimation goes. In this section we briefly outline two of the more important classes of flexibl e model.

11.5.1 Generalized Additive Models

We have seen how the generalized linear model extends the ideas of linear models. Yet further extension arises in the form of generalized additive models. These replace the simple weighted sums of the predictor variables by weighted sums of transformed versions of the predictor variables. To achieve greater flexibility, the relationships between the response variable and the predictor variables are estimated nonparametrically for example, by kernel or spline smoothing, so that the generalized linear model form $g(\mu(i)) = \sum \alpha_j x_j(i)$ becomes $g(\mu(i)) = \sum \alpha_j f_j(x_j(i))$. The right-hand side here is sometimes termed the additive predictor. Such models take to the nonparametric limit the idea of extending the scope of linear models by transforming the predictor variables. Generalized additive models of this form retain the merits of linear and generalized linear models. In particular, how g changes with any particular predictor variable does not depend on how other predictor variables change; interpretation is eased. Of course, this is at the cost of assuming that such an additive form does provide a good approximation to the "true" surface. The model can be readily generalized by including multiple predictor variables within individual f components of the sum, but this is at the cost of relaxing the simple additive interpretation. The additive form also means that we can examine each smoothed predictor variable separately, to see how well it fits the data.

In the special case in which g is the identity function, appropriate smoothing functions can be found by a backfitting algorithm. If the additive model $y(i) = \sum \alpha_j f_j(x_j(i)) + \epsilon(i)$ is correct, then

$$f_k(X_k) = E\left(Y - \sum_{j \neq k} \alpha_j f_j(X_j(i)) | X_k\right).$$

This leads to an iterative algorithm in which, at each step the "partial residuals" $y - \sum_{j \neq k} a_j f_j(x_j(i))$ for the k th predictor variable are smoothed, cycling through the predictor variables until the smoothed functions do not change. The precise details will, of course, depend on the choice of smoothing method: kernel, spline, or whatever.

To extend this from additive to generalized additive models, we make the same extension as above, where we extended the ideas from linear to generalized linear models. We have already outlined the iteratively weighted least squares algorithm for fitting generalized linear models. We showed that this was essentially an iteration of a weighted least squares solution applied to an "adjusted" response variable, defined by

NOTES

$$\sum_j x_j(i)a_j + (y(i) - \mu(i)) \frac{\partial \eta(i)}{\partial \mu(i)}$$

For generalized additive models, instead of the weighted linear regression we adopt an algorithm for fitting a weighted additive model.

11.5.2 Projection Pursuit Regression

Projection pursuit regression models can be proven to have the same ability to estimate arbitrary functions as neural networks, but they are not as widely used. This is perhaps unfortunate, since estimating their parameters can have advantages over the neural network situation. The additive models of the last section essentially focus on individual variables (albeit transformed versions of these). Such models can be extended so that each additive component involves several variables, but it is not clear how best to select such subsets. If the total number of available variables is large, then we may also be faced with a combinatorial explosion of possibilities. The basic projection pursuit regression model takes the form

$$Y = \alpha_0 + \sum f_k\left(\alpha_k^T \mathbf{X}\right) + \varepsilon.$$

This has obvious close similarities to the neural network model it is a linear combination of (potentially nonlinear) transformations of linear combinations of the raw variables. Here, however, the *f* functions are not constrained (as in neural networks) to take a particular form, but are usually found by smoothing, as in generalized additive models. This makes them a generalization of neural networks. Various forms of smoothing have been used, including spline methods, Friedman's "supersmoother" (which makes a local linear fit about the point where the smooth is required), and various polynomial functions. The term projection pursuit arises from the viewpoint that one is projecting **X** in direction a_k, and then seeking directions of projection that are optimal for some purpose. (In this case, optimal as components in a predictive model.) Various algorithms have been developed to estimate the parameters. In one, components of the sum are added sequentially up to some maximum value, and then sequentially dropped, each time selecting on the basis of least squares fit of the model to the data. For a given number of terms, the model is fitted using standard iterative procedures to estimate the parameters in the a_k vector. This fitting process is rather complex from a computational viewpoint, so that projection pursuit regression tends may not be practical for data sets that are massive (large n) and high-dimensional (large p).

12. Data Organization and Databases

Structure

Objective

After going through this lesson, you should be able to:

- Discuss about a index structures;
- Discuss relational databases and manipulating tables;
- Discuss structured query language;
- Discuss about a OLAP and string databases;
- Discuss about a massive data sets, data management, and data mining

12.1 Introduction

One of the features that distinguish data mining from other types of data analytic tasks is the quantity of data. In many data mining applications (such as Web log analysis for example) there may be millions of rows and thousands of columns in the standard form data matrix, so that questions of efficiency of data analysis algorithms are very important. An algorithm whose running time scales exponentially in the number n of rows may be unusable for all but the smallest data sets. Examples of operations that can be carried out in time O(n) or O(n log n) are counting simple frequencies from the data, finding the mode of a discrete variable or attribute, or sorting the data. Generally, such computations are feasible even for large data sets. However, even a linear time algorithm can be prohibitively costly to use if multiple passes through a data set are required.

If the number of rows n of a data set influences algorithm complexity, so also can the number of variables p. For some applications p is very small (less than 10, for example), but in others, like market basket analysis or analysis of text documents, we can encounter data sets with 10^5 or even 10^6 variables. In such situations we cannot use methods that involve, for example, operations as the $O(p^2)$ computation of pair wise measures of association for all pairs of attributes.

In any data analysis project it is useful to distinguish between two phases. The first is actually getting the data to the analysis algorithm, and the second is running the analysis method itself. The first phase might seem trivial, but it can often become the bottleneck. For example, in analyzing a set of data it may be necessary to apply an algorithm to many different subsets of the data. This means we have to be able to search and identify the members of each subset rapidly, and also to load that subset into main memory. Tree algorithms provide an obvious illustration of this, where the data set is progressively split into smaller subsets, each of which has to be identified before the tree can be extended. The purpose of data organization is to find methods for storing the data so that accessing subgroups of data is as fast as possible. Even in cases when all the data fit into main memory, data organization is important.

In addition to supporting efficient access to data for data mining algorithms, data organization plays an important role in the iterative and interactive nature of the overall data mining process. The aim of this chapter is to discuss briefly the memory hierarchy of modern computer and then present some index structures that database systems use to speed up the evaluation of queries. We then move to a discussion on relational databases and their query languages, as well as some special purpose database systems.

12.2 Memory Hierarchy

The memory of a computer is divided into several layers. These layers have different access times (where access time is the average time to retrieve a randomly selected byte of memory). Indeed, if disk storage were as fast as on-board cache, there would be no need to develop any sophisticated methods for data organization.

A general categorization of different memory structures is the following:

1. Registers of the processor. Typically there are fewer than 100 of these, and the processor can access data in the registers directly; that is, there is no slowdown associated with accessing a register.

2. On-processor or on-board cache. This is fast semiconductor memory implemented on the same chip as the processor or residing on the mother-board. Typical size is 16–1,000 kilobytes and access time is about 20 ns.

3. Main memory normal semiconductor memory, with sizes from 16 megabytes to several gigabytes, and access time about 70 ns.

4. Disk cache. Semiconductor memory implemented as an intermediate storage between main memory and disks.

5. Disk memory. Sizes vary from 1 gigabyte to hundreds or thousands of gigabytes for large arrays of disks. Typical access time is around 10 ms.

6. Magnetic tape. A magnetic tape can hold up to several gigabytes of data. Access time varies, but can be minutes.

The differences between the access times are truly large: in the 10 milliseconds needed for accessing a disk, we could perform up to a million accesses to fast cache. Another way to think about this is to pretend that access time is linearly proportional to actual distance. Thus, if we imagine main memory to be an effective distance of 1 meter away (within reach of your hand), the equivalent distance for disk memory is order of 10^5 times greater, i.e., 100 km!

Another major difference between main memory and disk is that individual bytes of main memory can be accessed, whereas for disk, whenever we access a byte, actually the whole disk page, about 4 kilobytes, containing that byte will be loaded to main memory. So if that page happens to contain information that can be used later, it will already be in fast memory. As an example, if we want to retrieve 1,000 integers, each taking 4 bytes to store, this can take between 1 and 1,000 disk accesses, depending on whether the integers are all stored in the same disk page or each on a page of their own.

The physical properties of the memory hierarchy lead to the following rules of thumb:

- If possible, data should be in main memory.
- In main memory, data items that are used together should be logically close to each other (that is, we should quickly be able to find the next element of a subset).
- On disk, data items that are used together should be also physically close to each other (that is, on the same disk page, if possible).

In practice, the user of a system typically has little control over the details of the way the data are placed in caches, or over the actual physical layout of data on disk. Normally, the systems try to load as much data as possible into main memory, and decide on their own how to deal the data objects onto disk pages. The user can influence the kinds of auxiliary structures that are created to access subgroups of the data.

12.3 Index Structures

A primary goal of data organization is to find ways of quickly locating all the data points that satisfy a given selection condition. Usually the selection condition is a conjunction of conditions on individual attributes, such as "Age ≤ 40" and "Income ≤ 20,000." We consider first data structures that are especially applicable to situations in which there is only one conjunct.

An index on an attribute A is a data structure that makes it possible to locate points with a given value of A more efficiently than by a sequential scan of the whole data set. Indices are typically built either by the use of B*-trees or by the use of hash functions.

12.3.1 B-trees

A search tree is probably the simplest index structure. Suppose we have a set S of data vectors {$\mathbf{x}(1)$, ..., $\mathbf{x}(n)$}, and that we want to find all points having a particular value of an ordinal attribute (variable) A as quickly as we can. A search tree is a binary tree structure such that each node has a value of A stored into it, and each leaf has a pointer to an element of S. Moreover, the tree is structured so that all elements of S pointed to by leaves from the left subtree of a node u containing value a will have values of A which are less than or equal to a. Likewise, all elements of S pointed to by leaves in the right subtree of u have values for A that are greater than a.

