LSE Machine Learning: Practical Applications
Reading: An Introduction to Statistical Learning

Table of contents

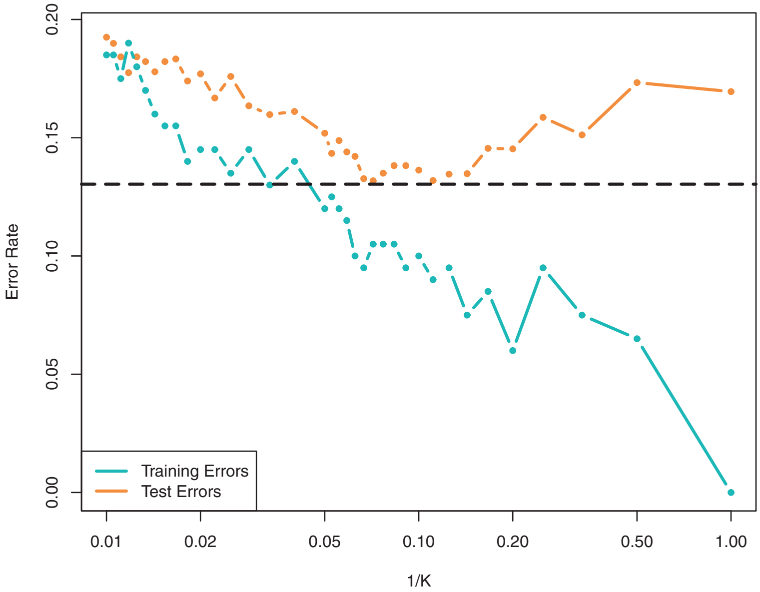

FIGURE 2.17. *The KNN training error rate (blue, 200 observations) and test error rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of flexibility (assessed using $1/K$) increases, or equivalently as the number of neighbors K decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.*

In both the regression and classification settings, choosing the correct level of flexibility is critical to the success of any statistical learning method. The bias-variance tradeoff, and the resulting U-shape in the test error, can make this a difficult task. In Chapter 5, we return to this topic and discuss various methods for estimating test error rates and thereby choosing the optimal level of flexibility for a given statistical learning method.

2.3 Lab: Introduction to R

In this lab, we will introduce some simple `R` commands. The best way to learn a new language is to try out the commands. `R` can be downloaded from

`http://cran.r-project.org/`

2.3.1 Basic Commands

`R` uses *functions* to perform operations. To run a function called `funcname`, we type `funcname(input1, input2)`, where the inputs (or *arguments*) `input1`

function
argument

and `input2` tell `R` how to run the function. A function can have any number of inputs. For example, to create a vector of numbers, we use the function `c()` (for *concatenate*). Any numbers inside the parentheses are joined together. The following command instructs `R` to join together the numbers 1, 3, 2, and 5, and to save them as a *vector* named `x`. When we type `x`, it gives us back the vector.

`c()`

vector

```
> x <- c(1,3,2,5)
> x
[1] 1 3 2 5
```

Note that the `>` is not part of the command; rather, it is printed by `R` to indicate that it is ready for another command to be entered. We can also save things using `=` rather than `<-`:

```
> x = c(1,6,2)
> x
[1] 1 6 2
> y = c(1,4,3)
```

Hitting the *up* arrow multiple times will display the previous commands, which can then be edited. This is useful since one often wishes to repeat a similar command. In addition, typing `?funcname` will always cause `R` to open a new help file window with additional information about the function `funcname`.

We can tell `R` to add two sets of numbers together. It will then add the first number from `x` to the first number from `y`, and so on. However, `x` and `y` should be the same length. We can check their length using the `length()` function.

`length()`

```
> length(x)
[1] 3
> length(y)
[1] 3
> x+y
[1]  2 10  5
```

The `ls()` function allows us to look at a list of all of the objects, such as data and functions, that we have saved so far. The `rm()` function can be used to delete any that we don't want.

