

4

Classification

The linear regression model discussed in Chapter 3 assumes that the response variable Y is quantitative. But in many situations, the response variable is instead *qualitative*. For example, eye color is qualitative, taking on values blue, brown, or green. Often qualitative variables are referred to as *categorical*; we will use these terms interchangeably. In this chapter, we study approaches for predicting qualitative responses, a process that is known as *classification*. Predicting a qualitative response for an observation can be referred to as *classifying* that observation, since it involves assigning the observation to a category, or class. On the other hand, often the methods used for classification first predict the probability of each of the categories of a qualitative variable, as the basis for making the classification. In this sense they also behave like regression methods.

There are many possible classification techniques, or *classifiers*, that one might use to predict a qualitative response. We touched on some of these in Sections 2.1.5 and 2.2.3. In this chapter we discuss three of the most widely-used classifiers: *logistic regression*, *linear discriminant analysis*, and *K-nearest neighbors*. We discuss more computer-intensive methods in later chapters, such as generalized additive models (Chapter 7), trees, random forests, and boosting (Chapter 8), and support vector machines (Chapter 9).

qualitative

classification

classifier

logistic
regression

linear
discriminant
analysis

K-nearest
neighbors

4.1 An Overview of Classification

Classification problems occur often, perhaps even more so than regression problems. Some examples include:

1. A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?
2. An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.
3. On the basis of DNA sequence data for a number of patients with and without a given disease, a biologist would like to figure out which DNA mutations are deleterious (disease-causing) and which are not.

Just as in the regression setting, in the classification setting we have a set of training observations $(x_1, y_1), \dots, (x_n, y_n)$ that we can use to build a classifier. We want our classifier to perform well not only on the training data, but also on test observations that were not used to train the classifier.

In this chapter, we will illustrate the concept of classification using the simulated `Default` data set. We are interested in predicting whether an individual will default on his or her credit card payment, on the basis of annual income and monthly credit card balance. The data set is displayed in Figure 4.1. We have plotted annual `income` and monthly credit card `balance` for a subset of 10,000 individuals. The left-hand panel of Figure 4.1 displays individuals who defaulted in a given month in orange, and those who did not in blue. (The overall default rate is about 3%, so we have plotted only a fraction of the individuals who did not default.) It appears that individuals who defaulted tended to have higher credit card balances than those who did not. In the right-hand panel of Figure 4.1, two pairs of boxplots are shown. The first shows the distribution of `balance` split by the binary `default` variable; the second is a similar plot for `income`. In this chapter, we learn how to build a model to predict `default` (Y) for any given value of `balance` (X_1) and `income` (X_2). Since Y is not quantitative, the simple linear regression model of Chapter 3 is not appropriate.

It is worth noting that Figure 4.1 displays a very pronounced relationship between the predictor `balance` and the response `default`. In most real applications, the relationship between the predictor and the response will not be nearly so strong. However, for the sake of illustrating the classification procedures discussed in this chapter, we use an example in which the relationship between the predictor and the response is somewhat exaggerated.

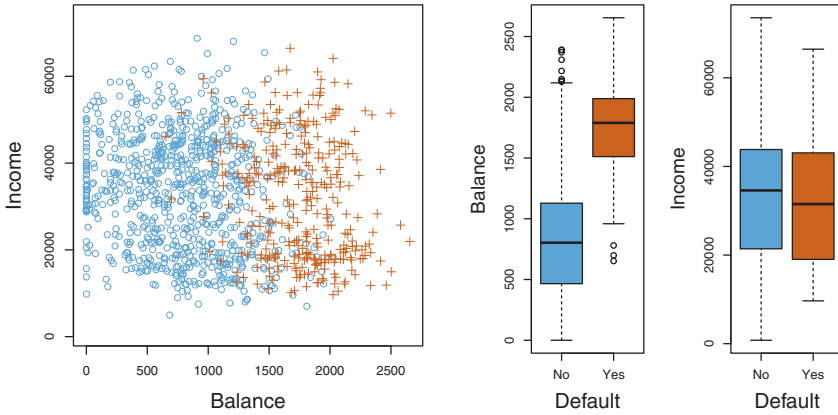


FIGURE 4.1. The **Default** data set. Left: The annual incomes and monthly credit card balances of a number of individuals. The individuals who defaulted on their credit card payments are shown in orange, and those who did not are shown in blue. Center: Boxplots of **balance** as a function of **default** status. Right: Boxplots of **income** as a function of **default** status.

4.2 Why Not Linear Regression?

We have stated that linear regression is not appropriate in the case of a qualitative response. Why not?

Suppose that we are trying to predict the medical condition of a patient in the emergency room on the basis of her symptoms. In this simplified example, there are three possible diagnoses: **stroke**, **drug overdose**, and **epileptic seizure**. We could consider encoding these values as a quantitative response variable, Y , as follows:

$$Y = \begin{cases} 1 & \text{if } \text{stroke}; \\ 2 & \text{if } \text{drug overdose}; \\ 3 & \text{if } \text{epileptic seizure}. \end{cases}$$

Using this coding, least squares could be used to fit a linear regression model to predict Y on the basis of a set of predictors X_1, \dots, X_p . Unfortunately, this coding implies an ordering on the outcomes, putting **drug overdose** in between **stroke** and **epileptic seizure**, and insisting that the difference between **stroke** and **drug overdose** is the same as the difference between **drug overdose** and **epileptic seizure**. In practice there is no particular reason that this needs to be the case. For instance, one could choose an equally reasonable coding,

$$Y = \begin{cases} 1 & \text{if } \text{epileptic seizure}; \\ 2 & \text{if } \text{stroke}; \\ 3 & \text{if } \text{drug overdose}. \end{cases}$$

which would imply a totally different relationship among the three conditions. Each of these codings would produce fundamentally different linear models that would ultimately lead to different sets of predictions on test observations.

If the response variable's values did take on a natural ordering, such as *mild*, *moderate*, and *severe*, and we felt the gap between mild and moderate was similar to the gap between moderate and severe, then a 1, 2, 3 coding would be reasonable. Unfortunately, in general there is no natural way to convert a qualitative response variable with more than two levels into a quantitative response that is ready for linear regression.

For a *binary* (two level) qualitative response, the situation is better. For instance, perhaps there are only two possibilities for the patient's medical condition: **stroke** and **drug overdose**. We could then potentially use the *dummy variable* approach from Section 3.3.1 to code the response as follows:

$$Y = \begin{cases} 0 & \text{if } \text{stroke}; \\ 1 & \text{if } \text{drug overdose}. \end{cases}$$

We could then fit a linear regression to this binary response, and predict **drug overdose** if $\hat{Y} > 0.5$ and **stroke** otherwise. In the binary case it is not hard to show that even if we flip the above coding, linear regression will produce the same final predictions.

For a binary response with a 0/1 coding as above, regression by least squares does make sense; it can be shown that the $X\hat{\beta}$ obtained using linear regression is in fact an estimate of $\Pr(\text{drug overdose}|X)$ in this special case. However, if we use linear regression, some of our estimates might be outside the $[0, 1]$ interval (see Figure 4.2), making them hard to interpret as probabilities! Nevertheless, the predictions provide an ordering and can be interpreted as crude probability estimates. Curiously, it turns out that the classifications that we get if we use linear regression to predict a binary response will be the same as for the linear discriminant analysis (LDA) procedure we discuss in Section 4.4.

However, the dummy variable approach cannot be easily extended to accommodate qualitative responses with more than two levels. For these reasons, it is preferable to use a classification method that is truly suited for qualitative response values, such as the ones presented next.

4.3 Logistic Regression

Consider again the **Default** data set, where the response **default** falls into one of two categories, **Yes** or **No**. Rather than modeling this response Y directly, logistic regression models the *probability* that Y belongs to a particular category.

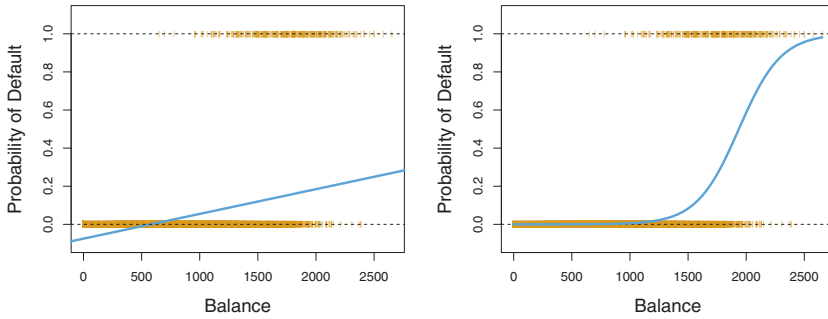


FIGURE 4.2. Classification using the `Default` data. Left: Estimated probability of `default` using linear regression. Some estimated probabilities are negative! The orange ticks indicate the 0/1 values coded for `default` (No or Yes). Right: Predicted probabilities of `default` using logistic regression. All probabilities lie between 0 and 1.

For the `Default` data, logistic regression models the probability of default. For example, the probability of default given `balance` can be written as

$$\Pr(\text{default} = \text{Yes} | \text{balance}).$$

The values of $\Pr(\text{default} = \text{Yes} | \text{balance})$, which we abbreviate $p(\text{balance})$, will range between 0 and 1. Then for any given value of `balance`, a prediction can be made for `default`. For example, one might predict `default = Yes` for any individual for whom $p(\text{balance}) > 0.5$. Alternatively, if a company wishes to be conservative in predicting individuals who are at risk for default, then they may choose to use a lower threshold, such as $p(\text{balance}) > 0.1$.

4.3.1 The Logistic Model

How should we model the relationship between $p(X) = \Pr(Y = 1 | X)$ and X ? (For convenience we are using the generic 0/1 coding for the response). In Section 4.2 we talked of using a linear regression model to represent these probabilities:

$$p(X) = \beta_0 + \beta_1 X. \quad (4.1)$$

If we use this approach to predict `default=Yes` using `balance`, then we obtain the model shown in the left-hand panel of Figure 4.2. Here we see the problem with this approach: for balances close to zero we predict a negative probability of default; if we were to predict for very large balances, we would get values bigger than 1. These predictions are not sensible, since of course the true probability of default, regardless of credit card balance, must fall between 0 and 1. This problem is not unique to the credit default data. Any time a straight line is fit to a binary response that is coded as

0 or 1, in principle we can always predict $p(X) < 0$ for some values of X and $p(X) > 1$ for others (unless the range of X is limited).

To avoid this problem, we must model $p(X)$ using a function that gives outputs between 0 and 1 for all values of X . Many functions meet this description. In logistic regression, we use the *logistic function*,

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}. \quad (4.2)$$

logistic
function

To fit the model (4.2), we use a method called *maximum likelihood*, which we discuss in the next section. The right-hand panel of Figure 4.2 illustrates the fit of the logistic regression model to the **Default** data. Notice that for low balances we now predict the probability of default as close to, but never below, zero. Likewise, for high balances we predict a default probability close to, but never above, one. The logistic function will always produce an *S-shaped* curve of this form, and so regardless of the value of X , we will obtain a sensible prediction. We also see that the logistic model is better able to capture the range of probabilities than is the linear regression model in the left-hand plot. The average fitted probability in both cases is 0.0333 (averaged over the training data), which is the same as the overall proportion of defaulters in the data set.

maximum
likelihood

After a bit of manipulation of (4.2), we find that

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X}. \quad (4.3)$$

The quantity $p(X)/[1 - p(X)]$ is called the *odds*, and can take on any value between 0 and ∞ . Values of the odds close to 0 and ∞ indicate very low and very high probabilities of default, respectively. For example, on average 1 in 5 people with an odds of 1/4 will default, since $p(X) = 0.2$ implies an odds of $\frac{0.2}{1-0.2} = 1/4$. Likewise on average nine out of every ten people with an odds of 9 will default, since $p(X) = 0.9$ implies an odds of $\frac{0.9}{1-0.9} = 9$. Odds are traditionally used instead of probabilities in horse-racing, since they relate more naturally to the correct betting strategy.

odds

By taking the logarithm of both sides of (4.3), we arrive at

$$\log \left(\frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X. \quad (4.4)$$

The left-hand side is called the *log-odds* or *logit*. We see that the logistic regression model (4.2) has a logit that is linear in X .

log-odds
logit

Recall from Chapter 3 that in a linear regression model, β_1 gives the average change in Y associated with a one-unit increase in X . In contrast, in a logistic regression model, increasing X by one unit changes the log odds by β_1 (4.4), or equivalently it multiplies the odds by e^{β_1} (4.3). However, because the relationship between $p(X)$ and X in (4.2) is not a straight line,

β_1 does *not* correspond to the change in $p(X)$ associated with a one-unit increase in X . The amount that $p(X)$ changes due to a one-unit change in X will depend on the current value of X . But regardless of the value of X , if β_1 is positive then increasing X will be associated with increasing $p(X)$, and if β_1 is negative then increasing X will be associated with decreasing $p(X)$. The fact that there is not a straight-line relationship between $p(X)$ and X , and the fact that the rate of change in $p(X)$ per unit change in X depends on the current value of X , can also be seen by inspection of the right-hand panel of Figure 4.2.

