10 *Predictive Modeling for Classification*

10.1 A Brief Overview of Predictive Modeling

Descriptive models, as described in chapter 9, 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.

In chapter 6 we discussed many of the basic functional forms of models that can be used for prediction. In this chapter and the next, we examine such models in more detail, and look at some of the specific aspects of the criteria and algorithms that permit such models to be fitted to the data.

Predictive modeling can be thought of as learning a mapping from an input set of vector measurements $\mathbf{x}$ 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 $\mathbf{x}(i)$ with a corresponding "target" value $y(i), 1 \leq i \leq n$. 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 $\mathbf{x}$ and a set of estimated parameters θ for the model f. Recall that f is the functional form of the model structure (chapter 6), the θs are the unknown parameters within f whose values we will determine by minimizing a suitable score function on the data (chapter 7), and the process of searching for the best θ values is the basis for the actual data mining algorithm (chapter 8). 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, as discussed in chapter 6, 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 $y(\hat{i}) = f(\mathbf{x}(i);\theta)$ and the true value $y(i)$—that is,

$$\begin{aligned} S(\theta) &= \sum_{D_{train}} d\Big(y(i), \hat{y}(i)\Big) \\ &= \sum_{D_{train}} d\Big(y(i), f(\mathbf{x}(i);\theta)\Big) \end{aligned} \qquad (10.1)$$

where the sum is taken over the tuples $(\mathbf{x}(i), y(i))$ 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 (see chapter 7 for further discussion in this context). 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 (see the discussion in chapter 8).

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 $S(\theta)$ 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 (as discussed in chapter 7).

We noted in chapter 6 that there are two important distinct kinds of tasks in predictive modeling depending on whether Y is categorical or real-valued. For categorical Y the task is called *classification* (or *supervised classification* to distinguish it from problems concerned with defining the classes in the first instance, such as cluster analysis), and for real-valued y the task is called *re-*

gression. Classification problems are the focus of this chapter, and regression problems are the focus of the next chapter. Although we can legitimately discuss both forms of modeling in the same general context (they share many of the same mathematical and statistical underpinnings), in the interests of organizational style we have assigned classification and regression each their own chapter. However, it is important for the reader to be aware that many of the model structures for classification that we discuss in this chapter have a "twin" in terms of being applicable to regression (chapter 11). For example, we discuss tree structures in the classification chapter, but they can also be used for regression. Similarly we discuss neural networks under regression, but they can also be used for classification.

In these two chapters we cover many of the more commonly used approaches to classification and regression problems—that is, the more commonly used tuples of model structures, score functions, and optimization techniques. The natural taxonomy of these algorithms tends to be closely aligned with the model structures being used for prediction (for example, tree structures, linear models, polynomials, and so on), leading to a division of the chapters largely into subsections according to different model structures. Even though specific combinations of models, score functions, and optimization strategies have become very popular ("standard" data mining algorithms) it is important to remember the general reductionist philosophy of data mining algorithms that we described in chapter 5; for a particular data mining problem we should always be aware of the option of tailoring the model, the score function, or the optimization strategy for the specific application at hand rather than just using an "off-the-shelf" technique.

10.2 Introduction to Classification Modeling

We introduced predictive models for classification in chapter 6. Here we briefly review some of the basic concepts. In classification we wish to learn a mapping from a vector of measurements $\mathbf{x}$ 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 $\{c_1, \ldots, c_m\}$ to denote this class variable for the rest of this chapter (instead of using Y). The observed or measured variables $X_1, \ldots, X_p$ 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 $\mathbf{x}$ 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 $1 \leq i \leq n$, $1 \leq j \leq p$. In our introductory discussion we will implicitly assume that we are using the so-called "0-1" loss function (see chapter 7), 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 $\mathbf{x}$ and produces as output a symbol from the set $\{c_1, \ldots, c_m\}$. 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 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 $\mathbf{X}$ space. That is, it is possible for members of more than one class to occur at some (perhaps all) values of $\mathbf{X}$—though the probability that members of each class occur at any given value $\mathbf{x}$ will be different. (It is the fact that these probabilities differ that permits us to make a classification. Broadly speaking, we assign a point $\mathbf{x}$ to the most probable class at $\mathbf{x}$.) 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* (Fisher, 1936), was based on precisely this idea: it sought that linear combination of the variables in $\mathbf{x}$ 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 $\mathbf{x}(i)$) 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 $\mathbf{x}$. 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 $\mathbf{x}$ 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 $\mathbf{x}$ for each class might be multivariate Normal, and the parameters θ_k would represent the mean (location) and variance (scale) characteristics for each class. If 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*

$$p(c_k|\mathbf{x}) = \frac{p(\mathbf{x}|c_k, \theta)p(c_k)}{\sum_{l=1}^{m} p(\mathbf{x}|c_l, \theta_l)p(c_l)}, \quad 1 \leq k \leq m. \tag{10.2}$$