Given a binary search tree for an attribute A, it is easy to find the data points from S that have a given value b for A. We simply start from the root of the tree, selecting the left or the right subtree by comparing b against the values stored in the nodes. When we get to a leaf, either we find a pointer to the record(s) with A = b, or we find that no such pointer exists.

It is also easy to find all the points from S that satisfy the condition b ≤ A ≤ c, a so-called "interval query." Simply locate the leaf where b should be (as above), locate the leaf where c should be, and the desired records are pointed to by the leaves between these two positions.

The time needed for finding the records with a given value for attribute A is proportional to the height of the tree plus the number of such records. In the worst case, the height of the tree is n, the number of points in the set S, but there are ways of ensuring that the height of the tree will be O(log n) (although they are beyond the scope of this text). In practice, binary search trees are relatively seldom used, since B*-trees, discussed below, are clearly superior for accessing data on a disk.

The basic idea for B*-trees are the same as for search trees: the pointers to the data objects are in the leaves of the tree, and interior nodes contain values of the attribute A that indicate where certain pointers are to be found. However, instead of having two children and one value for A per interior node, a B*-tree typically has hundreds of children and values.

In more detail, a B*-tree of degree M for set of values is a tree where

- all leaves are at the same depth;
- each leaf contains between M/2 and M keys (possible target values);
- each interior node (except possibly the root) has K children C_1, ..., C_K, where M/2 ≤ K ≤ M and K - 1 values a_1, ..., a_{K-1}; for all

NOTES

i, all the key values stored in the leaves of subtree C_i are larger than a_{i-1} and at most as large as a_i.

Searching from a B*-tree is carried out in the same way as from a binary search tree: for each interior node of the tree, the values a_i are used to select the correct subtree.

A B*-tree differs from the basic binary search tree in that the height is guaranteed to be O(log n), since all leaves are on the same depth. Actually, the depth of the tree is bounded by $\log_{M/2} n$. Typically, the value of M is selected so that each node of the tree fits into a single disk page. If M is 100, then $(M/2)^5$ is over 300 million, and we find that for most realistic values of n, the number of elements in the set, the tree will have at most five levels: This means that finding a data point from 300 million points on the basis of the value of a single attribute can be done in three disk accesses, as the root node and the second level of the tree can be held in main memory. Most database management systems use B*-tree structures as one of their index structures.

12.3.2 Hash Indices

Suppose again that we have a set S of data points, and that we want to find all points such that attribute A has value a. If the set of possible values of A is small, we can do the following: for each possible value, construct a list of pointers to the data points with that value for A. Then, given the query "Find the points with A = a," we need only to access the list for a. This method is not feasible, however, if there is a large number of potential values for A: we cannot maintain a list for each of the possible 2^{32} integers which can be represented by 32 bits, for example. What we can do is to apply a transformation to the A-values so as to reduce the range of possible values. In more detail, let Dom(A) be the set of possible values of A. A hash function is a function h from Dom(A) to {1, ..., M}, where M is the size of the hash table r. For each j Є {1, ..., M} we store into r[j] a list of pointers to those records $\mathbf{x}_i$ in S whose A value a_i satisfies $h(a_i) = j$. When we want to find all the data points with A = a, we simply compute h(a), go to location r[h(a)] and traverse the list of data points, for each of them checking whether the value of A really was a, or whether it was another value b with the property that h(b) = h(a) (this is called a collision).

A typical hash function is a mod M, when M is chosen to be suitable prime larger than n, the number of data points. If the hash function is well chosen and the hash table is sufficiently large, collisions are rare, and searching for the points with a given A value can be done in time essentially proportional to the number of such points. Hash indices, however, do not directly support interval queries.

12.4 Multidimensional Indexing

Traditional index structures such as hashing and B*-trees provide fast access to rows of tables on the basis of values of a given attribute or collection of attributes. In some applications, however, it is necessary to express selection conditions on the basis of several attributes, and normal index structures do not help. Consider, for example, the case of storing geographic information about cities. Suppose, for example, we wish to find all the cities with latitude between 30 N and 40 N, longitude between 60 W and 70 W, and population at least 1,000. Such a query is called a rectangular range query.

Suppose the cities table is large, containing millions of city names. How should the query be evaluated? A B*-tree index on the latitude attribute makes it possible to find the cities that satisfy the conditions for that attribute, but for finding the rows that satisfy the conditions on longitude among these, we have to resort to a sequential scan. Similarly, an index on longitude does not help much. What is needed is an index structure that makes it possible to use directly the conditions on both attributes.

Multidimensional indexing refers to techniques for finding rows of tables on the basis of conditions on multiple attributes. One of the widely used methods is the R*-tree. Each node in the tree corresponds to a region in the underlying space, and the node represents the points within that region. For dimensions up to about 10, the multidimensional index structures speed up searches on large databases. Fast evaluation of range queries for data sets with larger numbers of dimensions (e.g., in the 100s) is still an open problem.

12.5 Relational Databases

In data mining we often need to access a particular subset of the data and compute a function from the values of certain attributes on that subset. We have discussed some data structures that can help in finding the relevant data points quickly. Relational databases provide a unified mechanism for fast access to selected

NOTES

parts of the data.

In database terminology, a data model is a set of constructs that can be used to describe the structure of data, plus a set of operations for manipulating the data. (Note that this use of the word model is rather different from that given earlier in the book. Here it is a structure imposed on the data by design, rather than a structure discovered existing within the data. The dual use of the word model is perhaps unfortunate, and arises because of the different disciplines that have contributed to data mining; in this case, statistics and database theory. Fortunately, confusion seldom arises; which of the two meanings is intended will generally be clear from the context). The relational data model is based on the idea of representing data in tabular form. A table header (schema) consists of the table name and a set of named columns; the column names are also called attributes. The actual table (an instance of the schema), also called a relation, is a named set of rows. Each table entry in the column for attribute A is a value from the domain Dom(A) of A. Note that when the attributes are defined, the domain of each must also be specified. An attribute can be of any data type: categorical, numeric, etc. The order of the row and columns in a table is not significant.

We can put this more formally. A relation schema R is a set of attributes $\{A_1, ..., A_p\}$, where each attribute A_j has an associated domain $Dom(A_j)$. A row over the schema R is a mapping $t : R \rightarrow \cup_i Dom(A_j)$ where $t(A_j) \in Dom(A_j)$. A table or relation over the schema R is a collection of rows over R. A relational database schema **R** is a collection $\{R_1, ..., R_k\}$ of relation schemas (with possibly some constraints on the relation instances), and a relational database **r** over the schema **R** consists of a relation over R_i, for each $i = 1, ..., k$.

Thus the relational data model is based on the idea of tabular representation. The values in the cells may be arbitrary atomic values, such as real numbers, integers, or strings; sets or lists of values are not allowed. This means that, if, for example, we want to represent information about people, their ages, and phone numbers, we cannot store multiple phone numbers in one attribute. If restricted in this way, the model is said to have first normal form.

The relational model is widely used in data management, and virtually all major database systems are based on it. Some systems provide additional functionality, such as the possibility of using object-oriented data modeling methods.

Even in relatively small organizations, relational databases can have hundreds of tables and thousands of attributes. Managing the schema of the database can, therefore, be a complicated task. Sometimes it is claimed that for data analysis purposes it suffices to combine all the tables into a massive observation matrix, or "universal table," and that therefore in data mining one does not have to care about the fact that the data are in a database. However, an examination of simple examples shows that this is not feasible: the universal table would be so large that operations on it would be prohibitively costly.

12.6 Manipulating Tables

Being able to describe the structure of data and to store data using this structure is not sufficient in itself for data management: we also must be able to retrieve data from the database. We briefly describe two languages for manipulating collections of tables (that is, relational databases): relational algebra, in this section, and the Structured Query Language (SQL), in the next. Relational algebra is based on set-theoretic notation and is quite handy for theoretical purposes, while SQL is widely used in practice.

In the examples, we use r, s, etc. to refer to tables, and R, S, etc. to refer to the sets of attributes for those tables.

Relational algebra contains a set of basic operations for manipulating data given in tabular form, and several derived operations (operations that can be expressed as a sequence of basic operations) are also used. The operations include the three set operations union, intersection, and difference and the projection operation for removing columns, the selection operation for selecting rows, and the join and Cartesian product operations for combining rows from two tables.

Set Operations

Tables are sets of rows, and all operations in the relational algebra are set-oriented: they take sets as arguments and produce a set as their result. This makes it possible to compose relational queries: the results of a query are relations, as are the arguments. Conventional set operations are useful for manipulating tables. We shall include union, intersection, and difference (denoted by $r \cup s$, $r \cap s$, and $r \setminus s$, respectively) as the basic operations in relational algebra. The union operation combines two tables over the same set of attributes: the result r U s contains all the rows that occur in r or s. The intersection operation r ∩ s results in the table containing those rows that occur in r and in s. The difference operation r \ s gives the rows that occur in r but not in s. These operations all assume that r and s are tables over the same set of attributes.

As an example, suppose r is a table representing the prices of all soft drinks, and s is a table representing the prices of all products costing at most $2.00. Then r U s is the table of all soft drinks and products costing less than $2.00, r ∩ s is the table of all soft drinks costing less than $2.00, and r \ s contains one row for each soft drink that does not cost less than $2.00, i.e. that costs at least $2.00. The intersection operation could, of course, be defined using the union and difference operations: r ∩ s = (r U s) \ ((r \ s) U (s \ r)).

Care must be taken to ensure that the resulting set is a table, in the sense that it has a schema. Therefore r U s, r ∩ s and r \ s are defined only if r and s are tables over the same schema—that is, over the same set of attributes.

Intersection queries can be used in construction of rule sets, for example. Suppose, we have computed a table r corresponding to the observations that satisfy a condition F, and similarly another table s that corresponds to the observations satisfying condition G. The intersection r ∩ s corresponds to those observations that satisfy both conditions; the cardinality of the intersection tells what the overlap between the conditions is. If r and s are computed from the same base table of observations, we can also achieve the same effect by using the conjunction F Λ G as the selection condition in the query. Intersection queries occur most naturally in situations in which we need to check whether the same value occurs in two tables.