`ls()`
`rm()`

```
> ls()
[1] "x" "y"
> rm(x,y)
> ls()
character(0)
```

It's also possible to remove all objects at once:

```
> rm(list=ls())
```

The `matrix()` function can be used to create a matrix of numbers. Before we use the `matrix()` function, we can learn more about it: `matrix()`

```
> ?matrix
```

The help file reveals that the `matrix()` function takes a number of inputs, but for now we focus on the first three: the data (the entries in the matrix), the number of rows, and the number of columns. First, we create a simple matrix.

```
> x=matrix(data=c(1,2,3,4), nrow=2, ncol=2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Note that we could just as well omit typing `data=`, `nrow=`, and `ncol=` in the `matrix()` command above: that is, we could just type

```
> x=matrix(c(1,2,3,4),2,2)
```

and this would have the same effect. However, it can sometimes be useful to specify the names of the arguments passed in, since otherwise `R` will assume that the function arguments are passed into the function in the same order that is given in the function's help file. As this example illustrates, by default `R` creates matrices by successively filling in columns. Alternatively, the `byrow=TRUE` option can be used to populate the matrix in order of the rows.

```
> matrix(c(1,2,3,4),2,2,byrow=TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Notice that in the above command we did not assign the matrix to a value such as `x`. In this case the matrix is printed to the screen but is not saved for future calculations. The `sqrt()` function returns the square root of each element of a vector or matrix. The command `x^2` raises each element of `x` to the power `2`; any powers are possible, including fractional or negative powers. `sqrt()`

```
> sqrt(x)
     [,1] [,2]
[1,] 1.00 1.73
[2,] 1.41 2.00
> x^2
     [,1] [,2]
[1,]    1    9
[2,]    4   16
```

The `rnorm()` function generates a vector of random normal variables, with first argument `n` the sample size. Each time we call this function, we will get a different answer. Here we create two correlated sets of numbers, `x` and `y`, and use the `cor()` function to compute the correlation between them. `rnorm()` `cor()`

```
> x=rnorm(50)
> y=x+rnorm(50,mean=50,sd=.1)
> cor(x,y)
[1] 0.995
```

By default, `rnorm()` creates standard normal random variables with a mean of 0 and a standard deviation of 1. However, the mean and standard deviation can be altered using the `mean` and `sd` arguments, as illustrated above. Sometimes we want our code to reproduce the exact same set of random numbers; we can use the `set.seed()` function to do this. The `set.seed()` function takes an (arbitrary) integer argument. `set.seed()`

```
> set.seed(1303)
> rnorm(50)
[1] -1.1440  1.3421  2.1854  0.5364  0.0632  0.5022 -0.0004
. . .
```

We use `set.seed()` throughout the labs whenever we perform calculations involving random quantities. In general this should allow the user to reproduce our results. However, it should be noted that as new versions of `R` become available it is possible that some small discrepancies may form between the book and the output from `R`.

The `mean()` and `var()` functions can be used to compute the mean and variance of a vector of numbers. Applying `sqrt()` to the output of `var()` will give the standard deviation. Or we can simply use the `sd()` function. `mean()` `var()` `sd()`

```
> set.seed(3)
> y=rnorm(100)
> mean(y)
[1] 0.0110
> var(y)
[1] 0.7329
> sqrt(var(y))
[1] 0.8561
> sd(y)
[1] 0.8561
```

2.3.2 Graphics

The `plot()` function is the primary way to plot data in `R`. For instance, `plot(x,y)` produces a scatterplot of the numbers in `x` versus the numbers in `y`. There are many additional options that can be passed in to the `plot()` function. For example, passing in the argument `xlab` will result in a label on the x-axis. To find out more information about the `plot()` function, type `?plot`. `plot()`

```
> x=rnorm(100)
> y=rnorm(100)
> plot(x,y)
> plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis",
      main="Plot of X vs Y")
```

We will often want to save the output of an `R` plot. The command that we use to do this will depend on the file type that we would like to create. For instance, to create a pdf, we use the `pdf()` function, and to create a jpeg, we use the `jpeg()` function.

pdf()

jpeg()

```
> pdf("Figure.pdf")
> plot(x,y,col="green")
> dev.off()
null device
          1
```

The function `dev.off()` indicates to `R` that we are done creating the plot. Alternatively, we can simply copy the plot window and paste it into an appropriate file type, such as a Word document.

dev.off()