The posterior probabilities $p(c_k|\mathbf{x}, \theta_k)$ implicitly carve up the input space $\mathbf{x}$ 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 $\mathbf{x}$. For example, for the case in which all errors incur equal cost we should predict the class value c_k that has the highest posterior probability $p(c_k|\mathbf{x})$ (is most likely given the data) for any given $\mathbf{x}$ value. Note that this scheme is optimal in the sense that no other prediction method can do better (with the given variables $\mathbf{x}$)—it does not mean that it makes no errors. Indeed, in most real problems the optimal classification scheme will have a nonzero error rate, arising from the overlap of the distributions $p(\mathbf{x}|c_k, \theta_k)$. This overlap means that the maximum class probability $p(c_k|\mathbf{x}) < 1$, so that there is a non-zero probability $1-p(c_k|\mathbf{x})$ of data arising from the other (less likely) classes at $\mathbf{x}$, even though the optimal decision at $\mathbf{x}$ is to choose c_k. Extending this argument over the whole space, and averaging with respect to $\mathbf{x}$ (or summing over discrete-valued variables), the *Bayes Error Rate* is defined as

$$p_B^* = \int (1 - \max_k p(c_k|\mathbf{x}))p(\mathbf{x})d\mathbf{x}. \tag{10.3}$$

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.

Example 10.1

Figure 10.1 shows a simple artificial example with a single predictor variable X (the horizontal axis) and two classes. The upper two plots show how the

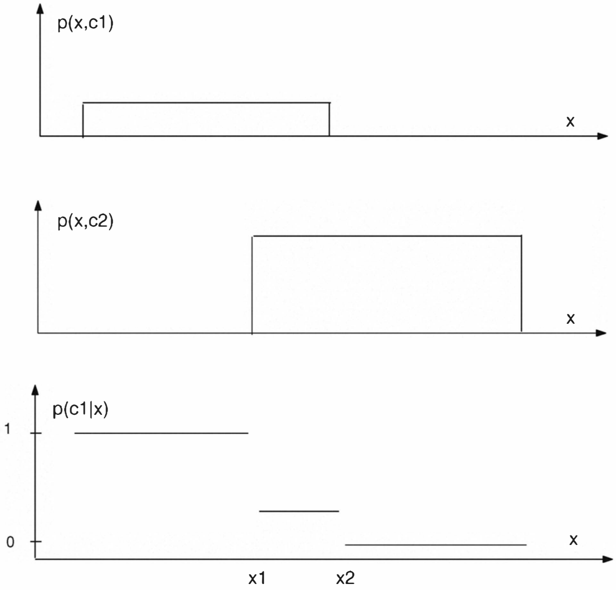

Figure 10.1 A simple example illustrating posterior class probabilities for a two-class one-dimensional classification problem.

data are distributed within class 1 and class 2 respectively. The plots show the joint probability of the class and the variable X, $p(x, c_k), k = 1, 2$. Each has a uniform distribution over a different range of X; class c_1 tends to have lower x values than class c_2. There is a region along the x axis (between values x_1 and x_2) where both class populations overlap.

The bottom plot shows the posterior class probability for class c_1, $p(c_1|x)$ as calculated via Bayes rule given the class distributions shown in the upper two plots. For values of $x \leq x_1$, the probability is 1 (since only class c_1 can produce data in that region), and for values of $x \geq x_2$ the probability is 0 (since only class c_2 can produce data in that region). The region of overlap (between x_1 and x_2) has a posterior probability of about 1/3 for class c_1 (by Bayes rule) since class c_2 is roughly twice as likely as class c_1 in this region. Thus, class c_2 is the Bayes-optimal decision for any $x \geq x_1$ (noting that in the regions where $p(x, c_1)$ or $p(x, c_2)$ are *both* zero, the posterior probability is undefined). However, note that between x_1 and x_2 there is some fundamental ambiguity about which class may be present given an x value in this region; that is, although c_2

is the more likely class there is a $1/3$ chance of c_1 occurring. In fact, since there is a $1/3$ chance of making an incorrect decision in this region, and let us guess from visual inspection that there is a 20% chance of an x value falling in this region, this leads to a rough estimate of a Bayes error rate of about $\frac{20}{3} \approx 6.67\%$ for this particular problem.

Now consider a situation in which $\mathbf{x}$ 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. This prompts this question: why should we not simply use many measurements in a classification problem, until the error rate is sufficiently low? The answer lies in the the bias-variance principle discussed in chapters 4 and 7. 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 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 $\mathbf{x}$ 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 perceptrons (see section 10.3) and the more general support vector machines (see section 10.9).

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, discussed in section 10.7. Note that decision trees (for example, CART from chapter 5) 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).