Projection

The purpose of the projection operation is to trim a table so that only the data in specific columns of interest remain. Given a table r with attributes R, and X ⊂ R, the projection of r on X is obtained by removing from the table all the columns outside X. A side effect of projecting a table is that the number of rows, as well as the number of columns, may decrease. If the argument table over R is projected on a set of attributes X, and if table r over R contains two rows that agree on the X attributes, but differ on some attribute in

R\ X, the projected rows would be identical. Such identical rows are commonly called duplicates. Since tables are sets, they cannot contain duplicates, and only one representative of each duplicate is retained. Because this feature is implicit in the concept of a set, it does not show up in the definition of the projection operation.

Commercial database systems often differ from the pure relational model on this point. In real implementations, tables are stored as files. Files, of course, can contain several identical records. Checking the uniqueness of records could take a lot of time. It is therefore customary that tables in commercial database management systems can contain duplicates.

The projection operation in relational databases is related to but not identical to the projection encountered in vector spaces. Both operations take points (called rows in databases) and produce points in a lower-dimensional space (rows with fewer attributes). In relational databases, we can project only to subspaces defined directly by the attributes; for vector spaces, projection can be defined for any subspace (that is, any linear combination of basis vectors (here attributes)).

Selection

The selection operation is used to select rows from a table. Given a Boolean condition F on the rows of a table r, the selection operation s_F applied to r yields the table s_F (r) consisting of those rows of r that satisfy the condition.

Selection is probably the most frequently used operation of the relational algebra: each time we want to focus on a particular row or subset of rows in a table, we need to use selection. Selection occurs often in the implementation of data mining algorithms. For example, in building a decision tree we want a list of the observations that belong to a particular node of the tree. This set of observations is exactly the answer to a selection query, where the selection condition is the conjunction of the conditions appearing in the nodes from the root of the tree to the node in question. Similarly, if we want to implement association rule algorithms using the relational algebra, one has to execute several selection queries; each one that looks at the subset of observations satisfying the condition that each variable in a candidate frequent set has value 1.

In pure relational algebra, selections are based on exact equalities

NOTES

or inequalities. For data mining, we often need concepts of inexact or approximate matching. If a predicate match for approximate matching between attribute values is available, we can (at least in some database systems) use that directly in database operations to select rows that satisfy the approximate matching condition.

12.7 The Structured Query Language (SQL)

Relational algebra is a useful and compact notation. In database management systems, SQL is the standard adopted by most database management system vendors. SQL implements a superset of the relational algebra. Here we introduce only the basic structure of SQL programs.

The basic statement of SQL is the "select-from -where" expression or query, which has the form

select	$A_1, A_2, \ldots, A_p$
from	$r_1, r_2, \ldots, r_k$
where	list of conditi ons

Here each r_i is a table, and each A_j is an attribute. The intuitive meaning is that for each possible choice of rows $t_1, \ldots, t_k$ from the tables $r_1, \ldots, r_k$, we test whether the conditions are true. If they are, a row consisting of the values of the attributes A_j is output.

The second line of the query, the from clause, specifies the tables to which the SQL statement is applied. The third line, the where clause, specifies the conditions that the rows in those tables must satisfy to be accepted into the result of the statement. The first line, the select clause, then specifies which attributes of the participating tables should appear in the result. It corresponds to the projection operation of relational algebra (not the selection operation). The "where" clause is used for representing the selection conditions occurring in the selection and the join operations. For a selection operation, the selection conditions are simply listed in the list of conditions of the where clause, separated by the keywords **and, or**, and **not**.

If some tables in the "from" clause have common attributes, the attribute names must be prefixed by a dot and the name of the table when they appear in the "select" clause or "where" clause. If all attributes of participating tables should appear in the result, the list of attributes in the "select" clause can be replaced by a star.

Aggregation in database queries refers to the combination of several values into one, by the sum or maximum operators, for example. Relational algebra does not have operations for aggregation, but SQL does. An aggregate is in general a quantity computed from the database whose value depends on several rows of the database.

SQL was developed for traditional database applications such as generating reports and concurrent access and updating of transaction data by many users in real-time. Thus, it is not a big surprise that the language as such does not provide a very good platform for implementing data mining algorithms. There are two reasons for this: lack of suitable primitives and the need for efficiency.

Regarding the primitives, in SQL it is quite easy to do counting and aggregation. Therefore, for example, the operations needed for association rule algorithms are straightforward to implement by accessing the data using SQL. For building decision trees we need to be able to count the number of observations that fulfill the conditions occurring in the tree nodes from the root to the node in question. This is possible to do by selection and count queries. Where the primitives of SQL fail is in common statistical operations, such as matrix inversion, singular value decomposition (SVD), and so forth.

Such operations would be extremely cumbersome to implement using SQL. This means that fitting complicated models is usually carried out outside the database system.

Even in cases when the SQL primitives are sufficient for expressing the operations in the data mining algorithm, there are reasons to implement the algorithm in a loosely-coupled manner, i.e., by downloading the relevant data to the algorithm. The reason is that the connection between a database management system and an application program typically enforces a large overhead for each query. Thus, while it is quite elegant to express the basic operations of association rule algorithms (for example) using SQL, such an implementation would typically be fairly slow. An additional cause for performance problems is that in association rule algorithms (for example) we must compute the frequency of a large number of candidate frequent sets. In a specialized implementation it is easy to do many of these counting operations in one pass through the data, whereas in an implementation based on using an SQL database management system, each candidate frequent set would cause a separate query to be issued.

12.8 Query Execution and Optimization

A query can be evaluated in various different ways. Consider, for example, the query

select t.product
from baskets t, baskets u
where t.transaction = u.transaction
        and u.product = "beer"

Here the notation baskets t, baskets u means that, in the query, t and u refer to rows of the baskets table. The notation is needed because we want to be able to refer to two different rows of the same table. The query finds all the products that have been bought in a transaction that also included beer.

The trivial method for evaluating such a query would be to try all possible pairs of rows from the baskets table, to check whether they agree on the basket -id attribute, and to test that the second row has "beer" in the product attribute. This would require n^2 operations on rows, where n is the size of the baskets table.

A more efficient method is to first locate the rows from the baskets table that have "beer" in the product attribute and sort the basket-ids of those rows into a list L. Then we can sort the baskets table using the basket -id attribute as the sort key and extract the products from the rows whose basket -id appears in the list L. Assuming that L is a relatively short list, this approach requires O(n) operations for finding the rows with beer, O(n log n) operations for sorting the rows, and O (n) operations for scanning the sorted list and selecting the correct values; i.e., altogether O(n log n) operations are needed. This is a clear improvement over the $O(n^2)$ operations needed for the naive method.

Query optimization is the task of finding the best possible evaluation method for a given query. Typically, query optimizers translate the SQL query into an expression tree, where the leaves represent tables and the internal nodes represent operations on the children of the nodes. Next, algebraic equalities between operations can be used to transform the tree into an equivalent form that is faster to evaluate. In the previous example, we have used the equation $\sigma_F(r \leftrightarrow s) = \sigma_F(r) \leftrightarrow s$, where F is a selection condition that concerns only the attributes of r. After a suitable expression tree is found, evaluation methods for each of the operations are selected. For example, a join operation can be evaluated in several different ways: by nested loops (as in the trivial method above), by sorting, or by using indices. The efficiency of each method depends on the sizes of the tables and the distribution of the values in the tables. Thus, query optimizers keep information about such changing quantities to find a good evaluation method. Theoretically, finding the best evaluation strategy for a given query is an NP -hard problem, so that finding the best method is not feasible. However, good query optimizers can be surprisingly effective.

Database management systems strive to provide good performance for a wide variety of queries. Thus, while for a single query it might be possible to write a program that computes the result more efficiently than a database management system would compute it, the strength of databases is that they provide fast execution for most of the queries. In data mining applications this is useful, as the queries are typically not known in advance (for example, in decision tree construction).

12.9 Data Warehousing and Online Analytical Processing (OLAP)

A retail database, with information about customers, transactions, products, prices, etc., is a typical example of an operational database: the database is used to conduct the daily operations of the organization, and the operations can rely quite heavily on it. Other examples of operational databases include airline reservation systems, bank account databases, etc. Strategic databases are databases that are used in decision making in the organization. The decision support viewpoint is quite closely aligned with the goal of data mining. Indeed one could say that a major goal of data mining is decision support.

Typically, an organization has several different operational databases. For example, a retail outlet might have a database about market baskets, a warehouse system, a customer database (or several), a payroll database, a database about suppliers, etc. Indeed, a diversified service company might even have several customer databases. Altogether, large organizations can have tens or hundreds of different operational databases. For decision support purposes one needs to combine information from various

NOTES

operational databases to find out overall patterns of activity within the company and with its customers. Building decision support applications that directly access the operational databases can be quite difficult.

Operational databases such as our hypothetical retail database, any customer database, or the reservation system of an airline, are most often used to answer well-defined and repetitive queries such as "What is the total price of the products in this basket," "What is the address of customer Smith," or "What is the balance of account 123456?" Such databases have to support a large number of transactions consisting of simple queries and updates on the contents of the data. This type of database usage is called online transaction processing (OLTP).

Decision support tasks require different types of queries: aggregation is far more important. A typical decision support query might be "Find the sales of all products by region and by month, and the difference compared to last year." The term online analytical processing (OLAP) refers to the use of databases for obtaining summaries of the data, with aggregation as the principal mechanism.

OLTP and OLAP pose different requirements on the database management system. OLTP requires that the data are completely up to date, allows the queries to modify the database, allow several transactions to execute concurrently without interfering with each other, requires that responses be fast, and so forth. However, the OLTP queries and updates themselves are relatively simple. In contrast, in OLAP the queries can be quite complex, but normally only one of them executes at a given time. OLAP queries do not modify the data, and in finding out facts about last year's sales it is not crucial to have today's sale information. The requirements are so different that it makes sense to use different types of storage organizations for handling the two applications.