4.3.2 Estimating the Regression Coefficients

The coefficients β_0 and β_1 in (4.2) are unknown, and must be estimated based on the available training data. In Chapter 3, we used the least squares approach to estimate the unknown linear regression coefficients. Although we could use (non-linear) least squares to fit the model (4.4), the more general method of *maximum likelihood* is preferred, since it has better statistical properties. The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ of default for each individual, using (4.2), corresponds as closely as possible to the individual's observed default status. In other words, we try to find $\hat{\beta}_0$ and $\hat{\beta}_1$ such that plugging these estimates into the model for $p(X)$, given in (4.2), yields a number close to one for all individuals who defaulted, and a number close to zero for all individuals who did not. This intuition can be formalized using a mathematical equation called a *likelihood function*:

$$\ell(\beta_0, \beta_1) = \prod_{i: y_i=1} p(x_i) \prod_{i': y_{i'}=0} (1 - p(x_{i'})). \quad (4.5) \quad \text{likelihood function}$$

The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to *maximize* this likelihood function.

Maximum likelihood is a very general approach that is used to fit many of the non-linear models that we examine throughout this book. In the linear regression setting, the least squares approach is in fact a special case of maximum likelihood. The mathematical details of maximum likelihood are beyond the scope of this book. However, in general, logistic regression and other models can be easily fit using a statistical software package such as **R**, and so we do not need to concern ourselves with the details of the maximum likelihood fitting procedure.

Table 4.1 shows the coefficient estimates and related information that result from fitting a logistic regression model on the **Default** data in order to predict the probability of **default=Yes** using **balance**. We see that $\hat{\beta}_1 = 0.0055$; this indicates that an increase in **balance** is associated with an increase in the probability of **default**. To be precise, a one-unit increase in **balance** is associated with an increase in the log odds of **default** by 0.0055 units.

	Coefficient	Std. error	Z-statistic	P-value
Intercept	-10.6513	0.3612	-29.5	<0.0001
balance	0.0055	0.0002	24.9	<0.0001

TABLE 4.1. For the **Default** data, estimated coefficients of the logistic regression model that predicts the probability of **default** using **balance**. A one-unit increase in **balance** is associated with an increase in the log odds of **default** by 0.0055 units.

Many aspects of the logistic regression output shown in Table 4.1 are similar to the linear regression output of Chapter 3. For example, we can measure the accuracy of the coefficient estimates by computing their standard errors. The z -statistic in Table 4.1 plays the same role as the t -statistic in the linear regression output, for example in Table 3.1 on page 68. For instance, the z -statistic associated with β_1 is equal to $\hat{\beta}_1/SE(\hat{\beta}_1)$, and so a large (absolute) value of the z -statistic indicates evidence against the null hypothesis $H_0 : \beta_1 = 0$. This null hypothesis implies that $p(X) = \frac{e^{\beta_0}}{1+e^{\beta_0}}$ —in other words, that the probability of **default** does not depend on **balance**. Since the p -value associated with **balance** in Table 4.1 is tiny, we can reject H_0 . In other words, we conclude that there is indeed an association between **balance** and probability of **default**. The estimated intercept in Table 4.1 is typically not of interest; its main purpose is to adjust the average fitted probabilities to the proportion of ones in the data.

4.3.3 Making Predictions

Once the coefficients have been estimated, it is a simple matter to compute the probability of **default** for any given credit card balance. For example, using the coefficient estimates given in Table 4.1, we predict that the default probability for an individual with a **balance** of \$1,000 is

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}} = \frac{e^{-10.6513 + 0.0055 \times 1,000}}{1 + e^{-10.6513 + 0.0055 \times 1,000}} = 0.00576,$$

which is below 1%. In contrast, the predicted probability of default for an individual with a balance of \$2,000 is much higher, and equals 0.586 or 58.6%.

One can use qualitative predictors with the logistic regression model using the dummy variable approach from Section 3.3.1. As an example, the **Default** data set contains the qualitative variable **student**. To fit the model we simply create a dummy variable that takes on a value of 1 for students and 0 for non-students. The logistic regression model that results from predicting probability of default from student status can be seen in Table 4.2. The coefficient associated with the dummy variable is positive,

	Coefficient	Std. error	Z-statistic	P-value
Intercept	-3.5041	0.0707	-49.55	<0.0001
student[Yes]	0.4049	0.1150	3.52	0.0004

TABLE 4.2. For the **Default** data, estimated coefficients of the logistic regression model that predicts the probability of **default** using student status. Student status is encoded as a dummy variable, with a value of 1 for a student and a value of 0 for a non-student, and represented by the variable **student[Yes]** in the table.

and the associated p-value is statistically significant. This indicates that students tend to have higher default probabilities than non-students:

$$\begin{aligned}\widehat{\Pr}(\text{default}=\text{Yes}|\text{student}=\text{Yes}) &= \frac{e^{-3.5041+0.4049 \times 1}}{1 + e^{-3.5041+0.4049 \times 1}} = 0.0431, \\ \widehat{\Pr}(\text{default}=\text{Yes}|\text{student}=\text{No}) &= \frac{e^{-3.5041+0.4049 \times 0}}{1 + e^{-3.5041+0.4049 \times 0}} = 0.0292.\end{aligned}$$

4.3.4 Multiple Logistic Regression

We now consider the problem of predicting a binary response using multiple predictors. By analogy with the extension from simple to multiple linear regression in Chapter 3, we can generalize (4.4) as follows:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p, \quad (4.6)$$

where $X = (X_1, \dots, X_p)$ are p predictors. Equation 4.6 can be rewritten as

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}. \quad (4.7)$$

Just as in Section 4.3.2, we use the maximum likelihood method to estimate $\beta_0, \beta_1, \dots, \beta_p$.

Table 4.3 shows the coefficient estimates for a logistic regression model that uses **balance**, **income** (in thousands of dollars), and **student** status to predict probability of **default**. There is a surprising result here. The p-values associated with **balance** and the dummy variable for **student** status are very small, indicating that each of these variables is associated with the probability of **default**. However, the coefficient for the dummy variable is negative, indicating that students are less likely to default than non-students. In contrast, the coefficient for the dummy variable is positive in Table 4.2. How is it possible for student status to be associated with an *increase* in probability of default in Table 4.2 and a *decrease* in probability of default in Table 4.3? The left-hand panel of Figure 4.3 provides a graphical illustration of this apparent paradox. The orange and blue solid lines show the average default rates for students and non-students, respectively,

	Coefficient	Std. error	Z-statistic	P-value
Intercept	−10.8690	0.4923	−22.08	<0.0001
balance	0.0057	0.0002	24.74	<0.0001
income	0.0030	0.0082	0.37	0.7115
student[Yes]	−0.6468	0.2362	−2.74	0.0062

TABLE 4.3. For the **Default** data, estimated coefficients of the logistic regression model that predicts the probability of **default** using **balance**, **income**, and student status. Student status is encoded as a dummy variable **student[Yes]**, with a value of 1 for a student and a value of 0 for a non-student. In fitting this model, **income** was measured in thousands of dollars.

as a function of credit card balance. The negative coefficient for **student** in the multiple logistic regression indicates that for a fixed value of **balance** and **income**, a student is less likely to default than a non-student. Indeed, we observe from the left-hand panel of Figure 4.3 that the student default rate is at or below that of the non-student default rate for every value of **balance**. But the horizontal broken lines near the base of the plot, which show the default rates for students and non-students averaged over all values of **balance** and **income**, suggest the opposite effect: the overall student default rate is higher than the non-student default rate. Consequently, there is a positive coefficient for **student** in the single variable logistic regression output shown in Table 4.2.

The right-hand panel of Figure 4.3 provides an explanation for this discrepancy. The variables **student** and **balance** are correlated. Students tend to hold higher levels of debt, which is in turn associated with higher probability of default. In other words, students are more likely to have large credit card balances, which, as we know from the left-hand panel of Figure 4.3, tend to be associated with high default rates. Thus, even though an individual student with a given credit card balance will tend to have a lower probability of default than a non-student with the same credit card balance, the fact that students on the whole tend to have higher credit card balances means that overall, students tend to default at a higher rate than non-students. This is an important distinction for a credit card company that is trying to determine to whom they should offer credit. A student is riskier than a non-student if no information about the student’s credit card balance is available. However, that student is less risky than a non-student *with the same credit card balance*!

This simple example illustrates the dangers and subtleties associated with performing regressions involving only a single predictor when other predictors may also be relevant. As in the linear regression setting, the results obtained using one predictor may be quite different from those obtained using multiple predictors, especially when there is correlation among the predictors. In general, the phenomenon seen in Figure 4.3 is known as *confounding*.

confounding

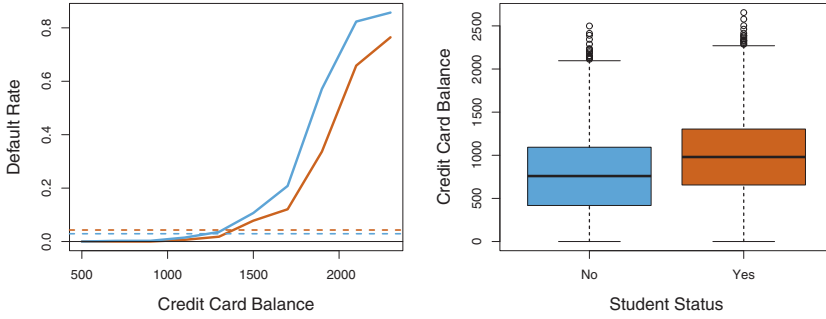


FIGURE 4.3. *Confounding in the **Default** data. Left: Default rates are shown for students (orange) and non-students (blue). The solid lines display default rate as a function of **balance**, while the horizontal broken lines display the overall default rates. Right: Boxplots of **balance** for students (orange) and non-students (blue) are shown.*

By substituting estimates for the regression coefficients from Table 4.3 into (4.7), we can make predictions. For example, a student with a credit card balance of \$1,500 and an income of \$40,000 has an estimated probability of default of

$$\hat{p}(X) = \frac{e^{-10.869 + 0.00574 \times 1,500 + 0.003 \times 40 - 0.6468 \times 1}}{1 + e^{-10.869 + 0.00574 \times 1,500 + 0.003 \times 40 - 0.6468 \times 1}} = 0.058. \quad (4.8)$$

A non-student with the same balance and income has an estimated probability of default of

$$\hat{p}(X) = \frac{e^{-10.869 + 0.00574 \times 1,500 + 0.003 \times 40 - 0.6468 \times 0}}{1 + e^{-10.869 + 0.00574 \times 1,500 + 0.003 \times 40 - 0.6468 \times 0}} = 0.105. \quad (4.9)$$

(Here we multiply the **income** coefficient estimate from Table 4.3 by 40, rather than by 40,000, because in that table the model was fit with **income** measured in units of \$1,000.)

4.3.5 Logistic Regression for >2 Response Classes

We sometimes wish to classify a response variable that has more than two classes. For example, in Section 4.2 we had three categories of medical condition in the emergency room: **stroke**, **drug overdose**, **epileptic seizure**. In this setting, we wish to model both $\Pr(Y = \text{stroke}|X)$ and $\Pr(Y = \text{drug overdose}|X)$, with the remaining $\Pr(Y = \text{epileptic seizure}|X) = 1 - \Pr(Y = \text{stroke}|X) - \Pr(Y = \text{drug overdose}|X)$. The two-class logistic regression models discussed in the previous sections have multiple-class extensions, but in practice they tend not to be used all that often. One of the reasons is that the method we discuss in the next section, *discriminant*

analysis, is popular for multiple-class classification. So we do not go into the details of multiple-class logistic regression here, but simply note that such an approach is possible, and that software for it is available in **R**.