3. **The class-conditional approach:** Here, the class-conditional distributions $p(\mathbf{x}|c_k, \theta_k)$ are modeled explicitly, and along with estimates of $p(c_k)$ are inverted via Bayes rule (equation 10.2) to arrive at $p(c_k|\mathbf{x})$ 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 $p(\mathbf{x}|c_k, \theta_k)$) 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 discussed in chapter 4. In practice the parameter estimates used in equation 10.2, $\hat{\theta}_k$, are often estimated via maximum likelihood for each class c_k, and "plugged in" to $p(\mathbf{x}|c_k, \theta_k)$. There are Bayesian alternatives that average over θ_k. Furthermore, the functional form of $p(\mathbf{x}|c_k, \theta_k)$ can be quite general—any of parametric (for example, Normal), semi-parametric (for example, finite mixtures), or non-parametric

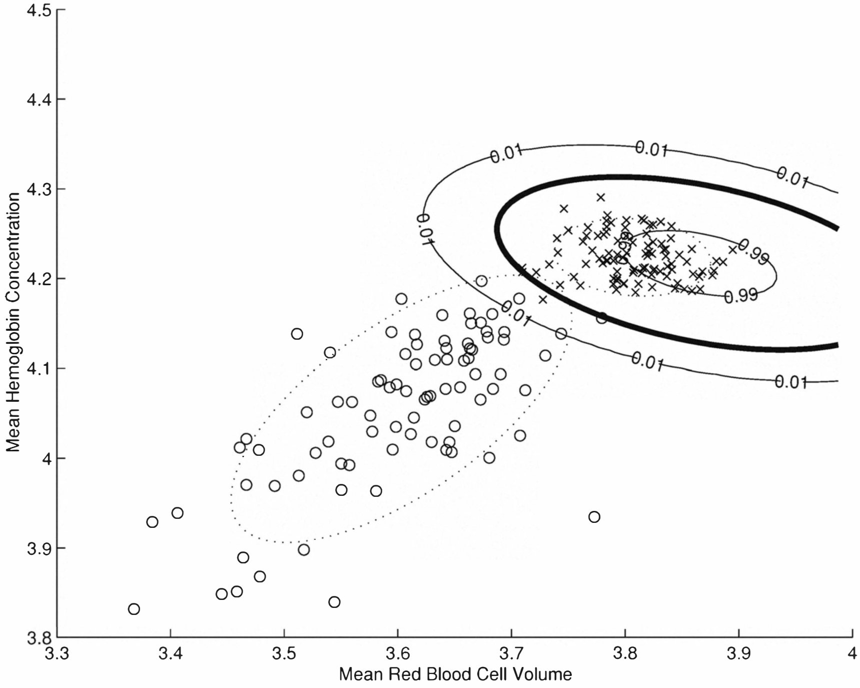

Figure 10.2 Posterior probability contours for $p(c_1|\mathbf{x})$ where c_1 is the label for the healthy class for the red blood cell data discussed in chapter 9. The heavy line is the decision boundary ($p(c_1|\mathbf{x}) = p(c_2|\mathbf{x}) = 0.5$) and the other two contour lines correspond to $p(c_1|\mathbf{x}) = 0.01$ and $p(c_1|\mathbf{x}) = 0.99$. Also plotted for reference are the original data points and the fitted covariance ellipses for each class (plotted as dotted lines).

(for example, kernels) can be used to estimate $p(\mathbf{x}|c_k, \theta_k)$. 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).

Example 10.2 Choosing the most likely class is in general equivalent to picking the value of k for which the discriminant function $g_k(\mathbf{x}) = p(c_k|\mathbf{x})$ is largest, $1 \leq m$. It is often convenient to redefine the discriminants as $g_k(\mathbf{x}) =$

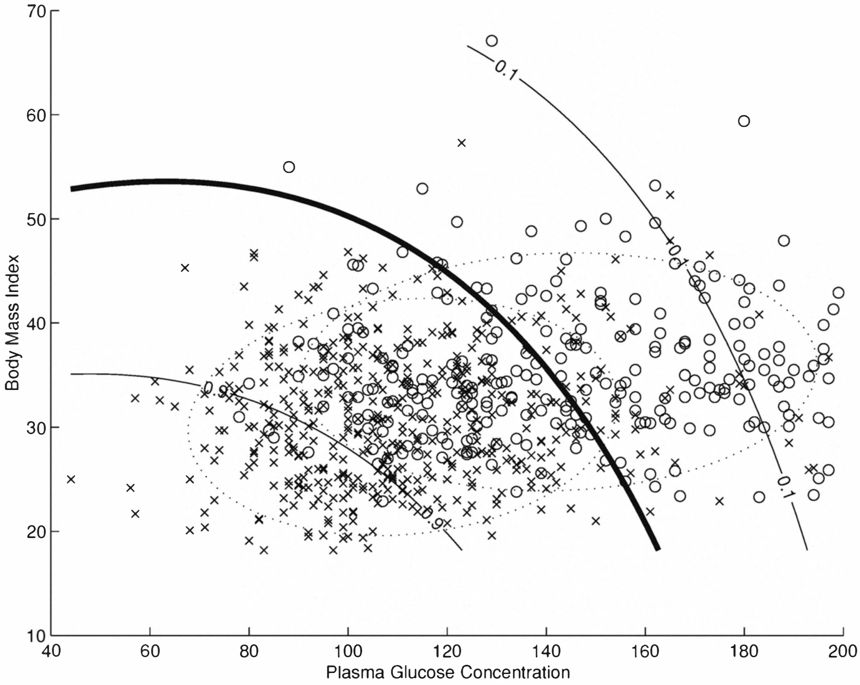

Figure 10.3 Posterior probability contours for $p(c_1|\mathbf{x})$ where c_1 is the label for the diabetic class for the Pima Indians data of chapter 3. The heavy line is the decision boundary ($p(c_1|\mathbf{x}) = p(c_2|\mathbf{x}) = 0.5$) and the other two contour lines correspond to $p(c_1|\mathbf{x}) = 0.1$ and $p(c_1|\mathbf{x}) = 0.9$. The fitted covariance ellipses for each class are plotted as dotted lines.