A data warehouse e is a database system used to store information from various operational databases for decision support purposes. A data warehouse for a retailer might include information from a market basket database, a supplier database, customer databases, etc. The data in the payroll database might not be in the data warehouse if they are not considered to be crucial in decision support. A data warehouse is not created just by dumping the data from various databases to a single disk. Several integration tasks have to be carried out, such as resolving possible inconsistencies between attribute names and usages, finding out the semantics of

NOTES

attributes and values, and so on. Building data warehouses is often an expensive operation, as it requires much manual intervention and a detailed understanding of the operational databases.

The difference between OLTP, OLAP, and data mining is not always clear cut. We can in fact see a continuum of queries: find the address of a customer; find the sales of this product in the last month; find the sales of all products by region and month; find the trends in the sales; find what products have similar sales patterns; find rules that predict the sale of a certain product customer segmentation/clustering. The first query is typically carried out by using an OLTP query, the second is a typical OLAP query, and the last two might be called data mining queries. But it is difficult to define exactly where data mining starts and OLAP ends.

12.10 Data Structures for OLAP

OLAP requires the computation of various aggregates from large base tables. Since many aggregates will be needed over and over again, it makes sense to store some of them. The data cube is a clever technique for viewing the results of various aggregations in a tabular way.

The previous example showed the sales table with the schema

sales(product,store,date,amount).

A possible row from this table might be

sales(red wine, store 1, August 25, 17.25),

indicating that the sales of red wine at store number 1 on August 25 were $17.25. Inventing a new value **all** to stand for any product, we might consider rows like

sales(**all**, store 1, August 25, 14214.70),

with the intended meaning that the total sales of all products in store 1 on August 25 were $14,214.70. In statistical terms, this gives us the marginal of the table, summing over values of the first attribute.

The data cube for the sales table contains all rows

sales(a, b, c, d),

NOTES

where a, b, and c, are either values from the domains of the corresponding attributes or the specific value **all**, and d is the corresponding sum. That is, the data cube consists of the raw table and all marginal tables: the one -dimensional ones, the two -dimensional ones, and so on up to those obtained by summing over each attribute individually.

12.11 String Databases

Interest in text and string -oriented databases has increased dramatically in recent years. Molecular biology is one of the reasons: modern biotechnology generates huge amounts of protein and DNA data sets that are often recorded as strings. Even more important has been the rise of the Web: search engines require efficient methods for finding documents that include a given set of terms. Relational databases are fine for storing data in a tabular form, but they are not well suited for representing and accessing large volumes of text. Recently, several commercial database systems have added support for the efficient querying of large text data fields.

Given a large collection of text, a typical query might be "find all occurrences of the word mining in the text." More generally, the problem is to find occurrences of a pattern P in a text T . The pattern P might be a simple string, a string with wildcards, or even a regular expression. The occurrence of P in T might be defined as an exact match or an approximate match, where errors are allowed.

The occurrences of the pattern P in text T can obviously be found by sequentially scanning the text and for each position testing whether P matches or not. Much more efficient solutions exist, however. For example, using the suffix tree data structure we can find the list of all occurrences of pattern p in time that is proportional to the length of p (and not dependent on the size of the text), and outputting the occurrences of p can be done in time $O(|p| + L)$, where L is the number of occurrences of p in the text. The suffix tree can be constructed in linear time in the size of the original text, and it is fast also in practice.

Schematically, a Web search engine might have two data structures: a relational table pages (page -address, page -text) and a suffix tree containing all the text of all the documents loaded into the system. When a user issues a query such as "find all documents containing the words data and mining," the suffix tree is used to find two lists pages: those containing the word data and those containing mining. Assuming the lists are sorted, it is straightforward to find the documents containing both words. Note, however, that the number of documents containing both data and mining is probably much less than the number containing one of the terms.

12.12 Massive Data Sets, Data Management, and Data Mining

So far in this chapter we have focused on database technology in a general sense. An important question remains as to how data mining and database technology interact. Our discussion of this interaction will be relatively brief, since there is no consensus to date among researchers and practitioners as to any "best" approach in terms of handling the interaction between data mining algorithms and database technology. At issue is the following: many massive data sets are either already stored in relational databases or could be more effectively managed and accessed during a data mining project if they were converted into relational database form. On the other hand, most data mining algorithms focus on the modeling and optimization aspects of the problem and effectively assume the data reside in a flat file in main memory. If the data to be mined are primarily on disk, and/or stored in a relational format (perhaps with an SQL interface), how then should we approach the question of interfacing our data mining algorithm to the data? This is the issue of data management, which, is typically not addressed explicitly in most descriptions of data mining algorithms. And perhaps this is indeed the most flexible approach, since the solutions we adopt in practice will be a function of various application factors, such as the amount of data, the amount of available main memory, how often the algorithm will need to be rerun, and so forth. Nonetheless, we can identify a few general approaches to this problem, which we discuss below.

12.12.1 Force the Data into Main Memory

The most obvious approach, and one that practitioners have used for years, is to see whether the data can in fact be stored in main memory and (subsequently) accessed efficiently by the data mining algorithm. As main memory technology allows random access memory sizes to grow into the gigabyte range, this approach can be quite practical for many "medium -sized" data analysis applications. Of course there are other applications, e.g., those with hundreds of millions of complex transactions, where we cannot hope to ever load the data into main memory in the foreseeable future. In such cases we can hope to subselect parts of the data, perhaps by generating a random sample of records so that we have n' transactions instead of n to deal with (where n' is much smaller than n).

We could also select subsets of features in some manner. For example, one of the authors worked on a predictive modeling application involving on the order of 1,000 variables and 200,000 customers. Decision trees we re built on random samples of 5,000 customers, and the union of variables from the resulting trees was then used to build models (using trees, nonlinear regression, and other techniques) on the entire set of 200,000 records. This is of course an entirely heuristic procedure, and an important variable might have been omitted from the trees as a result of the multiple random sampling during model building. Nonetheless, this is a fairly typical example of the type of "data engineering" that is often required in practice to obtain meaningful results in a reasonable amount of time. Note also that generating a random sample from a relational database can itself be a nontrivial process. There are, of course, numerous refinements to the basic idea of random sampling, e.g., taking an initial small sample to gets a general idea of the "data landscape," then further refining this sample in some automated manner, and so forth.

Of course even if the data fit in main memory, we still must be careful. It may well be that we have to subsample the data even further to get our data mining algorithm to run in reasonable time. Furthermore, naive implementations of algorithms may create large internal data structures when they run (e.g., unnecessary copies of data matrices), which in turn may cause available memory to be exceeded. Thus, it goes without saying that efficient implementation from a memory and time viewpoint is still important, even when the data all reside in main memory.

12.12.2 Scalable Versions of Data Mining Algorithms

The term scalable is somewhat loosely used in the data mining literature, but we can think of it as referring to data mining algorithms that scale gracefully and predictably (e.g., linearly) as the number of records n implementation of a decision tree algorithm will exhibit a dramatic slowdown in run-time performance once n becomes large enough that the algorithm needs to frequently access data on disk. In practice, research on scalability focuses more on the large n problem than on the large p problem: large p is inherently more difficult than large n.

One line of investigation in scalable data mining algorithms is to develop special-purpose scalable implementations of existing well-known algorithms that are guaranteed to return the same result as the original (naive) implementation, but that typically will run much faster on large data sets. An example of this general approach is that of Gehrke et al. Who propose a family of algorithms called BOAT (Bootstrapped Optimistic Algorithm for Tree Construction). The BOAT approach uses two scans through the entire data set. In the first scan an "optimistic tree" is constructed using a small random sample from the full data (and that can fit in main memory). The second scan then takes care of any differences between the initial tree and the tree that would have been built using all of the data. The resulting tree is then the same tree that the naive algorithm would have constructed (in a potentially inefficient manner). The method involves various clever data structures to keep track of tree -node statistics. Gehrke et al report fitting classification trees to nine-dimensional synthetically generated data sets with 10 million data vectors in about 200 seconds.

A related strategy is to derive new approximate algorithms that inherently have desirable scaling performance by virtue of relying on various heuristics based on a relatively small number of linear scans of the data. These algorithms typically return "good" solutions but are not necessarily in agreement with the original "nonscalable" version of the algorithm. For example, scalable clustering algorithms of this nature are described by Bradley, Fayyad, and Reina and Zhang, Ramakrishnan, and Livny.

12.12.3 Special-Purpose Algorithms for Disk Access

Yet another approach to the problem of dealing with data on disk has been the development of new algorithms that are closely coupled with relational databases and transaction data. The search component of association rule algorithms takes advantage of the typical sparsity of transaction data sets (i.e., most customers purchase relatively few items per transaction). At a high level, the algorithms typically involve breadth-first search strategies, where each level of the tree involves a single scan of the data that can be executed relatively easily. Agrawal et al. report results on synthetic data involving 1,000 items and up to 10 million transactions. They empirically demonstrate that the runtime of their algorithm scales up linearly on these data sets as a function of the number of transactions. Similar results have since been reported on a wide range of sparse transaction data sets and many variations of the

NOTES

basic algorithm have been developed.

12.12.4 Pseudo Data Sets and Sufficient Statistics

Figure 12.1 illustrates another general idea that can be thought of as a generalization of random sampling. An approximate (and typically much smaller) data set is created that can then be accessed (e.g., in main memory) by the data mining algorithm instead of dealing with the full data (on disk). This general approach can, of course, only approximate the results we would have obtained had the algorithm been run on the full data. However, if the approximate data set is constructed in a clever enough manner, we can often get almost the same results on only a fraction of the data. It is often the case in practice that as part of the overall data mining process we will run our data mining algorithm many times, with different models, different variables, and so forth, in an exploratory manner, before finally settling on a final model. The use of an approximate data set for such exploratory modeling can be particularly useful (rather than having to deal with the full data set).