4.4 Linear Discriminant Analysis

Logistic regression involves directly modeling $\Pr(Y = k|X = x)$ using the logistic function, given by (4.7) for the case of two response classes. In statistical jargon, we model the conditional distribution of the response Y , given the predictor(s) X . We now consider an alternative and less direct approach to estimating these probabilities. In this alternative approach, we model the distribution of the predictors X separately in each of the response classes (i.e. given Y), and then use Bayes' theorem to flip these around into estimates for $\Pr(Y = k|X = x)$. When these distributions are assumed to be normal, it turns out that the model is very similar in form to logistic regression.

Why do we need another method, when we have logistic regression? There are several reasons:

- When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.
- If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
- As mentioned in Section 4.3.5, linear discriminant analysis is popular when we have more than two response classes.

4.4.1 Using Bayes' Theorem for Classification

Suppose that we wish to classify an observation into one of K classes, where $K \geq 2$. In other words, the qualitative response variable Y can take on K possible distinct and unordered values. Let π_k represent the overall or *prior* probability that a randomly chosen observation comes from the k th class; this is the probability that a given observation is associated with the k th category of the response variable Y . Let $f_k(X) \equiv \Pr(X = x|Y = k)$ denote the *density function* of X for an observation that comes from the k th class. In other words, $f_k(x)$ is relatively large if there is a high probability that an observation in the k th class has $X \approx x$, and $f_k(x)$ is small if it is very

unlikely that an observation in the k th class has $X \approx x$. Then *Bayes' theorem* states that

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^K \pi_l f_l(x)}. \quad (4.10)$$
Bayes' theorem

In accordance with our earlier notation, we will use the abbreviation $p_k(X) = \Pr(Y = k|X)$. This suggests that instead of directly computing $p_k(X)$ as in Section 4.3.1, we can simply plug in estimates of π_k and $f_k(X)$ into (4.10). In general, estimating π_k is easy if we have a random sample of Y s from the population: we simply compute the fraction of the training observations that belong to the k th class. However, estimating $f_k(X)$ tends to be more challenging, unless we assume some simple forms for these densities. We refer to $p_k(x)$ as the *posterior* probability that an observation $X = x$ belongs to the k th class. That is, it is the probability that the observation belongs to the k th class, *given* the predictor value for that observation. posterior

We know from Chapter 2 that the Bayes classifier, which classifies an observation to the class for which $p_k(X)$ is largest, has the lowest possible error rate out of all classifiers. (This is of course only true if the terms in (4.10) are all correctly specified.) Therefore, if we can find a way to estimate $f_k(X)$, then we can develop a classifier that approximates the Bayes classifier. Such an approach is the topic of the following sections.

4.4.2 Linear Discriminant Analysis for $p = 1$

For now, assume that $p = 1$ —that is, we have only one predictor. We would like to obtain an estimate for $f_k(x)$ that we can plug into (4.10) in order to estimate $p_k(x)$. We will then classify an observation to the class for which $p_k(x)$ is greatest. In order to estimate $f_k(x)$, we will first make some assumptions about its form.

Suppose we assume that $f_k(x)$ is *normal* or *Gaussian*. In the one-dimensional setting, the normal density takes the form normal Gaussian

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right), \quad (4.11)$$

where μ_k and σ_k^2 are the mean and variance parameters for the k th class. For now, let us further assume that $\sigma_1^2 = \dots = \sigma_K^2$: that is, there is a shared variance term across all K classes, which for simplicity we can denote by σ^2 . Plugging (4.11) into (4.10), we find that

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_l)^2\right)}. \quad (4.12)$$

(Note that in (4.12), π_k denotes the prior probability that an observation belongs to the k th class, not to be confused with $\pi \approx 3.14159$, the mathematical constant.) The Bayes classifier involves assigning an observation

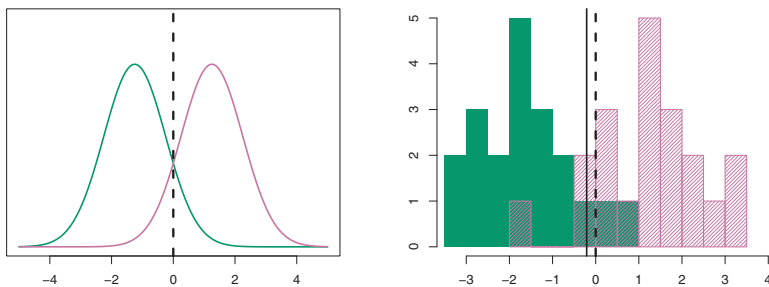


FIGURE 4.4. Left: Two one-dimensional normal density functions are shown. The dashed vertical line represents the Bayes decision boundary. Right: 20 observations were drawn from each of the two classes, and are shown as histograms. The Bayes decision boundary is again shown as a dashed vertical line. The solid vertical line represents the LDA decision boundary estimated from the training data.

$X = x$ to the class for which (4.12) is largest. Taking the log of (4.12) and rearranging the terms, it is not hard to show that this is equivalent to assigning the observation to the class for which

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \quad (4.13)$$

is largest. For instance, if $K = 2$ and $\pi_1 = \pi_2$, then the Bayes classifier assigns an observation to class 1 if $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$, and to class 2 otherwise. In this case, the Bayes decision boundary corresponds to the point where

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}. \quad (4.14)$$

An example is shown in the left-hand panel of Figure 4.4. The two normal density functions that are displayed, $f_1(x)$ and $f_2(x)$, represent two distinct classes. The mean and variance parameters for the two density functions are $\mu_1 = -1.25$, $\mu_2 = 1.25$, and $\sigma_1^2 = \sigma_2^2 = 1$. The two densities overlap, and so given that $X = x$, there is some uncertainty about the class to which the observation belongs. If we assume that an observation is equally likely to come from either class—that is, $\pi_1 = \pi_2 = 0.5$ —then by inspection of (4.14), we see that the Bayes classifier assigns the observation to class 1 if $x < 0$ and class 2 otherwise. Note that in this case, we can compute the Bayes classifier because we know that X is drawn from a Gaussian distribution within each class, and we know all of the parameters involved. In a real-life situation, we are not able to calculate the Bayes classifier.

In practice, even if we are quite certain of our assumption that X is drawn from a Gaussian distribution within each class, we still have to estimate the parameters μ_1, \dots, μ_K , π_1, \dots, π_K , and σ^2 . The *linear discriminant*

analysis (LDA) method approximates the Bayes classifier by plugging estimates for π_k , μ_k , and σ^2 into (4.13). In particular, the following estimates are used:

linear
discriminant
analysis

$$\begin{aligned}\hat{\mu}_k &= \frac{1}{n_k} \sum_{i:y_i=k} x_i \\ \hat{\sigma}^2 &= \frac{1}{n-K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2\end{aligned}\quad (4.15)$$

where n is the total number of training observations, and n_k is the number of training observations in the k th class. The estimate for μ_k is simply the average of all the training observations from the k th class, while $\hat{\sigma}^2$ can be seen as a weighted average of the sample variances for each of the K classes. Sometimes we have knowledge of the class membership probabilities π_1, \dots, π_K , which can be used directly. In the absence of any additional information, LDA estimates π_k using the proportion of the training observations that belong to the k th class. In other words,

$$\hat{\pi}_k = n_k/n. \quad (4.16)$$

The LDA classifier plugs the estimates given in (4.15) and (4.16) into (4.13), and assigns an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k) \quad (4.17)$$

is largest. The word *linear* in the classifier's name stems from the fact that the *discriminant functions* $\hat{\delta}_k(x)$ in (4.17) are linear functions of x (as opposed to a more complex function of x).

discriminant
function

The right-hand panel of Figure 4.4 displays a histogram of a random sample of 20 observations from each class. To implement LDA, we began by estimating π_k , μ_k , and σ^2 using (4.15) and (4.16). We then computed the decision boundary, shown as a black solid line, that results from assigning an observation to the class for which (4.17) is largest. All points to the left of this line will be assigned to the green class, while points to the right of this line are assigned to the purple class. In this case, since $n_1 = n_2 = 20$, we have $\hat{\pi}_1 = \hat{\pi}_2$. As a result, the decision boundary corresponds to the midpoint between the sample means for the two classes, $(\hat{\mu}_1 + \hat{\mu}_2)/2$. The figure indicates that the LDA decision boundary is slightly to the left of the optimal Bayes decision boundary, which instead equals $(\mu_1 + \mu_2)/2 = 0$. How well does the LDA classifier perform on this data? Since this is simulated data, we can generate a large number of test observations in order to compute the Bayes error rate and the LDA test error rate. These are 10.6% and 11.1%, respectively. In other words, the LDA classifier's error rate is only 0.5% above the smallest possible error rate! This indicates that LDA is performing pretty well on this data set.

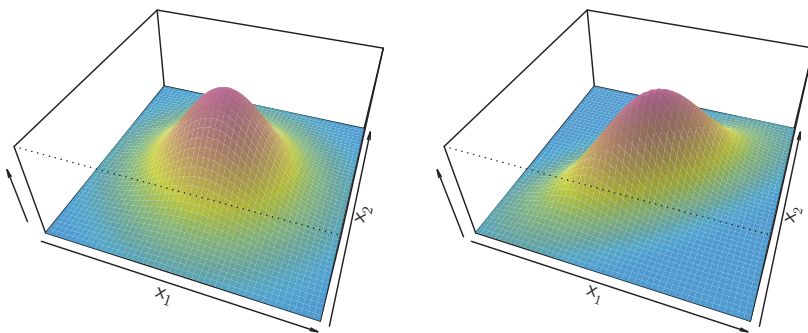


FIGURE 4.5. Two multivariate Gaussian density functions are shown, with $p = 2$. Left: The two predictors are uncorrelated. Right: The two variables have a correlation of 0.7.

To reiterate, the LDA classifier results from assuming that the observations within each class come from a normal distribution with a class-specific mean vector and a common variance σ^2 , and plugging estimates for these parameters into the Bayes classifier. In Section 4.4.4, we will consider a less stringent set of assumptions, by allowing the observations in the k th class to have a class-specific variance, σ_k^2 .

4.4.3 Linear Discriminant Analysis for $p > 1$

We now extend the LDA classifier to the case of multiple predictors. To do this, we will assume that $X = (X_1, X_2, \dots, X_p)$ is drawn from a *multivariate Gaussian* (or multivariate normal) distribution, with a class-specific mean vector and a common covariance matrix. We begin with a brief review of such a distribution.

multivariate
Gaussian

The multivariate Gaussian distribution assumes that each individual predictor follows a one-dimensional normal distribution, as in (4.11), with some correlation between each pair of predictors. Two examples of multivariate Gaussian distributions with $p = 2$ are shown in Figure 4.5. The height of the surface at any particular point represents the probability that both X_1 and X_2 fall in a small region around that point. In either panel, if the surface is cut along the X_1 axis or along the X_2 axis, the resulting cross-section will have the shape of a one-dimensional normal distribution. The left-hand panel of Figure 4.5 illustrates an example in which $\text{Var}(X_1) = \text{Var}(X_2)$ and $\text{Cor}(X_1, X_2) = 0$; this surface has a characteristic *bell shape*. However, the bell shape will be distorted if the predictors are correlated or have unequal variances, as is illustrated in the right-hand panel of Figure 4.5. In this situation, the base of the bell will have an elliptical, rather than circular,

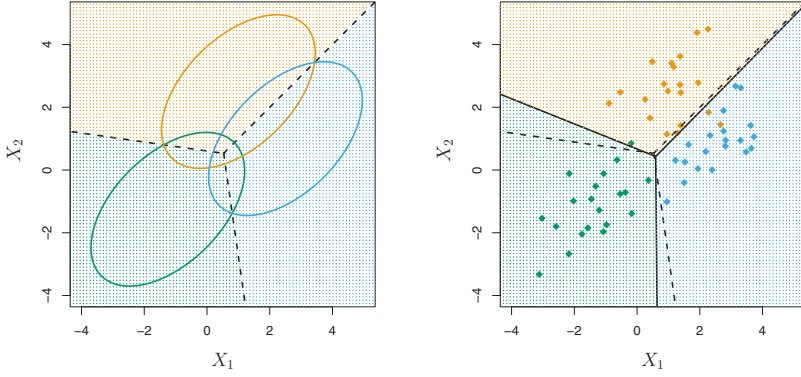


FIGURE 4.6. An example with three classes. The observations from each class are drawn from a multivariate Gaussian distribution with $p = 2$, with a class-specific mean vector and a common covariance matrix. Left: Ellipses that contain 95 % of the probability for each of the three classes are shown. The dashed lines are the Bayes decision boundaries. Right: 20 observations were generated from each class, and the corresponding LDA decision boundaries are indicated using solid black lines. The Bayes decision boundaries are once again shown as dashed lines.

shape. To indicate that a p -dimensional random variable X has a multivariate Gaussian distribution, we write $X \sim N(\mu, \Sigma)$. Here $E(X) = \mu$ is the mean of X (a vector with p components), and $\text{Cov}(X) = \Sigma$ is the $p \times p$ covariance matrix of X . Formally, the multivariate Gaussian density is defined as

$$f(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right). \quad (4.18)$$

In the case of $p > 1$ predictors, the LDA classifier assumes that the observations in the k th class are drawn from a multivariate Gaussian distribution $N(\mu_k, \Sigma)$, where μ_k is a class-specific mean vector, and Σ is a covariance matrix that is common to all K classes. Plugging the density function for the k th class, $f_k(X = x)$, into (4.10) and performing a little bit of algebra reveals that the Bayes classifier assigns an observation $X = x$ to the class for which

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \quad (4.19)$$

is largest. This is the vector/matrix version of (4.13).