$\log p(\mathbf{x}|c_k)p(c_k)$ (via Bayes rule). For multivariate real-valued data $\mathbf{x}$, a commonly used class-conditional model is the multivariate Normal as discussed in chapter 9. If we take log (base e) of the Normal multivariate density function, and ignore terms that do not include k we get discriminant functions of the following form:

$$g_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k) - \frac{p}{2}\log|\Sigma_k| - \log p(c_k) \quad 1 \le k \le m. \quad (10.4)$$

In the general case each of these $g_k(\mathbf{x})$ involve quadratics and pairwise products of the individual x variables. The decision boundary between any two classes k and l is defined by the solution to the equation $g_k(\mathbf{x}) - g_l(\mathbf{x}) = 0$

as a function of $\mathbf{x}$, and this will also be quadratic in $\mathbf{x}$ in the general case. Thus, a multivariate Normal class-conditional model leads to quadratic decision boundaries in general. In fact, if the covariance matrices Σ_k for each class k are constrained to be the same ($\Sigma_k = \Sigma$) it is straightforward to show that the $g_k(\mathbf{x})$ functions reduce to linear functions of $\mathbf{x}$ and the resulting decision boundaries are linear (that is, they define hyperplanes in the p-dimensional space).

Figure 10.2 shows the results of fitting a multivariate Normal classification model to the red blood cell data described in chapter 9. Maximum likelihood estimates (chapter 4) of $\mu_k, \Sigma_k, p(c_k)$ are obtained using the data from each of the two classes, $k = 1, 2$, and then "plugged in" to Bayes rule to determine the posterior probability function $p(c_k|\mathbf{x})$. In agreement with theory, we see that the resulting decision boundary is indeed quadratic in form (as indeed are the other plotted posterior probability contours). Note that the contours fall off rather sharply as one goes outwards from the mean of the healthy class (the crosses). Since the healthy class (class c_1) is characterized by lower variance in general than the anemic class (class c_2, the circles), the optimal classifier (assuming the Normal model) results in a boundary that completely encircles the healthy class.

Figure 10.3 shows the results of the same classification procedure (multivariate Normal, maximum likelihood estimates) but applied to a different data set. In this case two particular variables from the two-class Pima Indians data set (originally discussed in chapter 3) were used as the class variables, where problematic measurements taking value 0 (thought to be outliers, see discussion in chapter 3) were removed a priori. In contrast to the red blood cell data of figure 10.2, the two classes (healthy and diabetic) are heavily overlapped in these two dimensions. The estimated covariance matrices Σ_1 and Σ_2 are unconstrained, leading again to quadratic decision boundary and posterior probability contours. The degree of overlap is reflected in the posterior probability contours which are now much more spread out (they fall off slowly) than they were previously in figure 10.2.

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 $\mathbf{x}$), whereas the class-conditional/generative approach focuses on the distributions of $\mathbf{x}$ 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 $\mathbf{x}$ 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 $\mathbf{x}$ to one of m classes; however, each does so within a probabilistic framework, while the "true" discriminative classifier is not constrained to do so.

We will discuss examples of each of these approaches in the sections that follow. Which type of classifier works best in practice will depend on the nature of the problem. For some applications (such as in medical diagnosis) it may be quite useful for the classifier to generate posterior class probabilities rather than just class labels. Methods based on the class-conditional distributions also have the advantage of providing a full description for each class (which, for example, provides a natural way to detect outliers—inputs $\mathbf{x}$ that do not appear to belong to any of the known classes). However, as discussed in chapter 9, it may be quite difficult (if not impossible) to accurately estimate functions $p(\mathbf{x}|c_k, \theta_k)$ in high dimensions. In such situations the discriminative classifier may work better. In general, methods based on the class-conditional distributions will require fitting the most parameters (and thus will lead to the most complex modeling), the regression approach will require fewer, and the discriminative model fewest of all. Intuitively this makes sense, since the optimal discriminative model contains only a subset of the information of the optimal regression model (the boundaries, rather than the full class probability surfaces), and the optimal regression model contains less information than the optimal class-conditional distribution model.

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 chapter 11 on regression models we will discuss more general and recent neural network models.