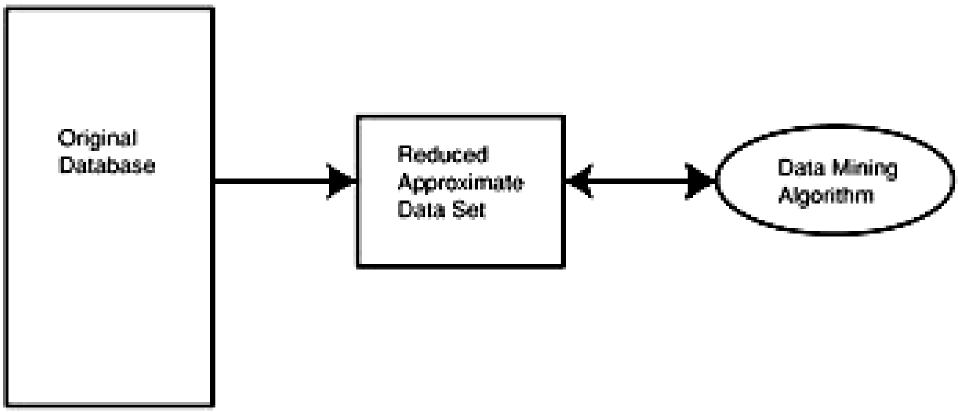


Figure 12.1: The concept of data mining algorithms which operate on an approximate version of the full data set.

In this general context Du Mouchel et al. propose a statistically motivated methodology for "data-squashing" which amounts to creating a set of n' weighted "pseudo" data points, where n' is much smaller than the original number n, and where the pseudo data points are automatically chosen by the algorithm to mimic the statistical structure of the original larger data set. The general idea is to approximate the structure of the likelihood function as closely as possible, even without the functional form of the model being used in the data mining algorithm being specified. The method was empirically demonstrated to provide significant reduction in prediction error on a logistic regression problem compared to simple random sampling of a data set .On a related theme, for some data sets it may be sufficient simply to store the original data via a more efficient data structure than as a flat file or multiple tables in a relational database. The A D-Tree data structure proposed by Moore and Lee provides an efficient mechanism for storing multivariate categorical data (i.e., counts). Data mining algorithms can then quickly access counts and related statistics from the AD-Tree much more quickly than if the algorithm had to access the original data. Computational speed- ups of 50 to 5,000-fold on various classification algorithms (compared to naive implementation of the algorithms) have been reported).

In conclusion, we see that many different techniques can be used to implement data mining algorithms that are efficient in both time and space when we deal with very large data sets. Indeed there are several other approaches we have not even mentioned here, including the use of online algorithms that see each data point only once (useful for applications where data are arriving rapidly in a continuous stream over time) and more hardware-oriented solutions such as parallel processing implementations of algorithms (in cases when both the algorithm and the data permit efficient parallel approaches). Choice of a particular technique often depends on quite practical aspects of the data mining application i.e., how quickly must the data mining algorithm produce an answer? Does the model need to be continually updated? and so forth. Research on scalable data mining algorithms is likely to continue for some time, and we can expect more developments in this area. The reader should be cautioned to be aware that, as in everything else, there is no free lunch! In other words, there are typically trade-offs involving model accuracy, algorithm speed and memory, and so forth. Informed judgment on which type of algorithm and data structures best suit your problem will require careful consideration of both algorithmic issues and application details about how the algorithm and model will be used in practice.

Summary

We have discussed the memory hierarchy of modern computer and then present some index structures that database systems use to speed up the evaluation of queries. We then move to a discussion on relational databases and their query languages, as well as some special purpose database systems.

NOTES

We have discussed the problem of finding useful patterns and rules from large data sets. Recall that a pattern is a local concept, telling us something about a particular aspect of the data, while a model can be thought of as giving a full description of the data.

Reference

1. Hand D, Mannila H. Smith P: Principles of Data Mining (PHI).

2. Pujari A: Data Mining Techniques, University Press (orient Longman.

Glossary

Accuracy: The measure of a model's ability to correctly label a previously unseen test case. If the label is categorical (classification), accuracy is commonly reported as the rate which a case will be labeled with the right category. For example, a model may be said to predict whether a customer responds to a promotional campaign with 85.5% accuracy. If the label is continuous, accuracy is commonly reported as the average distance between the predicted label and the correct value. For example, a model may be said to predict the amount a customer will spend on a given month within $55. See also Accuracy Estimation, Classification, Estimation, Model, and Statistical Significance.

Accuracy Estimation: The use of a validation process to approximate the true value of a model's accuracy based on a data sample. See also Accuracy, RMS, Resampling Techniques and Validation.

NOTES

Affinity Modeling: The generation of a model that predicts which products or services sell together.

Algorithm: A well specified sequence of steps that accepts an input and produces an output. See also Data Mining Algorithm.

Analysis description file: A file used by InductionEngine to summarize the type of processes that will be performed in the data file. (i.e. target column, cost-matrix value). This file is generated after the first time of data analysis, and it can be used to speed up the process of defining the process information for analyzing other data file that has exactly the same data structure next time. See: InductionEngine, Train file.

Artificial Intelligence (AI): The science of algorithms that exhibit intelligent (rational) behaviour. See: Abductive logic, Deductive logic, Inductive logic, Expert Systems, Machine Learning, Heuristics.

Association: When one data item is found to be conditionally dependent on each other we say that they are associated. The term has similar implications to the term "correlation" but is not as precisely defined. See: Association Rule, Correlation.

Automated Discretization: Discretization which sets the number of bins based on the range of a numeric value. Therefore, the user is not required to specify the number of bins. However, certain values may be 'lost' from the decision tree because of automatic binning, which is not the case with intelligent binning. See Binning , Discretization.

Bayes Theorem : Describes a useful relationship between the likelihood of a future event (posteriors) and the likelihood of a prior event (priors). Given a hypothesis h and a dataset D the likelihood that the hypothesis is correct for the dataset P(h|D) can be expressed as P(D|h)P(h)/P(D). The use of P(h), "the prior", is the source of some debate among statisticians. The theorem can be proved by application of the product rule P(h^D)=P(h|D)P(D)=P(D|h)P(h). See: Naive-Bayes Classifier.

Binning: Choosing the number of bins into which a numeric range is split. For example, if salaries range from $20,000 to $100,000, the values must be binned into some number of groups, probably between eight and twenty. Many data mining products require the user to manually set binning. See Automated Binning, Discretization.

NOTES

Black-Box Method: Any technique that does not explain how it achieved its results. A black-box method may be unsuitable for many applications where external experts are required to assess the reasonableness of the model. See Neural Nets.

Bootstrap: A technique used to estimate a model's accuracy. Bootstrap performs b experiments with a training set that is randomly sampled from the data set. Finally, the technique reports the average and standard deviation of the accuracy achieved on each of the b runs. Bootstrap differs from cross-validation in that test sets across experiments will likely share some rows, while in cross-validation is guaranteed to test each row in the data set once and only once. See also accuracy, resampling techniques and cross-validation.

C4.5: A decision tree algorithm developed by Ross Quinlan, and a direct descendant of the ID3 algorithm. C4.5 can process both discrete and continuous data and makes classifications. C4.5 implements the information gain measure as its splitting criterion and employs post-pruning. Through the 1990s it was the most common algorithm to compare results against. See ID3, Pruning, Gini.

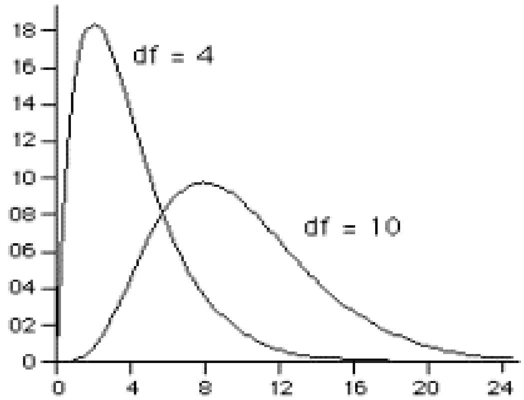
Campaign Response Modeling: This model predicts the people that will most likely respond to a promotional campaign.

CART: A decision tree algorithm developed in the early 80s by Breiman and other statisticians. CART stands for classification and regression trees. The algorithm uses a gini index for its splitting criterion. See: CHAID, Decision Tree, Gini.

Causal Factor: Any feature in the data which drives, influences, or causes another feature in the data. A simple example of cause & effect is that the sunrise causes the rooster to crow, not the other way around. See: Assoction, Correlation, Discriminating Factor.

CHAID: A decision tree splitting criterion based on the chi squared statistics formula, and the AID heuristics which enables it to also handle numerical data. See AID, Gini, Information Gain, statistics, heuristics.

Chi Square Distribution: A mathematical distribution with positive skew. The shape depends on the degree of freedom (df). The skew is less with more degree of freedom. The distribution is used directly or indirectly in many tests of significance. See also Chi Square Test.



Chi Square Test: The significance test used on contingency table to determine the relationship between two variables. Chi square test assumes that the data distribution follows the chi square distribution.

Classification/Classifier: The act of labeling a test case into one of a finite number of output classes. A model that classifies is sometimes referred to as a "classifier". Commonly a classifer's performance is measured by its ability to correctly label unseen test cases, that is its "accuracy". Inversely a classifier's performance may be measured by its "error rate". A more detailed insight into a classifier's performance is given by the Confusion Matrix structure because it captures how well the classifier predicts each of the available classes. If a Cost-Benefit Matrix is available then the classifier's performance is measured by the product of the Confusion and Cost-Benefit matrices. See also: Accuracy, Classification Algorithm, Confusion Matrix, Cost-Benefit Matrix, Estimation, Model, and Type I and Type II Errors.

Classification Algorithm: An algorithm that performs classification. Some algorithms first construct a model that then can be used to classify (e.g. Decision Tree, Logistic Regression), while other algorithms perform the labeling directly (e.g. k-Nearest-Neighbor). See also Decision Tree, k-Nearest-Neighbor, and Logistic Regression.

Cluster: A set of similar cases.

Clustering: The development of a model that labels a new record as a member of a group of similar records (a cluster). See clustering algorithms. For example, clustering could be used by a company to group customers according to income, age, prior purchase behavior. Cluster detection rarely provides actionable information, but rather

NOTES

feeds information to other data mining tasks. See also Clustering Algorithms, Segmentation and Profiling.