The function `seq()` can be used to create a sequence of numbers. For instance, `seq(a,b)` makes a vector of integers between `a` and `b`. There are many other options: for instance, `seq(0,1,length=10)` makes a sequence of `10` numbers that are equally spaced between `0` and `1`. Typing `3:11` is a shorthand for `seq(3,11)` for integer arguments.

seq()

```
> x=seq(1,10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x=1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x=seq(-pi,pi,length=50)
```

We will now create some more sophisticated plots. The `contour()` function produces a *contour plot* in order to represent three-dimensional data; it is like a topographical map. It takes three arguments:

contour()

contour plot

1. A vector of the `x` values (the first dimension),
2. A vector of the `y` values (the second dimension), and
3. A matrix whose elements correspond to the `z` value (the third dimension) for each pair of (`x`,`y`) coordinates.

As with the `plot()` function, there are many other inputs that can be used to fine-tune the output of the `contour()` function. To learn more about these, take a look at the help file by typing `?contour`.

```
> y=x
> f=outer(x,y,function(x,y)cos(y)/(1+x^2))
> contour(x,y,f)
> contour(x,y,f,nlevels=45,add=T)
> fa=(f-t(f))/2
> contour(x,y,fa,nlevels=15)
```

The `image()` function works the same way as `contour()`, except that it produces a color-coded plot whose colors depend on the `z` value. This is

image()

known as a *heatmap*, and is sometimes used to plot temperature in weather forecasts. Alternatively, `persp()` can be used to produce a three-dimensional plot. The arguments `theta` and `phi` control the angles at which the plot is viewed.

heatmap

persp()

```
> image(x,y,fa)
> persp(x,y,fa)
> persp(x,y,fa,theta=30)
> persp(x,y,fa,theta=30,phi=20)
> persp(x,y,fa,theta=30,phi=70)
> persp(x,y,fa,theta=30,phi=40)
```

2.3.3 Indexing Data

We often wish to examine part of a set of data. Suppose that our data is stored in the matrix `A`.

```
> A=matrix(1:16,4,4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Then, typing

```
> A[2,3]
[1] 10
```

will select the element corresponding to the second row and the third column. The first number after the open-bracket symbol `[` always refers to the row, and the second number always refers to the column. We can also select multiple rows and columns at a time, by providing vectors as the indices.

```
> A[c(1,3),c(2,4)]
     [,1] [,2]
[1,]    5   13
[2,]    7   15
> A[1:3,2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
> A[1:2,]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
> A[,1:2]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
```

```
[3,]    3    7
[4,]    4    8
```

The last two examples include either no index for the columns or no index for the rows. These indicate that `R` should include all columns or all rows, respectively. `R` treats a single row or column of a matrix as a vector.

```
> A[1,]
[1]  1  5  9 13
```

The use of a negative sign `-` in the index tells `R` to keep all rows or columns except those indicated in the index.

```
> A[-c(1,3),]
     [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    4    8   12   16
> A[-c(1,3),-c(1,3,4)]
[1] 6 8
```

The `dim()` function outputs the number of rows followed by the number of columns of a given matrix. `dim()`

```
> dim(A)
[1] 4 4
```

2.3.4 Loading Data

For most analyses, the first step involves importing a data set into `R`. The `read.table()` function is one of the primary ways to do this. The help file contains details about how to use this function. We can use the function `write.table()` to export data. `read.table()` `write.table()`

Before attempting to load a data set, we must make sure that `R` knows to search for the data in the proper directory. For example on a Windows system one could select the directory using the `Change dir...` option under the `File` menu. However, the details of how to do this depend on the operating system (e.g. Windows, Mac, Unix) that is being used, and so we do not give further details here. We begin by loading in the `Auto` data set. This data is part of the `ISLR` library (we discuss libraries in Chapter 3) but to illustrate the `read.table()` function we load it now from a text file. The following command will load the `Auto.data` file into `R` and store it as an object called `Auto`, in a format referred to as a *data frame*. (The text file can be obtained from this book's website.) Once the data has been loaded, the `fix()` function can be used to view it in a spreadsheet like window. However, the window must be closed before further `R` commands can be entered. data frame

```
> Auto=read.table("Auto.data")
> fix(Auto)
```

Note that Auto.data is simply a text file, which you could alternatively open on your computer using a standard text editor. It is often a good idea to view a data set using a text editor or other software such as Excel before loading it into `R`.

This particular data set has not been loaded correctly, because `R` has assumed that the variable names are part of the data and so has included them in the first row. The data set also includes a number of missing observations, indicated by a question mark `?`. Missing values are a common occurrence in real data sets. Using the option `header=T` (or `header=TRUE`) in the `read.table()` function tells `R` that the first line of the file contains the variable names, and using the option `na.strings` tells `R` that any time it sees a particular character or set of characters (such as a question mark), it should be treated as a missing element of the data matrix.