In its simplest form, the perceptron model (for two classes) is just a linear combination of the measurements in $\mathbf{x}$. Thus, define $h(\mathbf{x}) = \sum w_j x_j$, 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 $h(\mathbf{x})$ with a threshold, which we shall here take to be zero for simplicity. If all class 1 points have $h(\mathbf{x}) > 0$ and all class 2 points have $h(\mathbf{x}) < 0$, 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 $h(\mathbf{x}) > 0$ 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 $h(\mathbf{x}) < 0$, the weights are updated, so that $h(\mathbf{x})$ 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 find a separating surface, provided a sufficiently small value of λ is chosen. The updating algorithm is reminiscent of the gradient descent techniques discussed in chapter 8, 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:

$$S(\mathbf{w}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{p+1} w_j x_j(i) - y(i) \right)^2 . \tag{10.5}$$

Since this is a quadratic error function it has a single global minimum as a function of the weight vector $\mathbf{w}$ and is relatively straightforward to minimize (either by a local gradient descent rule as in chapter 8, or more directly in closed-form using linear algebra).

Numerous variations of the basic perceptron idea exist, 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 $\mathbf{X}$) 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 considered 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 example of a discriminative approach, since it does not explicitly estimate either the posterior probabilities of class membership or the class-conditional distributions. Fisher (1936) presents one of the earliest treatments of linear discriminant analysis (for the two-class case). Let $\hat{\mathbf{C}}$ be the pooled sample covariance matrix defined as

$$\hat{\mathbf{C}} = \frac{1}{n_1 + n_2}\left(n_1\hat{\mathbf{C}}_1 + n_2\hat{\mathbf{C}}_2\right), \tag{10.6}$$

where n_i is the number of training data points per class, and $\hat{\mathbf{C}}_i$ are the $p \times p$ sample (estimated) covariance matrices for each class, $1 \leq i \leq 2$ (as defined in chapter 2). To capture the notion of separability along any p-dimensional vector $\mathbf{w}$, Fisher defined a scalar score function as follows:

$$S(\mathbf{w}) = \frac{\mathbf{w}^T\hat{\mu}_1 - \mathbf{w}^T\hat{\mu}_2}{\mathbf{w}^T\hat{\mathbf{C}}\mathbf{w}}, \tag{10.7}$$

where $\hat{\mu}_1$ and $\hat{\mu}_2$ are the $p \times 1$ mean vectors for $\mathbf{x}$ 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 $\mathbf{w}$ 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 $S(\mathbf{w})$, the problem is to determine the direction $\mathbf{w}$ that maximizes this expression. In fact, there is a closed form solution for the maximizing $\mathbf{w}$, given by:

$$\hat{\mathbf{w}}_{lda} = \hat{\mathbf{C}}^{-1}(\hat{\mu}_1 - \hat{\mu}_2). \tag{10.8}$$

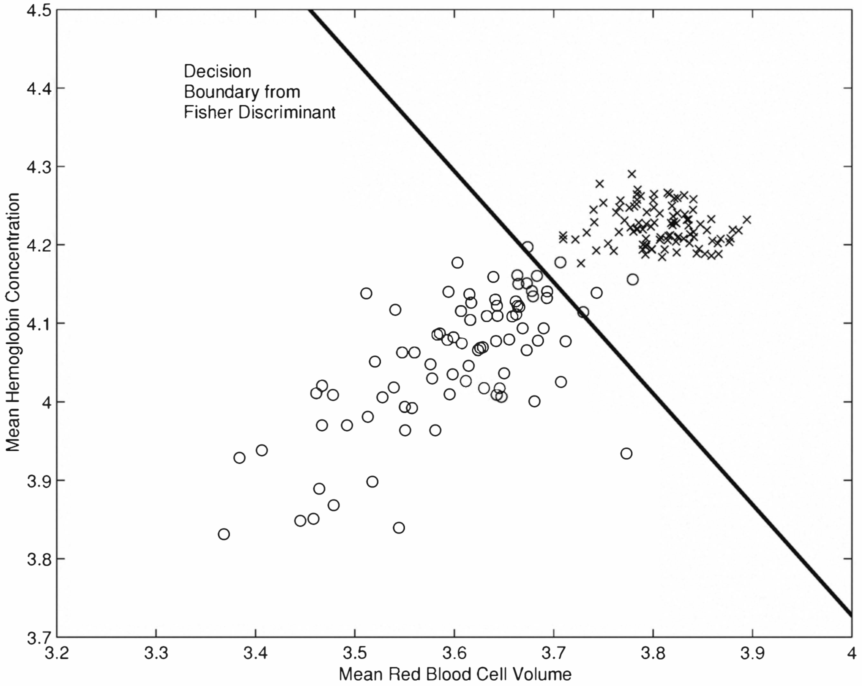

Figure 10.4 Decision boundary produced by the Fisher linear discriminant applied to the red blood cell data from chapter 9, where the crosses are the healthy class and the circles correspond to iron deficient anemia.

A new point is classified by projecting it onto the maximally separating direction, and classifying $\mathbf{x}$ to class 1 if

$$\mathbf{w}_{lda}^T\left(\mathbf{x} - \frac{1}{2}(\hat{\mu}_1 - \hat{\mu}_2)\right) > \log\frac{p(c_1)}{p(c_2)}, \tag{10.9}$$

where $p(c_1)$ and $p(c_2)$ are the respective class probabilities.