Clustering Algorithms: Given a data set these algorithms induce a model that classifies a new instance into a group of similar instances. Commonly the algorithms require that the number of (c) clusters to be identified is prespecified. E.g. find the c=10 best clusters. Given a distance metric, these algorithms will try to find groups of records that have low distances within the cluster but large distances with the records of other clusters. See also Agglomerative Clustering Algorithms, Clustering, Divisive Clustering Algorithms, K-means Algorithm, and Unsupervised Learning.

Confidence Window or Level: A statistical measurement of how sure one can be that a certain result is true. The window or level describes how close the value is likely to be to the exact result. See statistical significance.

Confounding Factor: (from the Latin confundere: to mix together) A distortion of an association between an intervention (I) and response (R) brought about by an extraneous cofactor (C). This problem occurs when the intervention is associated with C and C is an independent factor for the response. For example, (C) confounds the relationship between (R) and couponing (I), since R and C are related, and C is an independent risk factor for R.

When the differences between the treatment and control groups other than the treatment produce differences in response that are not distinguishable from the effect of the treatment, those differences between the groups are said to be confounded with the effect of the treatment (if any). For example, prominent statisticians questioned whether differences between individuals that led some to smoke and others not to (rather than the act of smoking itself) were responsible for the observed difference in the frequencies with which smokers and non-smokers contract various illnesses. If that were the case, those factors would be confounded with the effect of smoking. Confounding is quite likely to affect observational studies and experiments that are not randomized. Confounding tends to be decreased by randomization. See also Simpson's Paradox.

Confusion Matrix: A table that illustrates how well a classifier predicts. Instead of a simple misclassification error rate the table highlights where the model encounters difficulties. For each of the c output classes, the table presents an algorithm's likelihood of predicting each one of c classes. The sample confusion matrix below shows a classifier's accuracy on a problem with the three (c=3) output classes: cans, produce and dairy. The test set used to evaluate the algorithm contained 100 cases with a distribution of 30 cans, 35 produce and 35 dairy. A perfect classifier would have only made predictions along the diagonal, but the results below show that the algorithm was only correct on (20+25+24)/100 = 69% of the cases. The matrix also shows that the classifier often confuses dairy for cans (11 incorrect) and cans for dairy (9 wrong).

ACTUAL	PREDICTED			
	Cans	Produce	Dairy	SUM
Cans	20	2	11	34
Produce	2	25	1	28
Dairy	9	5	24	38
SUM	31	32	36	100

See: Cost-Benefit Matrix, Classification.

Contingency Tables: Used to examine the relationship between two continuous or categorical variables. Chi square test is used to test the significance between the column and the row frequencies, that is, whether the frequencies of one of the variables depends on the other.

Control Group Study (a.k.a. Randomized Controlled Study): Click here for more information. A model of evaluation in which the performance of cases who experience an intervention (the treatment group) is compared to the performance of cases (the control group) who did not experience the intervention in question. In medical studies where the intervention is the administration of drugs, for example, the control group is known as the placebo group because a neutral substance (placebo) is administered to the control group without the subjects (or researchers) knowing if it is an active drug or not. Typically, the intervention is considered successful if its performance exceeds that of the control group's by a statistically significant amount. When assignment to control and treatment groups is made at random, and no other factors enter into the assignment into control or treatment, any differences between the two groups are due either to the treatment or to random variation. When a given difference between the two groups is observed, say in spending on a particular set of items, it is possible to calculate the probability of this difference arising purely by chance. If the probability of an observed difference is very small (generally less than 5 percent but more stringent rules can be adopted) the observed difference is said to be due to the treatment. Click here for more information.

Correlation Coefficient (also Pearson's Product Moment Correlation Coefficient): A correlation coefficient is a number, usually between -1 and 1, that measures the degree to which two continuous columns are related. Usually the term really refers to the Peason's Product Moment Correlation Coefficient, usually denoted by r, which measures the linear association between two variables. If there is a perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is positive correlation, whenever one varialbe has a high (low) value, so does the other. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation of 0 means that there is no linear relationship between the variables. See also Spearman Rank Correlation Coefficient.

Cost-Benefit Matrix: (Click here for more information) A cost-benefit matrix is an input to the modeling process that allows predictive modelers to describe the costs and the benefits associated with each possible prediction. By default the cost-benefit matrix has a value of one (1.0) for correct predictions and zero (0.0) for incorrect predictions. This configuration asks that the predictive model optimize raw accuracy. In most real-world situations, however, an incorrect prediction has a net monetary cost (less than zero), and a correct prediction has a positive benefit. The correct or incorrect values that are chosen affect the values chosen for the matrix. The default cost matrix assumes no weighting for each output possibility. When the cost-benefit matrix has new non-default values assigned, the model optimizes the net benefit (profit) associated with each prediction. The cost-benefit matrix input is essential for businesses that want to optimize their return on investment. PredictionWorks supports the use of a cost-benefit matrix. Click here for more information.

Cross Sell Modeling: The generation of a model that predicts which products a specific customer would likely buy, or that predicts which customers would likely buy a specific product. This task is similar to Affinity Modeling and Campaign Response Modeling except that the resulting model is customer centric and targets existing customers instead of new prospects.

Cross-validation: A resampling technique used to estimate a model's accuracy. Cross-validation first segments the data rows into n nearly equally sized folds ($F_1..F_n$). Once the segmentation is accomplished, n experiments are run, each using F_i as a test set and the other n-1 folds appended together to form the train set. Finally,

NOTES

the technique reports the average and standard deviation of the accuracy achieved on each of the n runs. Too small a value for n will not achieve a confident accuracy estimate while too large a value for n will increase the variance of the estimate and will require increased computation. Empirical investigation into this technique has suggested the value of n=10 (10 fold cross-validation) to achieve useful results. See accuracy, resampling techniques and bootstrap.

Customer Relationship Management (CRM): The business processes that strengthens the relationship between a seller and their customers. To ensure positive contacts a CRM requires the measurement of each customer's value to the enterprise, the storing of all relevant transactional (behavioral) data, and the ability to predict future customer behavior. The implementation of a CRM process requires a significant technological investment in computing hardware and software, personnel and customer contact (touch point) systems.

Worldwide revenues in the customer relationship management (CRM) services markets will increase at a compound annual growth rate of 29 percent from $34.4 billion in 1999 to $125.2 billion in 2004, according to International Data Corp. (IDC). META Group predicts a 50 percent annual growth rate for the global CRM market and projects it will grow from more than $13 billion in 2000 to $67 billion in 2004.

CRM (customer relationship management) is an information industry term for methodologies, software, and usually Internet capabilities that help an enterprise manage customer relationships in an organized way. For example, an enterprise might build a database about its customers that described relationships in sufficient detail so that management, salespeople, people providing service, and perhaps the customer directly could access information, match customer needs with product plans and offerings, remind customers of service requirements, know what other products a customer had purchased, and so forth. According to one industry view, CRM consists of:

1. Helping an enterprise to enable its marketing departments to identify and target their best customers, manage marketing campaigns with clear goals and objectives, and generate quality leads for the sales team.
2. Assisting the organization to improve telesales, account, and sales management by optimizing information shared by multiple employees, and streamlining existing processes (for example, taking orders using mobile devices)

NOTES

3. Allowing the formation of individualized relationships with customers, with the aim of improving customer satisfaction and maximizing profits; identifying the most profitable customers and providing them the highest level of service.
4. Providing employees with the information and processes necessary to know their customers, understand their needs, and effectively build relationships between the company, its customer base, and distribution partner

Customer Value Modeling: The generation of a model that forecasts a customer's future spending in general or within specific business areas.

Data Selection: See Data Mining.

Data Mining: The automatic detection of trends and associations hidden in data, often in very structured data. Data Mining is sometimes thought of as a single phase of a larger process that includes Data Selection, Data Cleansing, Data Transformation, **Data Mining**, and Evaluation. See Data Mining Algorithms, Machine Learning, Statistics.

Data Cleansing: Locate and resolve problems with dataset:

- repeated rows, non-unique keys
- gaps in time
- missing data
- columns dominated by one value
- columns with a large number of categorical values
- discretizing or numeralizing columns

See Data Mining.

Data Mining Algorithm: An algorithm that accepts structured data and returns a model of the relationships within a data set. The algorithm's performance is measure by its accuracy, training/testing time, training/testing resource requirements and the model's understandability. See also Accuracy, Algorithm, Classification, Estimation, Parametric Modeling, and Non-Parametric Modeling.

Data Mining Task: A general problem for which data mining is called in to generate a model for. In this glossary data mining tasks are described according to the following template:

NOTES

Decision Tree: A model made up of a root, branches and leaves. Each branch in the tree contains a simple IF X THEN branch A1 ELSE IF ... ELSE branch An. Each leaf of the tree contains the prediction to be made, e.g. True 84%. Decision trees are similar to organization charts, with statistical information presented at each node. See: Axis Parallel Representations, Decision Tree Algorithm.

Decision Tree Algorithm: An predictive modeling technique from the fields of Machine Learning and Statistics that builds a simple tree-like structure to model the underlying pattern. The basic approach of the algorithm is to use a splitting criterion to determine the most predictive factor and place it as the first decision point in the tree (the root). The algorithm continually perform this search for predictive factors to build the branches of the tree until there is no more splits are necessary because only records of . Tree prunning raises accuracy on noisy data and can be performed as the tree is being constructed (pre-prunning), or after the construction (post-prunning). The algorithm is commonly used for classification problems that require the model represented in a human-readable model. PredictionWorks has several implementations of the Decision Tree Algorithm. Two of them use different splitting criterion (gini and entropy), and C4.5 is an implementation of a well-known algorithm by J.R. Quinlan. See also: Classification Algorithm, Estimation Algorithm, C4.5, Entropy, and Gini.

Demand Modeling: The generation of a model that forecasts when an item will be ordered and how large the order will be.