```
> Auto=read.table("Auto.data",header=T,na.strings="?")
> fix(Auto)
```

Excel is a common-format data storage program. An easy way to load such data into `R` is to save it as a csv (comma separated value) file and then use the `read.csv()` function to load it in.

```
> Auto=read.csv("Auto.csv",header=T,na.strings="?")
> fix(Auto)
> dim(Auto)
[1] 397  9
> Auto[1:4,]
```

The `dim()` function tells us that the data has 397 observations, or rows, and nine variables, or columns. There are various ways to deal with the missing data. In this case, only five of the rows contain missing observations, and so we choose to use the `na.omit()` function to simply remove these rows.

`dim()`

`na.omit()`

```
> Auto=na.omit(Auto)
> dim(Auto)
[1] 392   9
```

Once the data are loaded correctly, we can use `names()` to check the variable names.

`names()`

```
> names(Auto)
[1] "mpg"          "cylinders"    "displacement" "horsepower"
[5] "weight"       "acceleration" "year"         "origin"
[9] "name"
```

2.3.5 Additional Graphical and Numerical Summaries

We can use the `plot()` function to produce *scatterplots* of the quantitative variables. However, simply typing the variable names will produce an error message, because `R` does not know to look in the `Auto` data set for those variables.

scatterplot

```
> plot(cylinders, mpg)
Error in plot(cylinders, mpg) : object 'cylinders' not found
```

To refer to a variable, we must type the data set and the variable name joined with a `$` symbol. Alternatively, we can use the `attach()` function in order to tell `R` to make the variables in this data frame available by name.

attach()

```
> plot(Auto$cylinders, Auto$mpg)
> attach(Auto)
> plot(cylinders, mpg)
```

The `cylinders` variable is stored as a numeric vector, so `R` has treated it as quantitative. However, since there are only a small number of possible values for `cylinders`, one may prefer to treat it as a qualitative variable. The `as.factor()` function converts quantitative variables into qualitative variables.

as.factor()

```
> cylinders=as.factor(cylinders)
```

If the variable plotted on the x-axis is categorial, then *boxplots* will automatically be produced by the `plot()` function. As usual, a number of options can be specified in order to customize the plots.

boxplot

```
> plot(cylinders, mpg)
> plot(cylinders, mpg, col="red")
> plot(cylinders, mpg, col="red", varwidth=T)
> plot(cylinders, mpg, col="red", varwidth=T,horizontal=T)
> plot(cylinders, mpg, col="red", varwidth=T, xlab="cylinders",
    ylab="MPG")
```

The `hist()` function can be used to plot a *histogram*. Note that `col=2` has the same effect as `col="red"`.

hist()
histogram

```
> hist(mpg)
> hist(mpg,col=2)
> hist(mpg,col=2,breaks=15)
```

The `pairs()` function creates a *scatterplot matrix* i.e. a scatterplot for every pair of variables for any given data set. We can also produce scatterplots for just a subset of the variables.

scatterplot matrix

```
> pairs(Auto)
> pairs(~ mpg + displacement + horsepower + weight +
   acceleration, Auto)
```

In conjunction with the `plot()` function, `identify()` provides a useful interactive method for identifying the value for a particular variable for points on a plot. We pass in three arguments to `identify()`: the x-axis variable, the y-axis variable, and the variable whose values we would like to see printed for each point. Then clicking on a given point in the plot will cause `R` to print the value of the variable of interest. Right-clicking on the plot will exit the `identify()` function (control-click on a Mac). The numbers printed under the `identify()` function correspond to the rows for the selected points.

identify()