An example is shown in the left-hand panel of Figure 4.6. Three equally-sized Gaussian classes are shown with class-specific mean vectors and a common covariance matrix. The three ellipses represent regions that contain 95 % of the probability for each of the three classes. The dashed lines

are the Bayes decision boundaries. In other words, they represent the set of values x for which $\delta_k(x) = \delta_\ell(x)$; i.e.

$$x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k = x^T \Sigma^{-1} \mu_l - \frac{1}{2} \mu_l^T \Sigma^{-1} \mu_l \quad (4.20)$$

for $k \neq l$. (The $\log \pi_k$ term from (4.19) has disappeared because each of the three classes has the same number of training observations; i.e. π_k is the same for each class.) Note that there are three lines representing the Bayes decision boundaries because there are three *pairs of classes* among the three classes. That is, one Bayes decision boundary separates class 1 from class 2, one separates class 1 from class 3, and one separates class 2 from class 3. These three Bayes decision boundaries divide the predictor space into three regions. The Bayes classifier will classify an observation according to the region in which it is located.

Once again, we need to estimate the unknown parameters μ_1, \dots, μ_K , π_1, \dots, π_K , and Σ ; the formulas are similar to those used in the one-dimensional case, given in (4.15). To assign a new observation $X = x$, LDA plugs these estimates into (4.19) and classifies to the class for which $\hat{\delta}_k(x)$ is largest. Note that in (4.19) $\delta_k(x)$ is a linear function of x ; that is, the LDA decision rule depends on x only through a linear combination of its elements. Once again, this is the reason for the word *linear* in LDA.

In the right-hand panel of Figure 4.6, 20 observations drawn from each of the three classes are displayed, and the resulting LDA decision boundaries are shown as solid black lines. Overall, the LDA decision boundaries are pretty close to the Bayes decision boundaries, shown again as dashed lines. The test error rates for the Bayes and LDA classifiers are 0.0746 and 0.0770, respectively. This indicates that LDA is performing well on this data.

We can perform LDA on the **Default** data in order to predict whether or not an individual will default on the basis of credit card balance and student status. The LDA model fit to the 10,000 training samples results in a *training* error rate of 2.75%. This sounds like a low error rate, but two caveats must be noted.

- First of all, training error rates will usually be lower than test error rates, which are the real quantity of interest. In other words, we might expect this classifier to perform worse if we use it to predict whether or not a new set of individuals will default. The reason is that we specifically adjust the parameters of our model to do well on the training data. The higher the ratio of parameters p to number of samples n , the more we expect this *overfitting* to play a role. For these data we don't expect this to be a problem, since $p = 4$ and $n = 10,000$.

overfitting

- Second, since only 3.33% of the individuals in the training sample defaulted, a simple but useless classifier that always predicts that

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

TABLE 4.4. A confusion matrix compares the LDA predictions to the true default statuses for the 10,000 training observations in the **Default** data set. Elements on the diagonal of the matrix represent individuals whose default statuses were correctly predicted, while off-diagonal elements represent individuals that were misclassified. LDA made incorrect predictions for 23 individuals who did not default and for 252 individuals who did default.

each individual will not default, regardless of his or her credit card balance and student status, will result in an error rate of 3.33 %. In other words, the trivial *null* classifier will achieve an error rate that is only a bit higher than the LDA training set error rate.

In practice, a binary classifier such as this one can make two types of errors: it can incorrectly assign an individual who defaults to the *no default* category, or it can incorrectly assign an individual who does not default to the *default* category. It is often of interest to determine which of these two types of errors are being made. A *confusion matrix*, shown for the **Default** data in Table 4.4, is a convenient way to display this information. The table reveals that LDA predicted that a total of 104 people would default. Of these people, 81 actually defaulted and 23 did not. Hence only 23 out of 9,667 of the individuals who did not default were incorrectly labeled. This looks like a pretty low error rate! However, of the 333 individuals who defaulted, 252 (or 75.7 %) were missed by LDA. So while the overall error rate is low, the error rate among individuals who defaulted is very high. From the perspective of a credit card company that is trying to identify high-risk individuals, an error rate of $252/333 = 75.7\%$ among individuals who default may well be unacceptable.

Class-specific performance is also important in medicine and biology, where the terms *sensitivity* and *specificity* characterize the performance of a classifier or screening test. In this case the sensitivity is the percentage of true defaulters that are identified, a low 24.3 % in this case. The specificity is the percentage of non-defaulters that are correctly identified, here $(1 - 23/9,667) \times 100 = 99.8\%$.

Why does LDA do such a poor job of classifying the customers who default? In other words, why does it have such a low sensitivity? As we have seen, LDA is trying to approximate the Bayes classifier, which has the lowest *total* error rate out of all classifiers (if the Gaussian model is correct). That is, the Bayes classifier will yield the smallest possible total number of misclassified observations, irrespective of which class the errors come from. That is, some misclassifications will result from incorrectly assigning

		<i>True default status</i>		
		No	Yes	Total
<i>Predicted default status</i>	No	9,432	138	9,570
	Yes	235	195	430
	Total	9,667	333	10,000

TABLE 4.5. A confusion matrix compares the LDA predictions to the true default statuses for the 10,000 training observations in the **Default** data set, using a modified threshold value that predicts default for any individuals whose posterior default probability exceeds 20 %.

a customer who does not default to the default class, and others will result from incorrectly assigning a customer who defaults to the non-default class. In contrast, a credit card company might particularly wish to avoid incorrectly classifying an individual who will default, whereas incorrectly classifying an individual who will not default, though still to be avoided, is less problematic. We will now see that it is possible to modify LDA in order to develop a classifier that better meets the credit card company's needs.

The Bayes classifier works by assigning an observation to the class for which the posterior probability $p_k(X)$ is greatest. In the two-class case, this amounts to assigning an observation to the *default* class if

$$\Pr(\text{default} = \text{Yes} | X = x) > 0.5. \quad (4.21)$$

Thus, the Bayes classifier, and by extension LDA, uses a threshold of 50 % for the posterior probability of default in order to assign an observation to the *default* class. However, if we are concerned about incorrectly predicting the default status for individuals who default, then we can consider lowering this threshold. For instance, we might label any customer with a posterior probability of default above 20 % to the *default* class. In other words, instead of assigning an observation to the *default* class if (4.21) holds, we could instead assign an observation to this class if

$$\Pr(\text{default} = \text{Yes} | X = x) > 0.2. \quad (4.22)$$

The error rates that result from taking this approach are shown in Table 4.5. Now LDA predicts that 430 individuals will default. Of the 333 individuals who default, LDA correctly predicts all but 138, or 41.4 %. This is a vast improvement over the error rate of 75.7 % that resulted from using the threshold of 50 %. However, this improvement comes at a cost: now 235 individuals who do not default are incorrectly classified. As a result, the overall error rate has increased slightly to 3.73 %. But a credit card company may consider this slight increase in the total error rate to be a small price to pay for more accurate identification of individuals who do indeed default.

Figure 4.7 illustrates the trade-off that results from modifying the threshold value for the posterior probability of default. Various error rates are

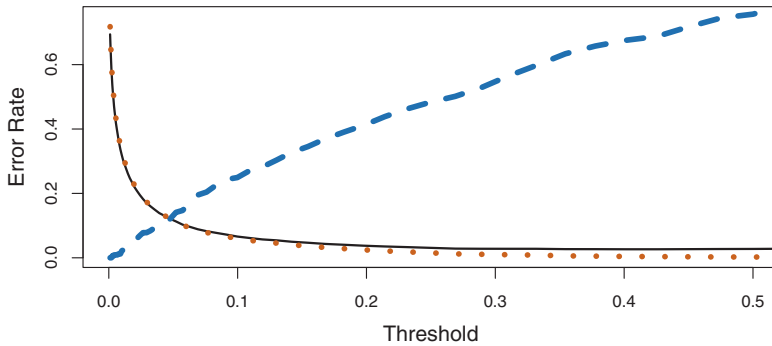


FIGURE 4.7. For the **Default** data set, error rates are shown as a function of the threshold value for the posterior probability that is used to perform the assignment. The black solid line displays the overall error rate. The blue dashed line represents the fraction of defaulting customers that are incorrectly classified, and the orange dotted line indicates the fraction of errors among the non-defaulting customers.

shown as a function of the threshold value. Using a threshold of 0.5, as in (4.21), minimizes the overall error rate, shown as a black solid line. This is to be expected, since the Bayes classifier uses a threshold of 0.5 and is known to have the lowest overall error rate. But when a threshold of 0.5 is used, the error rate among the individuals who default is quite high (blue dashed line). As the threshold is reduced, the error rate among individuals who default decreases steadily, but the error rate among the individuals who do not default increases. How can we decide which threshold value is best? Such a decision must be based on *domain knowledge*, such as detailed information about the costs associated with default.

The *ROC curve* is a popular graphic for simultaneously displaying the two types of errors for all possible thresholds. The name “ROC” is historic, and comes from communications theory. It is an acronym for *receiver operating characteristics*. Figure 4.8 displays the ROC curve for the LDA classifier on the training data. The overall performance of a classifier, summarized over all possible thresholds, is given by the *area under the (ROC) curve* (AUC). An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier. For this data the AUC is 0.95, which is close to the maximum of one so would be considered very good. We expect a classifier that performs no better than chance to have an AUC of 0.5 (when evaluated on an independent test set not used in model training). ROC curves are useful for comparing different classifiers, since they take into account all possible thresholds. It turns out that the ROC curve for the logistic regression model of Section 4.3.4 fit to these data is virtually indistinguishable from this one for the LDA model, so we do not display it here.

As we have seen above, varying the classifier threshold changes its true positive and false positive rate. These are also called the *sensitivity* and one

ROC curve

area under the (ROC) curve

sensitivity

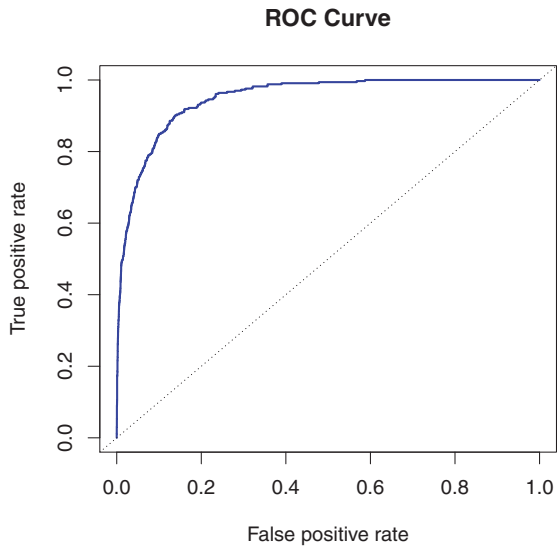


FIGURE 4.8. A ROC curve for the LDA classifier on the **Default** data. It traces out two types of error as we vary the threshold value for the posterior probability of default. The actual thresholds are not shown. The true positive rate is the sensitivity: the fraction of defaulters that are correctly identified, using a given threshold value. The false positive rate is 1-specificity: the fraction of non-defaulters that we classify incorrectly as defaulters, using that same threshold value. The ideal ROC curve hugs the top left corner, indicating a high true positive rate and a low false positive rate. The dotted line represents the “no information” classifier; this is what we would expect if student status and credit card balance are not associated with probability of default.