Diapers and Beer: A popular anecdote used to illustrate the unexpected but useful patterns discovered by data mining. The anecdote (probably apocryphal) recounts that a large supermarket chain used data mining to discover that customers often bought diapers and beer at the same time. When the retailer displayed two items together, sales increased for both items.

Discovery: Finding unexpected but useful trends and associations hidden in data. See modeling, associations.

Discriminating Factor: A measure of how important a causal factor is, used by decision trees to build the tree. See decision trees, causal factor.

Entropy: In data mining, a measure of the relative difference between two or more data partitions based on information theory. See also Gini.

NOTES

Entropy Heuristic: Use of entropy to determine the information gain of a particular attribute (predictor) in Decision Tree Algorithm. The attribute with the greatest entropy reduction is chosen as the test attribute in a Decision Tree model, because splitting on the attribute produces the purest / most uniformity data distribution. The purity of the data distribution affects the ultimate accuracy or the resulted model. The attribute with the greatest entropy reduction is also the attribute with the highest information gain. See also Gini Heuristic.

Estimation/Regressor: The act of labeling a test case with an continuous value. A model or algorithm that estimates is sometimes referred to as a "regressor". Commonly a regressor's repfermance is measured by its ability to predict a value that is near tot he actual value, such as with a correlation coefficient. See also Classification, Correlation Coefficient, and Estimation Algorithm.

Estimation Algorithm: An algorithm that performs estimation. Some algorithms first conduct a model that then can be used to estimate (e.g. Decision Tree, Linear Regression), while other algorithms perform the labeling directly (e.g. K-Nearest-Neighbor). See also: Decision Tree, Estimation, K-Nearest-Neighbor, Linear Regression.

Euclidean Distance: Measure of the distance between two points. For any two n-dimensional points $a = (a_1, a_2, ... a_n)$ and $b = (b_1, b_2, ... b_n)$, the distance between a and b is equal to:

$$\text{sqrt}((a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_n - b_n)^2)$$

Forecasting: Adapting data mining techniques to forecast future trends with statistical reliability. Forecasting is often confused with prediction, but is usually much more complex. See time series analysis/forecasting, what-if analysis, neural nets.

Fraud Detection: This modeling procedure predicts infrequent events that bear a large financial penalty. This type of modeling is commonly used to detect criminal activity such as credit card fraud, insurance claim fraud, and Internet/wireless hacking. Each type of fraud detection requires a slightly different technique. Generally, anomalous events that do not fit the normal usage patterns trigger fraud detection alarms. The main challenges to these tasks are due to the low frequency of the undesirable events, usually under one percent (1%). Usage of the cost-benefit matrix is critical to properly weigh the benefits of correct and incorrect predictions. These conditions often mean that 1) The lack of examples of fraudulent events makes it difficult to discriminate between legitimate and fraudulent behavior. 2) The overwhelming number of legitimate events leaves little room for lift on the already high accuracy from the simple model that simply predicts all events to be legitimate. 3) The existence of a cost-benefit matrix allows us to dismiss the simple model described in condition 2. The use of cost-benefit matrix information into predictive modeling, however, is a new concept.

Gini: A modern decision tree index algorithm developed by Ron Bryman. Gini handles both numbers and text, and offers good processing speed. See also C4.5, and CHAID.

Gini Heuristic: Use of Gini to determine the information gain of a particular attribute (predictor) in Decision Tree Algorithm. See also Entropy Heuristic.

Heuristics: A rule of thumb that speeds up the locating of an optimal result.

Horizon Effect: The event where a Decision Tree construction in halted prematurely because no further benefit seemed apparent. Usually happens as the result with Pre Pruning.

ID3: The first algorithm which was designed to build decision trees. ID3 was invented by Ross Quinlan at the University of Sidney Australia. ID3 was followed by ID4, ID6 and see 5. See C4.5, Gini, CHAID, CART.

Information Gain: A measurement used to select the test attribute (predictor) at each node during Decision Tree model construction. See also Attribute Selection Measure or Splitting Criterion.

Instance-based Learning: Machine learning technique in which training datasets are stored in entirety and a distance function is used to make predictions. See also: KNN

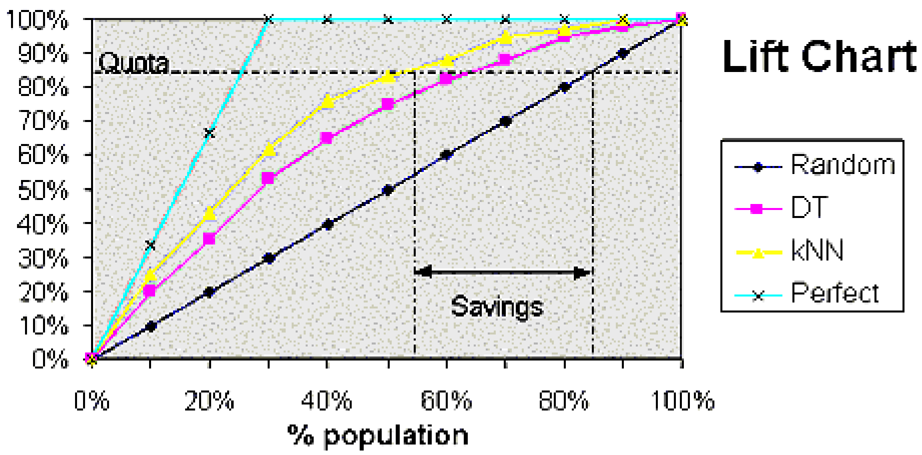
k-Nearest-Neighbor (kNN) Algorithm: An algorithm from the field of Pattern Recognition that generates both estimation and classification models. The algorithm assumes that similar cases behave similarly. The most common proximity measure is based on the Euclidean distance between two vectors. For classification problems the prediction is based on the statistical mode (most common) of the response value for the k closest cases. For example to predict the target value of a test case and k is set to the value seven (k=7) then the seven cases most similar to the test case would be fetched and the most common value from among the seven would be used to

NOTES

make the prediction. If the problem was an estimation challenge then the average from the seven would be used for the prediction. This algorithm works most all datatypes but is most effective in the presence of continuous columns where the Euclidean distance can be calculated.

Least Squares: A method used to find the line that goes through the datapoints with the shortest squares of distances between the datapoints and this line.

Gain Chart (Lift Curve): A method from direct marketing that helps to visualize a classifier's accuracy on binary (positive/negative) problems. Lift charts are commonly used in promotion campaign response modeling (responded/did not respond) to present how well a model performs when compared to a random mailing. The x-axis represents the percentage of the total population covered, say a city of 100,000. The y-axis presents the cumulative percentage of correctly classified positive cases, say 30,000 would respond if they received the mailout. The chart should include the performance of a random case selection (a straight line from [0%,0%] to [100%,100%]) and the performance of the model under investigation. Other possible lines in the chart include the performance of other competing models, the performance of a perfect classifier, and the quota to be achieved. From the chart below we notice that kNN (k-Nearest Neighbor) reaches the 85% quota faster than the DT (Decision Tree).



Linear Regression: An algorithm that generates estimation models from the field of Statistics. The algorithm assumes that a linear relationship exists between the independent variables and the response variable. PredictionWorks uses Least Squares as a measure of model fitness. See also Parametric Modeling, and Least Squares.

Log-Likelihood: The logarithm of a likelihood equation. It is used when the logarithm is easier to work with than the equation itself and the outcome is unaffected..

Logistic Regression: An algorithm that generates classification models from the field of Statistics. The target column may be NON-binary, but most implementations are limited to two value (binary) predictions. PredictionWorks' logistic regression algorithm uses maximum likelihood estimation to determine parameters of the regression equation. Forward stepwise selection is used to find the most predictive columns from which to build the model. The stopping criterion is based on a chi-square distribution. See also: Classification Algorithm, and Log-Likelihood.

Machine Learning: Is the research area within AI that studies algorithms which induce models from a set of data. Machine learning differs from statistics and pattern recognition in its focus on knowledge representation, symbolic data, automated discovery, and computational complexity analysis.

Majority Model Algorithm: A classification algorithm that simply predicts the most common value found in the target column. For example, if 87% of a data set's rows contain the value "Did_Not_Respond" in the target column, then the Majority Model will simply predict "Did_Not_Respond" for every row, and on average achieve a raw accuracy of 87%. If one single value (class) is very prevalent is the data set, say greater than 95%, then this model's raw accuracy will be difficult to imporve on. When this is the case, the situation usually requires a Cost-Benefit Matrix to represent the fact that predicting the majority class may have little value in the real world. For example, predicting that all customers will not respond to a direct mail campaign may be accurate but that model will generate no revenue. See also: Classification Algorithm, Minority Model Algorithm, Mean Model Algorithm, and Naïve Model Algorithm.

Market Basket Analysis: A technique, used in large retail chains, which studies every purchase made by customers to find out which sales are most commonly made together. See Diapers and Beer.

Maximum Likelihood Estimation: Method of choosing parameters for a regression equation that maximizes the likelihood of observing the target value. **Mean Model Algorithm:** An estimation algorithm that simply predicts the mean (average) value found in the target column. For example, if the average value of a data set's target column is 7.875, then the Mean Model will simply predict 7.875 for every row. Often the central tendency of a data set is better captured by the median value, however calculating a column's median is significantly more complex (requires sorting) than calculating the mean value. See also: Naïve Model Algoirthm. **Meta data file:** This file contains the necessary information about the data structure (ie delimiter, data type for each column). This file is generated after the first time of data analysis, and it can be used to speed up the process of defining the data structure for analyzing other data file that has exactly the same data structure next time. **Minority Model Algorithm:** A classification algorithm that simply predicts the least common value found in the target column. PredictionWorks implicitly supports this model through the Naive Model Algorithm by creating a separate model for every value in the target class. This model is of interest when a Cost-Benefit Matrix biases the value of predicting against the majority value (class). See also: Naive Model Algorithm. **Missing Value:** A data value that is not known or does not exist. Two common reasons for missing values are unfilled optional features, data entry malfunctions, and non-applicability of column. Some algorithms require that missing-values be filled-in (imputed). PredictionWorks' imputes values based on the mean if the column is numeric or the mode if the column is categorical.