```
> plot(horsepower ,mpg)
> identify(horsepower ,mpg ,name)
```

The `summary()` function produces a numerical summary of each variable in a particular data set. `summary()`

```
> summary(Auto)
      mpg          cylinders      displacement
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0
 1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0
 Median :22.75   Median :4.000   Median :151.0
 Mean   :23.45   Mean   :5.472   Mean   :194.4
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8
 Max.   :46.60   Max.   :8.000   Max.   :455.0

   horsepower        weight      acceleration
 Min.   : 46.0   Min.   :1613   Min.   : 8.00
 1st Qu.: 75.0   1st Qu.:2225   1st Qu.:13.78
 Median : 93.5   Median :2804   Median :15.50
 Mean   :104.5   Mean   :2978   Mean   :15.54
 3rd Qu.:126.0   3rd Qu.:3615   3rd Qu.:17.02
 Max.   :230.0   Max.   :5140   Max.   :24.80

      year           origin                      name
 Min.   :70.00   Min.   :1.000   amc matador       :  5
 1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
 Median :76.00   Median :1.000   toyota corolla    :  5
 Mean   :75.98   Mean   :1.577   amc gremlin       :  4
 3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
 Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
                                 (Other)           :365
```

For qualitative variables such as `name`, `R` will list the number of observations that fall in each category. We can also produce a summary of just a single variable.