Figure 10.4 shows the application of the Fisher linear discriminant method to the two class anemia classification problem discussed earlier. The linear decision boundary is not quite as good at separating the training data as the quadratic boundaries of example 10.2.

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 10.2 (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, as discussed in example 10.2. *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 $\hat{\mathbf{C}}_i$, $1 \leq i \leq m$. 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 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.

Example 10.3 The entropy criterion for a particular real-valued threshold test T (where T stands for a threshold test $X_j > T$ on one of the variables) is defined as the average entropy after the test is performed:

$$H(C|T) = p(T=0)H(C|T=0) + p(T=1)H(C|T=1) \qquad (10.10)$$

where the conditional entropy $H(C|T=1)$ is defined as

$$-\sum_{c_k} p(c_k|T=1)\log_2 p(c_k|T=1).$$

> The average entropy is then the uncertainty from each branch ($T = 1$ or $T = 0$) averaged over the probability of going down each branch. Since we are trying to split the data into subsets where as many of the data points belong to one class or the other, this is directly equivalent to minimizing the entropy in each branch. In practice, we search among all variables (and all tests or thresholds on each variable) for the single test T that results in minimum average entropy after the binary split.

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 overfitting. 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 (this is analogous to avoiding overfitting in neural networks by terminating the convergence procedure, as we will discuss in the next chapter). 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 discussed in chapter 11—which is why more 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 $\mathbf{x}$ 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 (for example, the CART algorithm described in chapter 5) are used to trade off goodness of fit to the training data against model complexity.

Two other strategies for avoiding the problem of overfitting 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.

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 staircaselike approximation to this diagonal decision surface. Figure 10.5 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.

10.6 Nearest Neighbor Methods

At their basic level, nearest neighbor methods are very straightforward: to classify a new object, with input vector $\mathbf{y}$, we simply examine the k closest training data set points to $\mathbf{y}$ 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

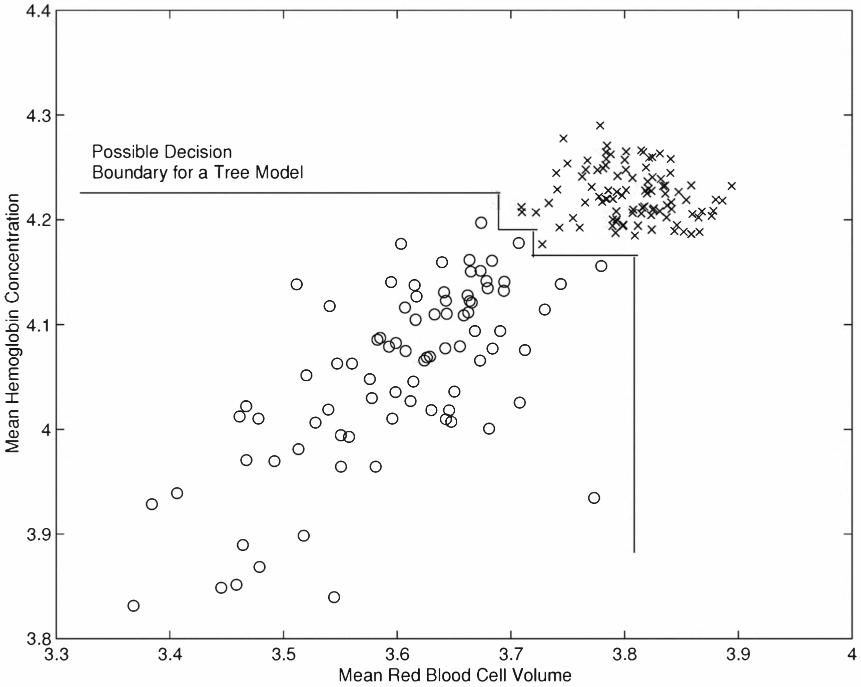

Figure 10.5 Decision boundary for a decision tree for the red blood cell data from chapter 9, composed of "axis-parallel" linear segments (contrast with the simpler boundaries in figure 10.4).

among these most similar objects.

In theoretical terms, we are taking a small volume of the space of variables, centered at $\mathbf{x}$, and with radius the distance to the kth 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.

Many applications of nearest neighbor methods adopt a Euclidean metric: if $\mathbf{y}$ is the input vector for the point to be classified, and $\mathbf{x}$ is the input vector for a training set point, then the Euclidean distance between them is $\sum_j (x_j - y_j)^2$. As discussed in chapter 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 units—as, 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 $\mathbf{y}$ have the same probability of belonging to class c_1 as does a point at $\mathbf{y}$, so no bias is introduced by including them in the k nearest neighbors. This is true no matter how far from $\mathbf{y}$ they are, provided they are on the contour. In contrast, points close to $\mathbf{y}$ but not on the contour of $P(c_1|\mathbf{x})$ through $\mathbf{y}$ 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 that we discussed in chapter 6. 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 $\mathbf{x}$, or choice of bandwidth that varies according to $\mathbf{x}$) 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 ∞) 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 overfits the data, given the large opportunity for overfitting 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 $\mathbf{x}$ 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 to their membership in clusters. When finding the nearest point for input point $\mathbf{y}$, the clusters nearest to $\mathbf{y}$ 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 $\mathbf{x}$, the estimated probability that it belongs to class c_1 is

$$p\left(c_1|\mathbf{x}\right) = \frac{1}{1 + \exp\left(\beta'\mathbf{x}\right)}. \tag{10.11}$$