Mode: In mathematics, the most common class in a set

Model: A structure that labels an event based on the event's characteristics. A model is measured by its accuracy and speed, and sometimes by its ability to clearly justify its prediction. See Accuracy, Modeling.

Model Validation: The testing of a model with unseen data. See: Model.

Modeling: Building a model from a set of training data. See also Model, Non-Parametric Modeling, and Parametric Modeling.

Naïve Bayes Classifier: An predictive modeling algorithm that uses the conditional probabilities estimates calculated from the training data to estimate the posterior probability of seeing an event. In a binary True/False classification problem, for example, each record **x** is tested to see if it is more likely that P(True)P(True|**x**) or P(False)P(False|**x**). The key assumption that makes the algorithm fast is that all predictive attributes are independent of each other.

This assumption allows the conditional probability P(True|**x**) to be rewritten as a simple multiplication of the conditinal probabilities for each individual attribute-value **x**iP, P(True|**x**) = (True|**x**1)´...´P(True|**x**m). Although this assumption is typically invalid for general datasets the algorithm generally continues to perform well. The algorithm is generally faster but less accurate than other predictive modeling techniques. It has however been found to be very accurate for text classification. See: Bayes Theorem.

$$v_{NaiveBayes} = \arg\max_{v_j \in \{Class\}} P(v_j) \prod_i P(a_i|v_j)$$

Naïve Model Algorithm: An algorithm that generates both classification and estimation predictions. The approach of the algorithm is simply to make predictions based on the distribution of the target column. There are several possible versions of this algorithm that make assumptions. PredictionWorks' implements the Majority Model, and Single Class Model. See also: Majority Model Algorithm, Minority Model Algorithm, Mean Model Algorithm, Single Class Model Algorithm, classification/estimation models, domain X, and target column.

Neural Networks: A predictive modeling technique that uses artificial neural networks (ANN) to model an underlying pattern. Neural networks are particularly good at modeling mathematical functions. See black box.

Non-Parametric Modeling: The development of a model without the need to assume that the data abide by a specific (parametric) distribution. Most data mining predictive modeling techniques are non-parametric in order to make them general enough to apply to most datasets. See: Parametric Modeling.

Overfitting: The act of mistaking noise in training data for true trends in the population. An overfitted model will make incorrect predictions in those regions it overfit. A general technique to compensate for a predictive modeling algorithm that overfits data is to use pruning. See: Pruning.

Parametric Modeling: The development of a model with the requirement that the data abide by some specific (parametric) distribution. The use of these algorithms generally requires a through understanding of their specific assumptions and how to test for their validity. If the assumption is correct, the generated model will be more accurate than a model generated by a non-parametric

NOTES

modeling algorithm. Statiscal modeling techniques, such as linear regression, are generally parametric. See Linear Regression, Non-Parametric Modeling.

Partitioning: Choosing data which is most interesting for mining. This is typically at least eighty percent of the work of data mining. See sampling.

Post-Pruning: A type of Pruning where the pruning process is performed after the model is constructed. The approach uses data that has been set aside to test for sections in the model that are inaccurate. Also see Pre-Pruning, Pruning and Overfitting.

Predictive Modeling: Modeling that emphasizes accuracy on unseen data.

Prediction: Using existing data to predict how other factors will behave, assuming that some facts are known about the new factor. Making a credit check of new customers by using data on existing customers is a good example of prediction. See What-If? analysis, time series analysis/forecasting, forecasting.

Pre-Pruning: A type of Pruning where each update to the model is first tested for its likelihood of being accurate. This approach is faster than Post Pruning, but will be less accurate due to the horizon effect. See also Pruning and Overfitting.

Price Elasticity Modeling: The generation of a model that forecasts a product's sales volume based on its price. For example, if the price of milk is increased from $3.50/g to $4/g the sales of this product from 683/week to 546/week.

Probability Distribution: A term from statistics for the likelihood associated with each of the values in a random variable. For example the pdf of a random variable for the tos of unbiased coin is {Tails=0.5, Heads=0.5}. See: Expected Value, Probability Density Function (pdf), Random Variable.

Probability Density Function (pdf): See: Probability Distribution.

Pruning: The technique of removing parts of a predictive model that do not capture true features of the underlying pattern. There are two types of pruning process, Pre-Pruning and Post Pruning. Decision Tree algorithms typically employ prunning to avoid overfitting. See also Overfitting.

Posterior Probability (a posteriori probability) Models: Classification model which estimates the probability that input x belongs to class C, denoted by P(C|x). With these estimates a set of inputs can be ranked from most to least certain so that resources can be focused on the top prospects. See also Campaign Response Modeling, and Gain Chart.

Random Error: Can be thought of as either sampling variation or as the sum total of other unobservable chance deviations that occur with sampled data. The key to working with random error is that it tends to balance out in predictable ways over the long run. It can thus be dealt with using routine methods of estimation and hypothesis testing.

Random Model Algorithm: An algorithm that generates classification predictions. The basic approach is to randomly predict one of the classes without consideration for its frequency.

Random Variable: A term from statistics for a symbol, e.g. X , that represents an experiment with a probabilistic outcome, e.g. the flipping of a coin. The values of a random variable represent an outcome of an experiment, e.g. {Heads,Heads}. In data mining each attribute in a dataset can be though of as a random variable. See: Expected Value, Probability Distribution.

Resampling Techniques: An empirical approach to model accuracy estimation based on the training and testing on multiple samples of the data set. See also Accuracy Estimation, Cross-Validation and Bootstrap.

Retention Modeling: The generation of a model that forecasts the probability that a customer will significantly reduce the business they bring in, or defect all together.

Root-mean-squared Error (RMS error): A measure of an estimation model's error. Given a data set, it is defined as the square root of the square of the difference between the true values and the model's predicted values. For a data set with n records

$$\frac{\sum_{i=1}^{n}\sqrt{(actual_i - predicted_i)^2}}{n}$$

Rule Induction: A method of performing discovery by inducing rules about data. Rule induction tests given values in the data set to see

NOTES

which other data are its strongest associated factors. See decision trees, discovery, causal factor.

Sampling: Taking a random sample of data in order to reduce the number of records to mine. Sampling is statistically complicated, but can be done in an RDBMS by use of a simple random number generator and column in the database. See partitioning.

Scoring: The process of applying a model to a database or list.

Segment: A well-defined set of cases.

Segmentation and Profiling: The generation of a model that groups like-minded customers into 'prototypical' customer types.

Simpson's Paradox: What is true for the parts is not necessarily true for the whole.

Single Class Model Algorithm: An algorithm that generates classification predictions. The approach of the algorithm is to choose a single class and predict that class every time. This algorithm is simpler than Majority Model because it does not need to know the frequency of the target column. Can only be better than Majority Model when a Cost Matrix is used to give priority a minority class. See also: Majority Model Algorithm, Target Column, and Cost-Benefit Matrix.

Statistical significance: A measure the statistical likeliness that a given numerical value is true. See confidence window or level.

Statistics: The field in mathematics which studies the quantification of variance. One of the basic building blocks of data mining. See Heuristics, Machine Learning.

Systematic Error: In contrast to random error, is less easily managed. According to modern epidemiologic theory, systematic error (or bias, as they say), can result from: information bias (due to errors in measurement), selection bias (due to flaws in the sampling), and confounding (due to the damaging / biasing effects of extraneous factors

Target Column: (a.k.a. dependent variable/attribute, label)the column whose values a predictive model has to accurately predict. If the target column is categorical the modeling challenge is referred to as classification, otherwise if the target column is numeric the

NOTES

challenge is referred to as an estimation problem. See also test file, train file, classifier, and estimation.
Test file: The dataset file on which predictions will be made. The best predictive model discovered during the training stage will be used to predict the values of the target column for each of the rows in the file. The file should be the same format as the train file although the values in the target column may be represented by a missing value such as a question mark. See also target column.

Time Series Analysis/Forecasting: A complicated technology which is used to give statistically accurate forecasting. This is often confused with prediction or simple forecasting, but time series analysis/forecasting is much more difficult, and mathematically based. See forecasting.

Total Wallet Modeling: The generation of a model that forecasts a customer's total spending for a particular product or service.

Train file: The dataset file that will be tested against with many algorithms to discover the best predictive model. One of file's the columns is selected as the target (dependent) column. The PredictionWorks' web-service accepts comma, tab or single-space delimited files. If the discovered model is will be used to predict future behavior then the predictor (independent) columns in the train file must contain information that was available before the value of the target column was known, and possibly further back. For example, if the target value contains whether a banner was clicked on or not then the data in the predictor columns must have data that was available before the person saw the banner. In the example of a mailed-out coupon, the predictor columns must have data that was available not just before the coupon was redeemed but data that was available when the coupon was mailed out.

Threshold: The minimum percentage of occurrence of a class needed to choose that class. e.g. If you have a dataset consisting of blue socks and red socks and your threshold is 0.6, you will need at least 60% of one colour of sock to choose that colour. See also: classification

Type I and Type II Errors: Decisions based on two-valued (binary) predictive models may be in error for two reasons:

- Type I Error (False Positive) errors occur when the difference with the null-hypothesis is significant, due to factors other than chance, when in fact it is not. The probability of this type of error is the same as the significance level of the test. Many domains consider this the most serious type of error to make. It is equivalent to a judge finding an innocent suspect guilty.

- Type II Error (False Negative) errors occur when the difference with the null-hypothesis is due to chance, when in fact it is not. Some domains, such as direct marketing represent this type of error as lost income because the person who would have responded positively was never contacted. See also Cost-Benefit Matrix.

Value Drivers Modeling: The generation of a model that predicts the top reasons for a customer to continue to do business with the company.

Variance: A term from statistics for the measure of the dispersion of a random variable's probability distribution around its expected value. Typically written as Var(Y). Variance is calculated as $E\left((Y - \mu_Y)^2\right)$.

See: Expected Value, Probability Distribution, Random Variable.

Visualization: Visual representation of discovered patterns.