```
> summary(mpg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.00   17.00   22.75   23.45   29.00   46.60
```

Once we have finished using `R`, we type `q()` in order to shut it down, or quit. When exiting `R`, we have the option to save the current *workspace* so that all objects (such as data sets) that we have created in this `R` session will be available next time. Before exiting `R`, we may want to save a record of all of the commands that we typed in the most recent session; this can be accomplished using the `savehistory()` function. Next time we enter `R`, we can load that history using the `loadhistory()` function. `q()` workspace `savehistory()` `loadhistory()`

3

Linear Regression

This chapter is about *linear regression*, a very simple approach for supervised learning. In particular, linear regression is a useful tool for predicting a quantitative response. Linear regression has been around for a long time and is the topic of innumerable textbooks. Though it may seem somewhat dull compared to some of the more modern statistical learning approaches described in later chapters of this book, linear regression is still a useful and widely used statistical learning method. Moreover, it serves as a good jumping-off point for newer approaches: as we will see in later chapters, many fancy statistical learning approaches can be seen as generalizations or extensions of linear regression. Consequently, the importance of having a good understanding of linear regression before studying more complex learning methods cannot be overstated. In this chapter, we review some of the key ideas underlying the linear regression model, as well as the least squares approach that is most commonly used to fit this model.

Recall the `Advertising` data from Chapter 2. Figure 2.1 displays `sales` (in thousands of units) for a particular product as a function of advertising budgets (in thousands of dollars) for `TV`, `radio`, and `newspaper` media. Suppose that in our role as statistical consultants we are asked to suggest, on the basis of this data, a marketing plan for next year that will result in high product sales. What information would be useful in order to provide such a recommendation? Here are a few important questions that we might seek to address:

1. *Is there a relationship between advertising budget and sales?*
   Our first goal should be to determine whether the data provide

G. James et al., *An Introduction to Statistical Learning: with Applications in R*, Springer Texts in Statistics 103, DOI 10.1007/978-1-4614-7138-7_3,

evidence of an association between advertising expenditure and sales. If the evidence is weak, then one might argue that no money should be spent on advertising!

2. *How strong is the relationship between advertising budget and sales?* Assuming that there is a relationship between advertising and sales, we would like to know the strength of this relationship. In other words, given a certain advertising budget, can we predict sales with a high level of accuracy? This would be a strong relationship. Or is a prediction of sales based on advertising expenditure only slightly better than a random guess? This would be a weak relationship.

3. *Which media contribute to sales?* Do all three media—TV, radio, and newspaper—contribute to sales, or do just one or two of the media contribute? To answer this question, we must find a way to separate out the individual effects of each medium when we have spent money on all three media.

4. *How accurately can we estimate the effect of each medium on sales?* For every dollar spent on advertising in a particular medium, by what amount will sales increase? How accurately can we predict this amount of increase?

5. *How accurately can we predict future sales?* For any given level of television, radio, or newspaper advertising, what is our prediction for sales, and what is the accuracy of this prediction?

6. *Is the relationship linear?* If there is approximately a straight-line relationship between advertising expenditure in the various media and sales, then linear regression is an appropriate tool. If not, then it may still be possible to transform the predictor or the response so that linear regression can be used.

7. *Is there synergy among the advertising media?* Perhaps spending \$50,000 on television advertising and \$50,000 on radio advertising results in more sales than allocating \$100,000 to either television or radio individually. In marketing, this is known as a *synergy* effect, while in statistics it is called an *interaction* effect. synergy interaction

It turns out that linear regression can be used to answer each of these questions. We will first discuss all of these questions in a general context, and then return to them in this specific context in Section 3.4.

3.1 Simple Linear Regression

Simple linear regression lives up to its name: it is a very straightforward approach for predicting a quantitative response Y on the basis of a single predictor variable X. It assumes that there is approximately a linear relationship between X and Y. Mathematically, we can write this linear relationship as

simple linear regression

$$Y \approx \beta_0 + \beta_1 X. \tag{3.1}$$