		Predicted class		
		– or Null	+ or Non-null	Total
True class	– or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
Total		N*	P*	

TABLE 4.6. Possible results when applying a classifier or diagnostic test to a population.

minus the *specificity* of our classifier. Since there is an almost bewildering array of terms used in this context, we now give a summary. Table 4.6 shows the possible results when applying a classifier (or diagnostic test) to a population. To make the connection with the epidemiology literature, we think of “+” as the “disease” that we are trying to detect, and “–” as the “non-disease” state. To make the connection to the classical hypothesis testing literature, we think of “–” as the null hypothesis and “+” as the alternative (non-null) hypothesis. In the context of the **Default** data, “+” indicates an individual who defaults, and “–” indicates one who does not.

Name	Definition	Synonyms
False Pos. rate	FP/N	Type I error, 1–Specificity
True Pos. rate	TP/P	1–Type II error, power, sensitivity, recall
Pos. Pred. value	TP/P*	Precision, 1–false discovery proportion
Neg. Pred. value	TN/N*	

TABLE 4.7. Important measures for classification and diagnostic testing, derived from quantities in Table 4.6.

Table 4.7 lists many of the popular performance measures that are used in this context. The denominators for the false positive and true positive rates are the actual population counts in each class. In contrast, the denominators for the positive predictive value and the negative predictive value are the total predicted counts for each class.

4.4.4 Quadratic Discriminant Analysis

As we have discussed, LDA assumes that the observations within each class are drawn from a multivariate Gaussian distribution with a class-specific mean vector and a covariance matrix that is common to all K classes. *Quadratic discriminant analysis* (QDA) provides an alternative approach. Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a Gaussian distribution, and plugging estimates for the parameters into Bayes' theorem in order to perform prediction. However, unlike LDA, QDA assumes that each class has its own covariance matrix. That is, it assumes that an observation from the k th class is of the form $X \sim N(\mu_k, \Sigma_k)$, where Σ_k is a covariance matrix for the k th class. Under this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

quadratic
discriminant
analysis

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k) + \log \pi_k \\ &= -\frac{1}{2}x^T \Sigma_k^{-1}x + x^T \Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma_k^{-1}\mu_k + \log \pi_k \quad (4.23)\end{aligned}$$

is largest. So the QDA classifier involves plugging estimates for Σ_k , μ_k , and π_k into (4.23), and then assigning an observation $X = x$ to the class for which this quantity is largest. Unlike in (4.19), the quantity x appears as a *quadratic* function in (4.23). This is where QDA gets its name.

Why does it matter whether or not we assume that the K classes share a common covariance matrix? In other words, why would one prefer LDA to QDA, or vice-versa? The answer lies in the bias-variance trade-off. When there are p predictors, then estimating a covariance matrix requires estimating $p(p+1)/2$ parameters. QDA estimates a separate covariance matrix for each class, for a total of $Kp(p+1)/2$ parameters. With 50 predictors this is some multiple of 1,225, which is a lot of parameters. By instead assuming that the K classes share a common covariance matrix, the LDA model

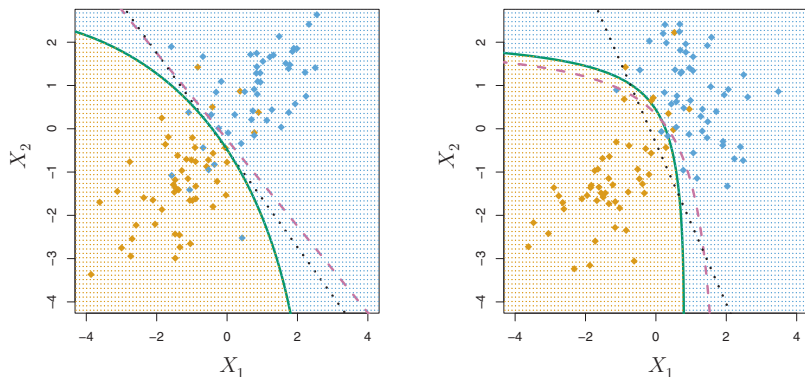


FIGURE 4.9. Left: The Bayes (purple dashed), LDA (black dotted), and QDA (green solid) decision boundaries for a two-class problem with $\Sigma_1 = \Sigma_2$. The shading indicates the QDA decision rule. Since the Bayes decision boundary is linear, it is more accurately approximated by LDA than by QDA. Right: Details are as given in the left-hand panel, except that $\Sigma_1 \neq \Sigma_2$. Since the Bayes decision boundary is non-linear, it is more accurately approximated by QDA than by LDA.

becomes linear in x , which means there are Kp linear coefficients to estimate. Consequently, LDA is a much less flexible classifier than QDA, and so has substantially lower variance. This can potentially lead to improved prediction performance. But there is a trade-off: if LDA's assumption that the K classes share a common covariance matrix is badly off, then LDA can suffer from high bias. Roughly speaking, LDA tends to be a better bet than QDA if there are relatively few training observations and so reducing variance is crucial. In contrast, QDA is recommended if the training set is very large, so that the variance of the classifier is not a major concern, or if the assumption of a common covariance matrix for the K classes is clearly untenable.

Figure 4.9 illustrates the performances of LDA and QDA in two scenarios. In the left-hand panel, the two Gaussian classes have a common correlation of 0.7 between X_1 and X_2 . As a result, the Bayes decision boundary is linear and is accurately approximated by the LDA decision boundary. The QDA decision boundary is inferior, because it suffers from higher variance without a corresponding decrease in bias. In contrast, the right-hand panel displays a situation in which the orange class has a correlation of 0.7 between the variables and the blue class has a correlation of -0.7 . Now the Bayes decision boundary is quadratic, and so QDA more accurately approximates this boundary than does LDA.

4.5 A Comparison of Classification Methods

In this chapter, we have considered three different classification approaches: logistic regression, LDA, and QDA. In Chapter 2, we also discussed the K -nearest neighbors (KNN) method. We now consider the types of scenarios in which one approach might dominate the others.

Though their motivations differ, the logistic regression and LDA methods are closely connected. Consider the two-class setting with $p = 1$ predictor, and let $p_1(x)$ and $p_2(x) = 1 - p_1(x)$ be the probabilities that the observation $X = x$ belongs to class 1 and class 2, respectively. In the LDA framework, we can see from (4.12) to (4.13) (and a bit of simple algebra) that the log odds is given by

$$\log \left(\frac{p_1(x)}{1 - p_1(x)} \right) = \log \left(\frac{p_1(x)}{p_2(x)} \right) = c_0 + c_1 x, \quad (4.24)$$

where c_0 and c_1 are functions of μ_1, μ_2 , and σ^2 . From (4.4), we know that in logistic regression,

$$\log \left(\frac{p_1}{1 - p_1} \right) = \beta_0 + \beta_1 x. \quad (4.25)$$

Both (4.24) and (4.25) are linear functions of x . Hence, both logistic regression and LDA produce linear decision boundaries. The only difference between the two approaches lies in the fact that β_0 and β_1 are estimated using maximum likelihood, whereas c_0 and c_1 are computed using the estimated mean and variance from a normal distribution. This same connection between LDA and logistic regression also holds for multidimensional data with $p > 1$.

Since logistic regression and LDA differ only in their fitting procedures, one might expect the two approaches to give similar results. This is often, but not always, the case. LDA assumes that the observations are drawn from a Gaussian distribution with a common covariance matrix in each class, and so can provide some improvements over logistic regression when this assumption approximately holds. Conversely, logistic regression can outperform LDA if these Gaussian assumptions are not met.

Recall from Chapter 2 that KNN takes a completely different approach from the classifiers seen in this chapter. In order to make a prediction for an observation $X = x$, the K training observations that are closest to x are identified. Then X is assigned to the class to which the plurality of these observations belong. Hence KNN is a completely non-parametric approach: no assumptions are made about the shape of the decision boundary. Therefore, we can expect this approach to dominate LDA and logistic regression when the decision boundary is highly non-linear. On the other hand, KNN does not tell us which predictors are important; we don't get a table of coefficients as in Table 4.3.

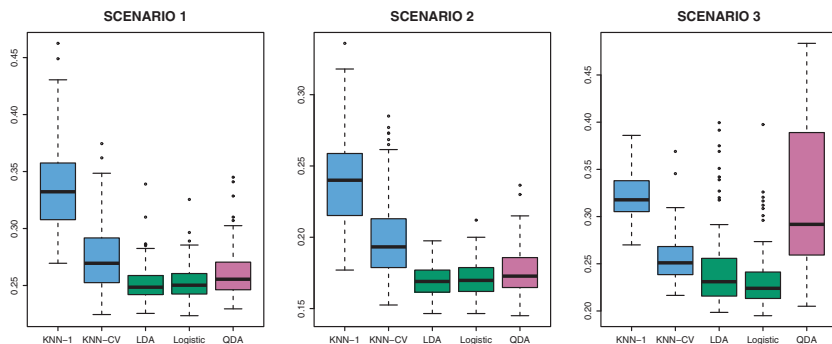


FIGURE 4.10. *Boxplots of the test error rates for each of the linear scenarios described in the main text.*

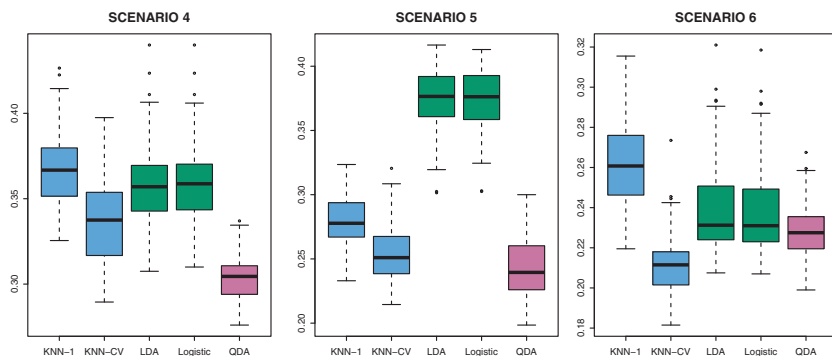


FIGURE 4.11. *Boxplots of the test error rates for each of the non-linear scenarios described in the main text.*

Finally, QDA serves as a compromise between the non-parametric KNN method and the linear LDA and logistic regression approaches. Since QDA assumes a quadratic decision boundary, it can accurately model a wider range of problems than can the linear methods. Though not as flexible as KNN, QDA can perform better in the presence of a limited number of training observations because it does make some assumptions about the form of the decision boundary.

To illustrate the performances of these four classification approaches, we generated data from six different scenarios. In three of the scenarios, the Bayes decision boundary is linear, and in the remaining scenarios it is non-linear. For each scenario, we produced 100 random training data sets. On each of these training sets, we fit each method to the data and computed the resulting test error rate on a large test set. Results for the linear scenarios are shown in Figure 4.10, and the results for the non-linear scenarios are in Figure 4.11. The KNN method requires selection of K , the number of neighbors. We performed KNN with two values of K : $K = 1$,

and a value of K that was chosen automatically using an approach called *cross-validation*, which we discuss further in Chapter 5.

In each of the six scenarios, there were $p = 2$ predictors. The scenarios were as follows:

Scenario 1: There were 20 training observations in each of two classes. The observations within each class were uncorrelated random normal variables with a different mean in each class. The left-hand panel of Figure 4.10 shows that LDA performed well in this setting, as one would expect since this is the model assumed by LDA. KNN performed poorly because it paid a price in terms of variance that was not offset by a reduction in bias. QDA also performed worse than LDA, since it fit a more flexible classifier than necessary. Since logistic regression assumes a linear decision boundary, its results were only slightly inferior to those of LDA.

Scenario 2: Details are as in Scenario 1, except that within each class, the two predictors had a correlation of -0.5 . The center panel of Figure 4.10 indicates little change in the relative performances of the methods as compared to the previous scenario.

Scenario 3: We generated X_1 and X_2 from the t -distribution, with 50 observations per class. The t -distribution has a similar shape to the normal distribution, but it has a tendency to yield more extreme points—that is, more points that are far from the mean. In this setting, the decision boundary was still linear, and so fit into the logistic regression framework. The set-up violated the assumptions of LDA, since the observations were not drawn from a normal distribution. The right-hand panel of Figure 4.10 shows that logistic regression outperformed LDA, though both methods were superior to the other approaches. In particular, the QDA results deteriorated considerably as a consequence of non-normality.