Since the probabilities of belonging to the two classes sum to one, by subtraction, the probability of belonging to class 2 is

$$p\left(c_2|\mathbf{x}\right) = \frac{\exp\left(\beta'\mathbf{x}\right)}{1 + \exp\left(\beta'\mathbf{x}\right)}. \tag{10.12}$$

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,

$$\log\frac{p\left(c_2|\mathbf{x}\right)}{p\left(c_1|\mathbf{x}\right)} = \beta'\mathbf{x}. \tag{10.13}$$

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 $\mathbf{x}$ variables if the distributions can be modeled by log-linear models (mentioned in chapter 9) 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. This is described in chapter 11, in the more general context of *generalized linear models*.

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 $\mathbf{X}$), 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 pointed out in chapters 6 and 9 that 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

$$p(\mathbf{x}|c_k) = p(x_1, \ldots, x_p|c_k) = \prod_{j=1}^{p} p(x_j|c_k), \quad 1 \leq k \leq m \tag{10.14}$$

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 a 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 probability that a point with measurement vector $\mathbf{x}$ will belong to the kth class is

$$\begin{aligned} p(c_k|\mathbf{x}) &\propto p(\mathbf{x}|c_k)p(c_k) \\ &= p(c_k)\prod_{j=1}^{p} p(x_j|c_k) \quad 1 \leq k \leq m \end{aligned} \tag{10.15}$$

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

$$\log\frac{p\left(c_1|\mathbf{x}\right)}{p\left(c_2|\mathbf{x}\right)} = \log\frac{p\left(c_1\right)}{p\left(c_2\right)} + \sum\log\frac{p\left(x_j|c_1\right)}{p\left(x_j|c_2\right)}. \tag{10.16}$$

Thus the log odds that a case belongs to class c_1 is 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, $\log\frac{p(x_j|c_1)}{p(x_j|c_2)}$, 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 10.15 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 has led to its widespread popularity, particularly in the machine learning literature.

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" pairwise 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 (or belief network—see chapter 6) 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 (as discussed in chapter 9).

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 (as discussed in chapter 5 under the back-propagation algorithm and again to be discussed in chapter 11 for regression) 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 (we will return to them in the context of regression in chapter 11). 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 nonlinear 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 vector 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 location 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 overfitting (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 of chapter 4, 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 membership 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. (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 (see chapter 7 and elsewhere), 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 entire 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. The area has been the subject of several review papers—see the further reading section for details.

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 coefficient* 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 $\mathbf{x}$ variables, and others to continuous $\mathbf{x}$, 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. The Further Reading section gives references to material discussing such issues.

In general, the assessment of classification models is an important area, and one that has been the subject of a huge amount of study.

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).

We have discussed this problem in a general modeling context in chapter 6, 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. Examples of this approach include principal components analysis (in which we try to find the directions in the input space that have the highest variance, essentially a data compression technique—see chapters 3 and 6), projection pursuit (in which an algorithm searches for interesting linear projections—see chapters 6 and 11), and related techniques such as factor analysis and independent components analysis. While these techniques can be quite powerful in their own right, they suffer the disadvantage of not necessarily being well matched to the overall goal of improving classification performance. A case in point is principal component analysis. Figure 10.6 shows an illustrative example in which the first principal component direction (the direction in which the data would be projected and potentially used as input to a classifier) is completely orthogonal to the best linear discriminant for the problem—that is, it is completely in the wrong direction for the classification task! This is not a problem with the principal component methodology per se but simply an illustration of matching an inappropriate technique to the classification task. This is of course a somewhat artificial and pathological example; in practice principal component projections can often be quite useful for classification, but nonetheless it is important to keep the objectives in mind.

10.12 Further Reading

Fisher's original paper on linear discriminant analysis dates from 1936. Duda, Hart, and Stork (2001) (the second edition of the classic pattern recognition