Scenario 4: The data were generated from a normal distribution, with a correlation of 0.5 between the predictors in the first class, and correlation of -0.5 between the predictors in the second class. This setup corresponded to the QDA assumption, and resulted in quadratic decision boundaries. The left-hand panel of Figure 4.11 shows that QDA outperformed all of the other approaches.

Scenario 5: Within each class, the observations were generated from a normal distribution with uncorrelated predictors. However, the responses were sampled from the logistic function using X_1^2 , X_2^2 , and $X_1 \times X_2$ as predictors. Consequently, there is a quadratic decision boundary. The center panel of Figure 4.11 indicates that QDA once again performed best, followed closely by KNN-CV. The linear methods had poor performance.

Scenario 6: Details are as in the previous scenario, but the responses were sampled from a more complicated non-linear function. As a result, even the quadratic decision boundaries of QDA could not adequately model the data. The right-hand panel of Figure 4.11 shows that QDA gave slightly better results than the linear methods, while the much more flexible KNN-CV method gave the best results. But KNN with $K = 1$ gave the worst results out of all methods. This highlights the fact that even when the data exhibits a complex non-linear relationship, a non-parametric method such as KNN can still give poor results if the level of smoothness is not chosen correctly.

These six examples illustrate that no one method will dominate the others in every situation. When the true decision boundaries are linear, then the LDA and logistic regression approaches will tend to perform well. When the boundaries are moderately non-linear, QDA may give better results. Finally, for much more complicated decision boundaries, a non-parametric approach such as KNN can be superior. But the level of smoothness for a non-parametric approach must be chosen carefully. In the next chapter we examine a number of approaches for choosing the correct level of smoothness and, in general, for selecting the best overall method.

Finally, recall from Chapter 3 that in the regression setting we can accommodate a non-linear relationship between the predictors and the response by performing regression using transformations of the predictors. A similar approach could be taken in the classification setting. For instance, we could create a more flexible version of logistic regression by including X^2 , X^3 , and even X^4 as predictors. This may or may not improve logistic regression's performance, depending on whether the increase in variance due to the added flexibility is offset by a sufficiently large reduction in bias. We could do the same for LDA. If we added all possible quadratic terms and cross-products to LDA, the form of the model would be the same as the QDA model, although the parameter estimates would be different. This device allows us to move somewhere between an LDA and a QDA model.

4.6 Lab: Logistic Regression, LDA, QDA, and KNN

4.6.1 The Stock Market Data

We will begin by examining some numerical and graphical summaries of the `Smarket` data, which is part of the `ISLR` library. This data set consists of percentage returns for the S&P 500 stock index over 1,250 days, from the beginning of 2001 until the end of 2005. For each date, we have recorded the percentage returns for each of the five previous trading days, `Lag1` through `Lag5`. We have also recorded `Volume` (the number of shares traded

on the previous day, in billions), **Today** (the percentage return on the date in question) and **Direction** (whether the market was **Up** or **Down** on this date).

```
> library(ISLR)
> names(Smarket)
[1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"
[6] "Lag5"      "Volume"    "Today"     "Direction"
> dim(Smarket)
[1] 1250      9
> summary(Smarket)
      Year      Lag1      Lag2
Min.   :2001   Min.   : -4.92200   Min.   : -4.92200
1st Qu.:2002   1st Qu.: -0.63950   1st Qu.: -0.63950
Median :2003   Median :  0.03900   Median :  0.03900
Mean   :2003   Mean   :  0.00383   Mean   :  0.00392
3rd Qu.:2004   3rd Qu.:  0.59675   3rd Qu.:  0.59675
Max.   :2005   Max.   :  5.73300   Max.   :  5.73300

      Lag3      Lag4      Lag5
Min.   : -4.92200   Min.   : -4.92200   Min.   : -4.92200
1st Qu.: -0.64000   1st Qu.: -0.64000   1st Qu.: -0.64000
Median :  0.03850   Median :  0.03850   Median :  0.03850
Mean   :  0.00172   Mean   :  0.00164   Mean   :  0.00561
3rd Qu.:  0.59675   3rd Qu.:  0.59675   3rd Qu.:  0.59700
Max.   :  5.73300   Max.   :  5.73300   Max.   :  5.73300

      Volume      Today      Direction
Min.   :0.356    Min.   : -4.92200   Down:602
1st Qu.:1.257    1st Qu.: -0.63950   Up  :648
Median :1.423    Median :  0.03850
Mean   :1.478    Mean   :  0.00314
3rd Qu.:1.642    3rd Qu.:  0.59675
Max.   :3.152    Max.   :  5.73300
> pairs(Smarket)
```

The `cor()` function produces a matrix that contains all of the pairwise correlations among the predictors in a data set. The first command below gives an error message because the **Direction** variable is qualitative.

```
> cor(Smarket)
Error in cor(Smarket) : 'x' must be numeric
> cor(Smarket[, -9])
      Year      Lag1      Lag2      Lag3      Lag4      Lag5
Year    1.0000  0.02970  0.03060  0.03319  0.03569  0.02979
Lag1    0.0297  1.00000 -0.02629 -0.01080 -0.00299 -0.00567
Lag2    0.0306 -0.02629  1.00000 -0.02590 -0.01085 -0.00356
Lag3    0.0332 -0.01080 -0.02590  1.00000 -0.02405 -0.01881
Lag4    0.0357 -0.00299 -0.01085 -0.02405  1.00000 -0.02708
Lag5    0.0298 -0.00567 -0.00356 -0.01881 -0.02708  1.00000
Volume  0.5390  0.04091 -0.04338 -0.04182 -0.04841 -0.02200
Today   0.0301 -0.02616 -0.01025 -0.00245 -0.00690 -0.03486
      Volume      Today
Year    0.5390  0.03010
```

```
Lag1      0.0409 -0.02616
Lag2     -0.0434 -0.01025
Lag3     -0.0418 -0.00245
Lag4     -0.0484 -0.00690
Lag5     -0.0220 -0.03486
Volume    1.0000  0.01459
Today     0.0146  1.00000
```

As one would expect, the correlations between the lag variables and today's returns are close to zero. In other words, there appears to be little correlation between today's returns and previous days' returns. The only substantial correlation is between **Year** and **Volume**. By plotting the data we see that **Volume** is increasing over time. In other words, the average number of shares traded daily increased from 2001 to 2005.

```
> attach(Smarket)
> plot(Volume)
```

4.6.2 Logistic Regression

Next, we will fit a logistic regression model in order to predict **Direction** using **Lag1** through **Lag5** and **Volume**. The `glm()` function fits *generalized linear models*, a class of models that includes logistic regression. The syntax of the `glm()` function is similar to that of `lm()`, except that we must pass in the argument `family=binomial` in order to tell **R** to run a logistic regression rather than some other type of generalized linear model.

`glm()`
generalized
linear model

```
> glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume ,
  data=Smarket ,family=binomial)
> summary(glm.fit)
```

Call:

```
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5
    + Volume, family = binomial, data = Smarket)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.45	-1.20	1.07	1.15	1.33

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-0.12600	0.24074	-0.52	0.60
Lag1	-0.07307	0.05017	-1.46	0.15
Lag2	-0.04230	0.05009	-0.84	0.40
Lag3	0.01109	0.04994	0.22	0.82
Lag4	0.00936	0.04997	0.19	0.85
Lag5	0.01031	0.04951	0.21	0.83
Volume	0.13544	0.15836	0.86	0.39

```
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 1731.2 on 1249 degrees of freedom
Residual deviance: 1727.6 on 1243 degrees of freedom
AIC: 1742

Number of Fisher Scoring iterations: 3
```

The smallest p-value here is associated with **Lag1**. The negative coefficient for this predictor suggests that if the market had a positive return yesterday, then it is less likely to go up today. However, at a value of 0.15, the p-value is still relatively large, and so there is no clear evidence of a real association between **Lag1** and **Direction**.

We use the `coef()` function in order to access just the coefficients for this fitted model. We can also use the `summary()` function to access particular aspects of the fitted model, such as the p-values for the coefficients.

```
> coef(glm.fit)
(Intercept)      Lag1      Lag2      Lag3      Lag4
-0.12600    -0.07307    -0.04230    0.01109    0.00936
      Lag5      Volume
0.01031    0.13544

> summary(glm.fit)$coef
      Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12600    0.2407  -0.523   0.601
Lag1        -0.07307    0.0502  -1.457   0.145
Lag2        -0.04230    0.0501  -0.845   0.398
Lag3         0.01109    0.0499   0.222   0.824
Lag4         0.00936    0.0500   0.187   0.851
Lag5         0.01031    0.0495   0.208   0.835
Volume       0.13544    0.1584   0.855   0.392

> summary(glm.fit)$coef[,4]
(Intercept)      Lag1      Lag2      Lag3      Lag4
0.601        0.145      0.398      0.824      0.851
      Lag5      Volume
0.835        0.392
```

The `predict()` function can be used to predict the probability that the market will go up, given values of the predictors. The `type="response"` option tells **R** to output probabilities of the form $P(Y = 1|X)$, as opposed to other information such as the logit. If no data set is supplied to the `predict()` function, then the probabilities are computed for the training data that was used to fit the logistic regression model. Here we have printed only the first ten probabilities. We know that these values correspond to the probability of the market going up, rather than down, because the `contrasts()` function indicates that **R** has created a dummy variable with a 1 for Up.

```
> glm.probs=predict(glm.fit,type="response")
> glm.probs[1:10]
 1      2      3      4      5      6      7      8      9     10
0.507 0.481 0.481 0.515 0.511 0.507 0.493 0.509 0.518 0.489
```

```
> contrasts(Direction)
      Up
Down  0
Up    1
```

In order to make a prediction as to whether the market will go up or down on a particular day, we must convert these predicted probabilities into class labels, **Up** or **Down**. The following two commands create a vector of class predictions based on whether the predicted probability of a market increase is greater than or less than 0.5.

```
> glm.pred=rep("Down",1250)
> glm.pred[glm.probs>.5]="Up"
```

The first command creates a vector of 1,250 **Down** elements. The second line transforms to **Up** all of the elements for which the predicted probability of a market increase exceeds 0.5. Given these predictions, the **table()** function can be used to produce a confusion matrix in order to determine how many observations were correctly or incorrectly classified.

```
> table(glm.pred,Direction)
      Direction
glm.pred Down  Up
Down    145 141
Up      457 507
> (507+145)/1250
[1] 0.5216
> mean(glm.pred==Direction)
[1] 0.5216
```

The diagonal elements of the confusion matrix indicate correct predictions, while the off-diagonals represent incorrect predictions. Hence our model correctly predicted that the market would go up on 507 days and that it would go down on 145 days, for a total of $507 + 145 = 652$ correct predictions. The **mean()** function can be used to compute the fraction of days for which the prediction was correct. In this case, logistic regression correctly predicted the movement of the market 52.2% of the time.

At first glance, it appears that the logistic regression model is working a little better than random guessing. However, this result is misleading because we trained and tested the model on the same set of 1,250 observations. In other words, $100 - 52.2 = 47.8\%$ is the *training* error rate. As we have seen previously, the training error rate is often overly optimistic—it tends to underestimate the test error rate. In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the *held out* data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

To implement this strategy, we will first create a vector corresponding to the observations from 2001 through 2004. We will then use this vector to create a held out data set of observations from 2005.

```
> train=(Year<2005)
> Smarket.2005=Smarket[!train,]
> dim(Smarket.2005)
[1] 252    9
> Direction.2005=Direction[!train]
```

The object `train` is a vector of 1,250 elements, corresponding to the observations in our data set. The elements of the vector that correspond to observations that occurred before 2005 are set to `TRUE`, whereas those that correspond to observations in 2005 are set to `FALSE`. The object `train` is a *Boolean* vector, since its elements are `TRUE` and `FALSE`. Boolean vectors can be used to obtain a subset of the rows or columns of a matrix. For instance, the command `Smarket[train,]` would pick out a submatrix of the stock market data set, corresponding only to the dates before 2005, since those are the ones for which the elements of `train` are `TRUE`. The `!` symbol can be used to reverse all of the elements of a Boolean vector. That is, `!train` is a vector similar to `train`, except that the elements that are `TRUE` in `train` get swapped to `FALSE` in `!train`, and the elements that are `FALSE` in `train` get swapped to `TRUE` in `!train`. Therefore, `Smarket[!train,]` yields a submatrix of the stock market data containing only the observations for which `train` is `FALSE`—that is, the observations with dates in 2005. The output above indicates that there are 252 such observations.

boolean

We now fit a logistic regression model using only the subset of the observations that correspond to dates before 2005, using the `subset` argument. We then obtain predicted probabilities of the stock market going up for each of the days in our test set—that is, for the days in 2005.