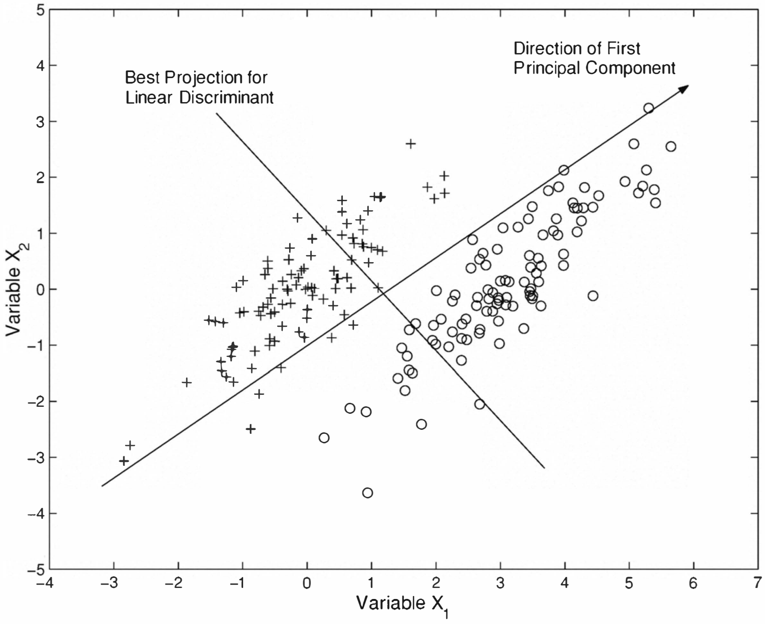

Figure 10.6 An illustration of the potential pitfalls of using principal component analysis as a preprocessor for classification. This is an artificial two-dimensional classification problem, with data from each class plotted with different symbols. The first principal component direction (which would be the first candidate direction on which to project the data if this were actually a high-dimensional problem) is in fact almost completely orthogonal to the best linear projection for discrimination as determined by Fisher's linear discriminant technique.

text by Duda and Hart (1973)) contains a wealth of detail on a variety of classification methods, with a particularly detailed treatment of Normal multivariate classifiers (chapter 3) and linear discriminant and perceptron learning algorithms (chapter 5). Statistically oriented reviews of classification are given by Hand (1981, 1997), Devijver and Kittler (1982), Fukunaga (1990), McLachlan (1992), Ripley (1996), Devroye, Gyorfi, and Lugosi (1996), and Webb (1999). Bishop (1995) provides a neural network perspective, Mitchell (1997) offers a viewpoint from artificial intelligence, and Witten and Frank (2000) provide a data-mining oriented introduction to classification.

Dasarathy (1991) contains many of the classic papers on nearest neighbor classification from the statistical pattern recognition literature and general descriptions of nearest neighbor methods, including outlines of methods for reducing the size of the retained set, may be found in Hand (1981) and McLachlan (1992). Choice of metric for nearest neighbor methods is discussed in Short and Fukunaga (1981), Fukunaga and Flick (1984), and Myles and Hand (1990). Hastie and Tibshirani (1996) describe an adaptive local technique for estimating a metric. Asymptotic properties of nearest neighbor rules are described in Devroye and Wagner (1982). The related kernel method is discussed in Hand (1982). The problem of the meaning of "nearest" neighbor in high dimensions is considered in Beyer et al. (1999) and Bennett, Fayad, and Geiger (1999), who also discuss the use of clustering for approximate searches.

One of the earliest descriptions of tree-based models is in Morgan and Sonquist (1963). The application of decision trees to classification was popularized in machine learning by Quinlan (1986, 1993). In statistics, the book by Breiman et al. (1984) describing the CART (*Classification And Regression Trees*) algorithm was highly influential in the widespread adoption and application of tree models. Chapter 7 of Ripley (1996) contains an extensive overview of the different contributions to the tree-learning literature from statistics, computer science, and engineering. A recent survey article is Murthy (1998). Scalable algorithms for constructing decision trees are considered in Shafer, Agrawal, and Mehta (1996), Gehrke, Ramakrishnan, and Ganti (1998), and Rastogi and Shim (1998). The Sprint method of Shafer, Agrawal, and Mehta (1996) operates in a very small amount of main memory but applies only to the CART splitting criterion. The RainForest framework of Gehrke, Ramakrishnan, and Ganti (1998) can be used to scale up a variety of splitting criteria, but its memory usage depends on the sizes of domains of the variables. The method of Rastogi and Shim (1998) interleaves tree building and pruning, thus preventing unnecessary access to the data. A nice survey of scalability issues is Ganti, Gehrke, and Ramakrishnan (1999).

Discussions of the independence Bayes method include Russek, Kronmal, and Fisher (1983), Hilden (1984), Kohavi (1996), Domingos and Pazzani (1997), and Hand and Yu (1999).

Descriptions of support vector machines are given by Vapnik (1995), Burges (1998), and Vapnik (1998). Scholkopf, Burges, and Smola (1999) is a collection of recent papers on the same topic, and Platt (1999) describes a useful technique for speeding up the training of these classifiers.

Techniques for combining classifiers, such as model averaging, are described in Xu, Krzyzak, and Suen (1992), Wolpert (1992), Buntine (1992), Ho, Hull, and Srihari (1994), Schaffer (1994), and Oliver and Hand (1996). Freund and Schapire (1996) describe the boosting technique; more recent theoretical treatments are provided by Schapire et al. (1998) and Friedman, Hastie, and Tibshirani (2000).

A detailed review of assessment and evaluation methods for classification algorithms is given in Hand (1997). Reviews of error rate estimation methods in particular are given in Toussaint (1974), Hand (1986), McLachlan (1987), and Schiavo and Hand (1999). The reject option is treated in detail in Devijver and Kittler (1982). MacMillan and Creelman (1991) provide an overview of ROC and related methods.

A seminal discussion of missing data, their different types, and how to handle them, is given in Little and Rubin (1987).