```
> glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
  data=Smarket, family=binomial, subset=train)
> glm.probs=predict(glm.fit, Smarket.2005, type="response")
```

Notice that we have trained and tested our model on two completely separate data sets: training was performed using only the dates before 2005, and testing was performed using only the dates in 2005. Finally, we compute the predictions for 2005 and compare them to the actual movements of the market over that time period.

```
> glm.pred=rep("Down",252)
> glm.pred[glm.probs>.5]="Up"
> table(glm.pred,Direction.2005)
      Direction.2005
glm.pred Down Up
      Down    77 97
      Up    34 44
> mean(glm.pred==Direction.2005)
```

```
[1] 0.48
> mean(glm.pred!=Direction.2005)
[1] 0.52
```

The `!=` notation means *not equal to*, and so the last command computes the test set error rate. The results are rather disappointing: the test error rate is 52%, which is worse than random guessing! Of course this result is not all that surprising, given that one would not generally expect to be able to use previous days' returns to predict future market performance. (After all, if it were possible to do so, then the authors of this book would be out striking it rich rather than writing a statistics textbook.)

We recall that the logistic regression model had very underwhelming p-values associated with all of the predictors, and that the smallest p-value, though not very small, corresponded to `Lag1`. Perhaps by removing the variables that appear not to be helpful in predicting `Direction`, we can obtain a more effective model. After all, using predictors that have no relationship with the response tends to cause a deterioration in the test error rate (since such predictors cause an increase in variance without a corresponding decrease in bias), and so removing such predictors may in turn yield an improvement. Below we have refit the logistic regression using just `Lag1` and `Lag2`, which seemed to have the highest predictive power in the original logistic regression model.

```
> glm.fit=glm(Direction~Lag1+Lag2,data=Smarket,family=binomial,
  subset=train)
> glm.probs=predict(glm.fit,Smarket.2005,type="response")
> glm.pred=rep("Down",252)
> glm.pred[glm.probs>.5]="Up"
> table(glm.pred,Direction.2005)
      Direction.2005
glm.pred Down  Up
   Down    35  35
   Up     76 106
> mean(glm.pred==Direction.2005)
[1] 0.56
> 106/(106+76)
[1] 0.582
```

Now the results appear to be more promising: 56% of the daily movements have been correctly predicted. The confusion matrix suggests that on days when logistic regression predicts that the market will decline, it is only correct 50% of the time. However, on days when it predicts an increase in the market, it has a 58% accuracy rate.

Suppose that we want to predict the returns associated with particular values of `Lag1` and `Lag2`. In particular, we want to predict `Direction` on a day when `Lag1` and `Lag2` equal 1.2 and 1.1, respectively, and on a day when they equal 1.5 and -0.8 . We do this using the `predict()` function.

```
> predict(glm.fit, newdata=data.frame(Lag1=c(1.2, 1.5),
  Lag2=c(1.1, -0.8)), type="response")
      1      2
0.4791 0.4961
```

4.6.3 Linear Discriminant Analysis

Now we will perform LDA on the `Smarket` data. In R, we fit a LDA model using the `lda()` function, which is part of the `MASS` library. Notice that the syntax for the `lda()` function is identical to that of `lm()`, and to that of `glm()` except for the absence of the `family` option. We fit the model using only the observations before 2005.

```
> library(MASS)
> lda.fit=lda(Direction~Lag1+Lag2, data=Smarket, subset=train)
> lda.fit
Call:
lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)

Prior probabilities of groups:
  Down    Up 
0.492 0.508 

Group means:
      Lag1      Lag2 
Down  0.0428  0.0339 
Up    -0.0395 -0.0313 

Coefficients of linear discriminants:
      LD1 
Lag1 -0.642 
Lag2 -0.514 
> plot(lda.fit)
```

The LDA output indicates that $\hat{\pi}_1 = 0.492$ and $\hat{\pi}_2 = 0.508$; in other words, 49.2% of the training observations correspond to days during which the market went down. It also provides the group means; these are the average of each predictor within each class, and are used by LDA as estimates of μ_k . These suggest that there is a tendency for the previous 2 days' returns to be negative on days when the market increases, and a tendency for the previous days' returns to be positive on days when the market declines. The *coefficients of linear discriminants* output provides the linear combination of `Lag1` and `Lag2` that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of $X = x$ in (4.19). If $-0.642 \times \text{Lag1} - 0.514 \times \text{Lag2}$ is large, then the LDA classifier will predict a market increase, and if it is small, then the LDA classifier will predict a market decline. The `plot()` function produces plots of the *linear discriminants*, obtained by computing $-0.642 \times \text{Lag1} - 0.514 \times \text{Lag2}$ for each of the training observations.

The `predict()` function returns a list with three elements. The first element, `class`, contains LDA's predictions about the movement of the market. The second element, `posterior`, is a matrix whose k th column contains the posterior probability that the corresponding observation belongs to the k th class, computed from (4.10). Finally, `x` contains the linear discriminants, described earlier.

```
> lda.pred=predict(lda.fit, Smarket.2005)
> names(lda.pred)
[1] "class"      "posterior"  "x"
```

As we observed in Section 4.5, the LDA and logistic regression predictions are almost identical.

```
> lda.class=lda.pred$class
> table(lda.class,Direction.2005)
      Direction.2005
lda.pred Down  Up
      Down   35  35
      Up    76 106
> mean(lda.class==Direction.2005)
[1] 0.56
```

Applying a 50 % threshold to the posterior probabilities allows us to recreate the predictions contained in `lda.pred$class`.

```
> sum(lda.pred$posterior[,1]>=.5)
[1] 70
> sum(lda.pred$posterior[,1]<.5)
[1] 182
```

Notice that the posterior probability output by the model corresponds to the probability that the market will *decrease*:

```
> lda.pred$posterior[1:20,1]
> lda.class[1:20]
```

If we wanted to use a posterior probability threshold other than 50 % in order to make predictions, then we could easily do so. For instance, suppose that we wish to predict a market decrease only if we are very certain that the market will indeed decrease on that day—say, if the posterior probability is at least 90 %.

```
> sum(lda.pred$posterior[,1]>.9)
[1] 0
```

No days in 2005 meet that threshold! In fact, the greatest posterior probability of decrease in all of 2005 was 52.02 %.

4.6.4 Quadratic Discriminant Analysis

We will now fit a QDA model to the `Smarket` data. QDA is implemented in `R` using the `qda()` function, which is also part of the `MASS` library. The syntax is identical to that of `lda()`. `qda()`

```
> qda.fit=qda(Direction~Lag1+Lag2,data=Smarket ,subset=train)
> qda.fit
Call:
qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)

Prior probabilities of groups:
  Down      Up
0.492 0.508

Group means:
      Lag1      Lag2
Down  0.0428  0.0339
Up    -0.0395 -0.0313
```

The output contains the group means. But it does not contain the coefficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear, function of the predictors. The `predict()` function works in exactly the same fashion as for LDA.

```
> qda.class=predict(qda.fit,Smarket.2005)$class
> table(qda.class,Direction.2005)
      Direction.2005
qda.class Down  Up
      Down   30  20
      Up    81 121
> mean(qda.class==Direction.2005)
[1] 0.599
```

Interestingly, the QDA predictions are accurate almost 60% of the time, even though the 2005 data was not used to fit the model. This level of accuracy is quite impressive for stock market data, which is known to be quite hard to model accurately. This suggests that the quadratic form assumed by QDA may capture the true relationship more accurately than the linear forms assumed by LDA and logistic regression. However, we recommend evaluating this method's performance on a larger test set before betting that this approach will consistently beat the market!

4.6.5 *K-Nearest Neighbors*

We will now perform KNN using the `knn()` function, which is part of the `class` library. This function works rather differently from the other model-fitting functions that we have encountered thus far. Rather than a two-step approach in which we first fit the model and then we use the model to make predictions, `knn()` forms predictions using a single command. The function requires four inputs. `knn()`

1. A matrix containing the predictors associated with the training data, labeled `train.X` below.
2. A matrix containing the predictors associated with the data for which we wish to make predictions, labeled `test.X` below.

3. A vector containing the class labels for the training observations, labeled `train.Direction` below.
4. A value for K , the number of nearest neighbors to be used by the classifier.

We use the `cbind()` function, short for *column bind*, to bind the `Lag1` and `Lag2` variables together into two matrices, one for the training set and the other for the test set. `cbind()`

```
> library(class)
> train.X=cbind(Lag1,Lag2)[train,]
> test.X=cbind(Lag1,Lag2)[!train,]
> train.Direction=Direction[train]
```

Now the `knn()` function can be used to predict the market's movement for the dates in 2005. We set a random seed before we apply `knn()` because if several observations are tied as nearest neighbors, then `R` will randomly break the tie. Therefore, a seed must be set in order to ensure reproducibility of results.

```
> set.seed(1)
> knn.pred=knn(train.X,test.X,train.Direction,k=1)
> table(knn.pred,Direction.2005)
      Direction.2005
knn.pred Down Up
      Down   43 58
      Up    68 83
> (83+43)/252
[1] 0.5
```

The results using $K = 1$ are not very good, since only 50% of the observations are correctly predicted. Of course, it may be that $K = 1$ results in an overly flexible fit to the data. Below, we repeat the analysis using $K = 3$.

```
> knn.pred=knn(train.X,test.X,train.Direction,k=3)
> table(knn.pred,Direction.2005)
      Direction.2005
knn.pred Down Up
      Down   48 54
      Up    63 87
> mean(knn.pred==Direction.2005)
[1] 0.536
```

The results have improved slightly. But increasing K further turns out to provide no further improvements. It appears that for this data, QDA provides the best results of the methods that we have examined so far.

4.6.6 An Application to Caravan Insurance Data

Finally, we will apply the KNN approach to the `Caravan` data set, which is part of the `ISLR` library. This data set includes 85 predictors that measure

demographic characteristics for 5,822 individuals. The response variable is **Purchase**, which indicates whether or not a given individual purchases a caravan insurance policy. In this data set, only 6 % of people purchased caravan insurance.

```
> dim(Caravan)
[1] 5822 86
> attach(Caravan)
> summary(Purchase)
  No   Yes
5474 348
> 348/5822
[1] 0.0598
```

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the *distance* between the observations, and hence on the KNN classifier, than variables that are on a small scale. For instance, imagine a data set that contains two variables, **salary** and **age** (measured in dollars and years, respectively). As far as KNN is concerned, a difference of \$1,000 in salary is enormous compared to a difference of 50 years in age. Consequently, **salary** will drive the KNN classification results, and **age** will have almost no effect. This is contrary to our intuition that a salary difference of \$1,000 is quite small compared to an age difference of 50 years. Furthermore, the importance of scale to the KNN classifier leads to another issue: if we measured **salary** in Japanese yen, or if we measured **age** in minutes, then we'd get quite different classification results from what we get if these two variables are measured in dollars and years.

A good way to handle this problem is to *standardize* the data so that all variables are given a mean of zero and a standard deviation of one. Then all variables will be on a comparable scale. The **scale()** function does just this. In standardizing the data, we exclude column 86, because that is the qualitative **Purchase** variable.

```
> standardized.X=scale(Caravan[, -86])
> var(Caravan[, 1])
[1] 165
> var(Caravan[, 2])
[1] 0.165
> var(standardized.X[, 1])
[1] 1
> var(standardized.X[, 2])
[1] 1
```

Now every column of **standardized.X** has a standard deviation of one and a mean of zero.

We now split the observations into a test set, containing the first 1,000 observations, and a training set, containing the remaining observations.

We fit a KNN model on the training data using $K = 1$, and evaluate its performance on the test data.

```
> test=1:1000
> train.X=standardized.X[-test,]
> test.X=standardized.X[test,]
> train.Y=Purchase[-test]
> test.Y=Purchase[test]
> set.seed(1)
> knn.pred=knn(train.X,test.X,train.Y,k=1)
> mean(test.Y!=knn.pred)
[1] 0.118
> mean(test.Y!="No")
[1] 0.059
```

The vector `test` is numeric, with values from 1 through 1,000. Typing `standardized.X[test,]` yields the submatrix of the data containing the observations whose indices range from 1 to 1,000, whereas typing `standardized.X[-test,]` yields the submatrix containing the observations whose indices do *not* range from 1 to 1,000. The KNN error rate on the 1,000 test observations is just under 12%. At first glance, this may appear to be fairly good. However, since only 6% of customers purchased insurance, we could get the error rate down to 6% by always predicting **No** regardless of the values of the predictors!

Suppose that there is some non-trivial cost to trying to sell insurance to a given individual. For instance, perhaps a salesperson must visit each potential customer. If the company tries to sell insurance to a random selection of customers, then the success rate will be only 6%, which may be far too low given the costs involved. Instead, the company would like to try to sell insurance only to customers who are likely to buy it. So the overall error rate is not of interest. Instead, the fraction of individuals that are correctly predicted to buy insurance is of interest.

It turns out that KNN with $K = 1$ does far better than random guessing among the customers that are predicted to buy insurance. Among 77 such customers, 9, or 11.7%, actually do purchase insurance. This is double the rate that one would obtain from random guessing.

```
> table(knn.pred,test.Y)
      test.Y
knn.pred  No  Yes
      No  873  50
      Yes   68   9
> 9/(68+9)
[1] 0.117
```

Using $K = 3$, the success rate increases to 19%, and with $K = 5$ the rate is 26.7%. This is over four times the rate that results from random guessing. It appears that KNN is finding some real patterns in a difficult data set!


```

> knn.pred=knn(train.X,test.X,train.Y,k=3)
> table(knn.pred,test.Y)
      test.Y
knn.pred  No  Yes
      No  920  54
      Yes   21   5
> 5/26
[1] 0.192
> knn.pred=knn(train.X,test.X,train.Y,k=5)
> table(knn.pred,test.Y)
      test.Y
knn.pred  No  Yes
      No  930  55
      Yes   11   4
> 4/15
[1] 0.267

```

As a comparison, we can also fit a logistic regression model to the data. If we use 0.5 as the predicted probability cut-off for the classifier, then we have a problem: only seven of the test observations are predicted to purchase insurance. Even worse, we are wrong about all of these! However, we are not required to use a cut-off of 0.5. If we instead predict a purchase any time the predicted probability of purchase exceeds 0.25, we get much better results: we predict that 33 people will purchase insurance, and we are correct for about 33% of these people. This is over five times better than random guessing!

```

> glm.fit=glm(Purchase~.,data=Caravan,family=binomial,
  subset=-test)
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> glm.probs=predict(glm.fit,Caravan[test,],type="response")
> glm.pred=rep("No",1000)
> glm.pred[glm.probs>.5]="Yes"
> table(glm.pred,test.Y)
      test.Y
glm.pred  No  Yes
      No  934  59
      Yes   7   0
> glm.pred=rep("No",1000)
> glm.pred[glm.probs>.25]="Yes"
> table(glm.pred,test.Y)
      test.Y
glm.pred  No  Yes
      No  919  48
      Yes   22  11
> 11/(22+11)
[1] 0.333

```

4.7 Exercises

Conceptual

1. Using a little bit of algebra, prove that (4.2) is equivalent to (4.3). In other words, the logistic function representation and logit representation for the logistic regression model are equivalent.
2. It was stated in the text that classifying an observation to the class for which (4.12) is largest is equivalent to classifying an observation to the class for which (4.13) is largest. Prove that this is the case. In other words, under the assumption that the observations in the k th class are drawn from a $N(\mu_k, \sigma^2)$ distribution, the Bayes' classifier assigns an observation to the class for which the discriminant function is maximized.
3. This problem relates to the QDA model, in which the observations within each class are drawn from a normal distribution with a class-specific mean vector and a class specific covariance matrix. We consider the simple case where $p = 1$; i.e. there is only one feature.

Suppose that we have K classes, and that if an observation belongs to the k th class then X comes from a one-dimensional normal distribution, $X \sim N(\mu_k, \sigma_k^2)$. Recall that the density function for the one-dimensional normal distribution is given in (4.11). Prove that in this case, the Bayes' classifier is *not* linear. Argue that it is in fact quadratic.

Hint: For this problem, you should follow the arguments laid out in Section 4.4.2, but without making the assumption that $\sigma_1^2 = \dots = \sigma_K^2$.

4. When the number of features p is large, there tends to be a deterioration in the performance of KNN and other *local* approaches that perform prediction using only observations that are *near* the test observation for which a prediction must be made. This phenomenon is known as the *curse of dimensionality*, and it ties into the fact that non-parametric approaches often perform poorly when p is large. We will now investigate this curse.



curse of dimensionality

- (a) Suppose that we have a set of observations, each with measurements on $p = 1$ feature, X . We assume that X is uniformly (evenly) distributed on $[0, 1]$. Associated with each observation is a response value. Suppose that we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with $X = 0.6$,

we will use observations in the range $[0.55, 0.65]$. On average, what fraction of the available observations will we use to make the prediction?

- (b) Now suppose that we have a set of observations, each with measurements on $p = 2$ features, X_1 and X_2 . We assume that (X_1, X_2) are uniformly distributed on $[0, 1] \times [0, 1]$. We wish to predict a test observation's response using only observations that are within 10 % of the range of X_1 and within 10 % of the range of X_2 closest to that test observation. For instance, in order to predict the response for a test observation with $X_1 = 0.6$ and $X_2 = 0.35$, we will use observations in the range $[0.55, 0.65]$ for X_1 and in the range $[0.3, 0.4]$ for X_2 . On average, what fraction of the available observations will we use to make the prediction?
- (c) Now suppose that we have a set of observations on $p = 100$ features. Again the observations are uniformly distributed on each feature, and again each feature ranges in value from 0 to 1. We wish to predict a test observation's response using observations within the 10 % of each feature's range that is closest to that test observation. What fraction of the available observations will we use to make the prediction?
- (d) Using your answers to parts (a)–(c), argue that a drawback of KNN when p is large is that there are very few training observations “near” any given test observation.
- (e) Now suppose that we wish to make a prediction for a test observation by creating a p -dimensional hypercube centered around the test observation that contains, on average, 10 % of the training observations. For $p = 1, 2$, and 100, what is the length of each side of the hypercube? Comment on your answer.

Note: A hypercube is a generalization of a cube to an arbitrary number of dimensions. When $p = 1$, a hypercube is simply a line segment, when $p = 2$ it is a square, and when $p = 100$ it is a 100-dimensional cube.

5. We now examine the differences between LDA and QDA.

- (a) If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?
- (b) If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?
- (c) In general, as the sample size n increases, do we expect the test prediction accuracy of QDA relative to LDA to improve, decline, or be unchanged? Why?

- (d) True or False: Even if the Bayes decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible enough to model a linear decision boundary. Justify your answer.
6. Suppose we collect data for a group of students in a statistics class with variables X_1 = hours studied, X_2 = undergrad GPA, and Y = receive an A. We fit a logistic regression and produce estimated coefficient, $\hat{\beta}_0 = -6$, $\hat{\beta}_1 = 0.05$, $\hat{\beta}_2 = 1$.
- (a) Estimate the probability that a student who studies for 40 h and has an undergrad GPA of 3.5 gets an A in the class.
- (b) How many hours would the student in part (a) need to study to have a 50 % chance of getting an A in the class?
7. Suppose that we wish to predict whether a given stock will issue a dividend this year (“Yes” or “No”) based on X , last year’s percent profit. We examine a large number of companies and discover that the mean value of X for companies that issued a dividend was $\bar{X} = 10$, while the mean for those that didn’t was $\bar{X} = 0$. In addition, the variance of X for these two sets of companies was $\hat{\sigma}^2 = 36$. Finally, 80 % of companies issued dividends. Assuming that X follows a normal distribution, predict the probability that a company will issue a dividend this year given that its percentage profit was $X = 4$ last year.
- Hint: Recall that the density function for a normal random variable is $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$. You will need to use Bayes’ theorem.*
8. Suppose that we take a data set, divide it into equally-sized training and test sets, and then try out two different classification procedures. First we use logistic regression and get an error rate of 20 % on the training data and 30 % on the test data. Next we use 1-nearest neighbors (i.e. $K = 1$) and get an average error rate (averaged over both test and training data sets) of 18 %. Based on these results, which method should we prefer to use for classification of new observations? Why?
9. This problem has to do with *odds*.
- (a) On average, what fraction of people with an odds of 0.37 of defaulting on their credit card payment will in fact default?
- (b) Suppose that an individual has a 16 % chance of defaulting on her credit card payment. What are the odds that she will default?

Applied

10. This question should be answered using the **Weekly** data set, which is part of the **ISLR** package. This data is similar in nature to the **Smarket** data from this chapter's lab, except that it contains 1,089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.
 - (a) Produce some numerical and graphical summaries of the **Weekly** data. Do there appear to be any patterns?
 - (b) Use the full data set to perform a logistic regression with **Direction** as the response and the five lag variables plus **Volume** as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?
 - (c) Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.
 - (d) Now fit the logistic regression model using a training data period from 1990 to 2008, with **Lag2** as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).
 - (e) Repeat (d) using LDA.
 - (f) Repeat (d) using QDA.
 - (g) Repeat (d) using KNN with $K = 1$.
 - (h) Which of these methods appears to provide the best results on this data?
 - (i) Experiment with different combinations of predictors, including possible transformations and interactions, for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the held out data. Note that you should also experiment with values for K in the KNN classifier.
11. In this problem, you will develop a model to predict whether a given car gets high or low gas mileage based on the **Auto** data set.
 - (a) Create a binary variable, **mpg01**, that contains a 1 if **mpg** contains a value above its median, and a 0 if **mpg** contains a value below its median. You can compute the median using the **median()** function. Note you may find it helpful to use the **data.frame()** function to create a single data set containing both **mpg01** and the other **Auto** variables.

- (b) Explore the data graphically in order to investigate the association between `mpg01` and the other features. Which of the other features seem most likely to be useful in predicting `mpg01`? Scatterplots and boxplots may be useful tools to answer this question. Describe your findings.
- (c) Split the data into a training set and a test set.
- (d) Perform LDA on the training data in order to predict `mpg01` using the variables that seemed most associated with `mpg01` in (b). What is the test error of the model obtained?
- (e) Perform QDA on the training data in order to predict `mpg01` using the variables that seemed most associated with `mpg01` in (b). What is the test error of the model obtained?
- (f) Perform logistic regression on the training data in order to predict `mpg01` using the variables that seemed most associated with `mpg01` in (b). What is the test error of the model obtained?
- (g) Perform KNN on the training data, with several values of K , in order to predict `mpg01`. Use only the variables that seemed most associated with `mpg01` in (b). What test errors do you obtain? Which value of K seems to perform the best on this data set?

12. This problem involves writing functions.

- (a) Write a function, `Power()`, that prints out the result of raising 2 to the 3rd power. In other words, your function should compute 2^3 and print out the results.

Hint: Recall that `x^a` raises `x` to the power `a`. Use the `print()` function to output the result.

- (b) Create a new function, `Power2()`, that allows you to pass *any* two numbers, `x` and `a`, and prints out the value of `x^a`. You can do this by beginning your function with the line

```
> Power2=function(x,a){
```

You should be able to call your function by entering, for instance,

```
> Power2(3,8)
```

on the command line. This should output the value of 3^8 , namely, 6,561.

- (c) Using the `Power2()` function that you just wrote, compute 10^3 , 8^{17} , and 131^3 .
- (d) Now create a new function, `Power3()`, that actually *returns* the result `x^a` as an R object, rather than simply printing it to the screen. That is, if you store the value `x^a` in an object called `result` within your function, then you can simply `return()` this result, using the following line:

```
return()
```

```
return(result)
```

The line above should be the last line in your function, before the `}` symbol.

- (e) Now using the `Power3()` function, create a plot of $f(x) = x^2$. The x -axis should display a range of integers from 1 to 10, and the y -axis should display x^2 . Label the axes appropriately, and use an appropriate title for the figure. Consider displaying either the x -axis, the y -axis, or both on the log-scale. You can do this by using `log='x'`, `log='y'`, or `log='xy'` as arguments to the `plot()` function.
- (f) Create a function, `PlotPower()`, that allows you to create a plot of x against x^a for a fixed a and for a range of values of x . For instance, if you call

```
> PlotPower(1:10,3)
```

then a plot should be created with an x -axis taking on values 1, 2, ..., 10, and a y -axis taking on values $1^3, 2^3, \dots, 10^3$.

13. Using the `Boston` data set, fit classification models in order to predict whether a given suburb has a crime rate above or below the median. Explore logistic regression, LDA, and KNN models using various subsets of the predictors. Describe your findings.