You might read "$\approx$" as *"is approximately modeled as"*. We will sometimes describe (3.1) by saying that we are *regressing* Y *on* X (or Y *onto* X). For example, X may represent `TV` advertising and Y may represent `sales`. Then we can regress `sales` onto `TV` by fitting the model

$$\texttt{sales} \approx \beta_0 + \beta_1 \times \texttt{TV}.$$

In Equation 3.1, β_0 and β_1 are two unknown constants that represent the *intercept* and *slope* terms in the linear model. Together, β_0 and β_1 are known as the model *coefficients* or *parameters*. Once we have used our training data to produce estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ for the model coefficients, we can predict future sales on the basis of a particular value of TV advertising by computing

intercept
slope
coefficient
parameter

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x, \tag{3.2}$$

where $\hat{y}$ indicates a prediction of Y on the basis of $X = x$. Here we use a *hat* symbol, $\hat{}$, to denote the estimated value for an unknown parameter or coefficient, or to denote the predicted value of the response.

3.1.1 Estimating the Coefficients

In practice, β_0 and β_1 are unknown. So before we can use (3.1) to make predictions, we must use data to estimate the coefficients. Let

$$(x_1, y_1),\ (x_2, y_2), \ldots,\ (x_n, y_n)$$

represent n observation pairs, each of which consists of a measurement of X and a measurement of Y. In the `Advertising` example, this data set consists of the TV advertising budget and product sales in $n = 200$ different markets. (Recall that the data are displayed in Figure 2.1.) Our goal is to obtain coefficient estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ such that the linear model (3.1) fits the available data well—that is, so that $y_i \approx \hat{\beta}_0 + \hat{\beta}_1 x_i$ for $i = 1, \ldots, n$. In other words, we want to find an intercept $\hat{\beta}_0$ and a slope $\hat{\beta}_1$ such that the resulting line is as close as possible to the $n = 200$ data points. There are a number of ways of measuring *closeness*. However, by far the most common approach involves minimizing the *least squares* criterion, and we take that approach in this chapter. Alternative approaches will be considered in Chapter 6.

least squares

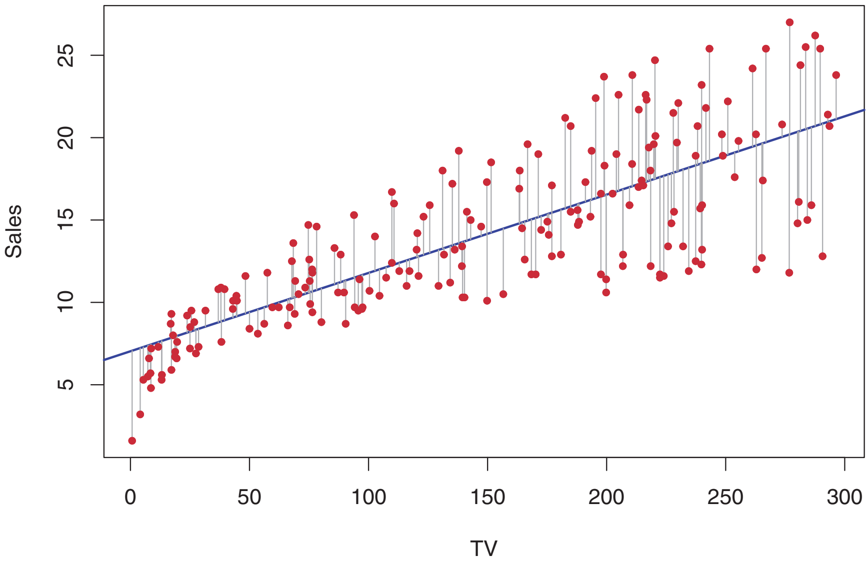

FIGURE 3.1. *For the* `Advertising` *data, the least squares fit for the regression of* `sales` *onto* `TV` *is shown. The fit is found by minimizing the sum of squared errors. Each grey line segment represents an error, and the fit makes a compromise by averaging their squares. In this case a linear fit captures the essence of the relationship, although it is somewhat deficient in the left of the plot.*

Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y}_i$ represents the ith *residual*—this is the difference between the ith observed response value and the ith response value that is predicted by our linear model. We define the *residual sum of squares* (RSS) as

residual

residual sum of squares

$$\mathrm{RSS} = e_1^2 + e_2^2 + \cdots + e_n^2,$$

or equivalently as

$$\mathrm{RSS} = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \ldots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2. \quad (3.3)$$

The least squares approach chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimize the RSS. Using some calculus, one can show that the minimizers are

$$\begin{aligned} \hat{\beta}_1 &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}, \end{aligned} \quad (3.4)$$

where $\bar{y} \equiv \frac{1}{n} \sum_{i=1}^n y_i$ and $\bar{x} \equiv \frac{1}{n} \sum_{i=1}^n x_i$ are the sample means. In other words, (3.4) defines the *least squares coefficient estimates* for simple linear regression.

Figure 3.1 displays the simple linear regression fit to the `Advertising` data, where $\hat{\beta}_0 = 7.03$ and $\hat{\beta}_1 = 0.0475$. In other words, according to

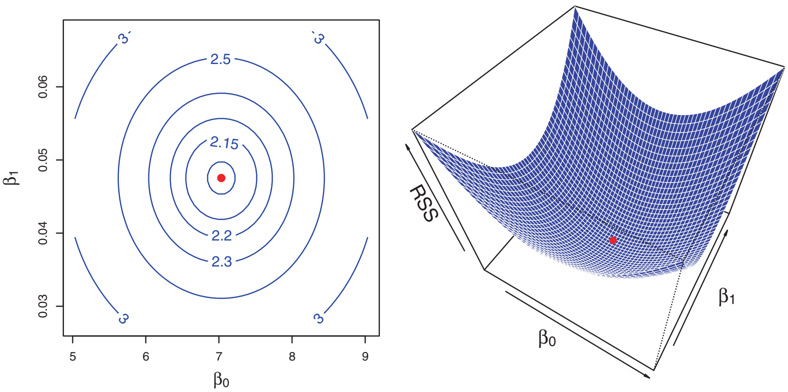

FIGURE 3.2. *Contour and three-dimensional plots of the RSS on the* `Advertising` *data, using* `sales` *as the response and* `TV` *as the predictor. The red dots correspond to the least squares estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, given by (3.4).*

this approximation, an additional $1,000 spent on TV advertising is associated with selling approximately 47.5 additional units of the product. In Figure 3.2, we have computed RSS for a number of values of β_0 and β_1, using the advertising data with `sales` as the response and `TV` as the predictor. In each plot, the red dot represents the pair of least squares estimates $(\hat{\beta}_0, \hat{\beta}_1)$ given by (3.4). These values clearly minimize the RSS.

3.1.2 Assessing the Accuracy of the Coefficient Estimates

Recall from (2.1) that we assume that the *true* relationship between X and Y takes the form $Y = f(X) + \epsilon$ for some unknown function f, where ϵ is a mean-zero random error term. If f is to be approximated by a linear function, then we can write this relationship as

$$Y = \beta_0 + \beta_1 X + \epsilon. \tag{3.5}$$

Here β_0 is the intercept term—that is, the expected value of Y when $X = 0$, and β_1 is the slope—the average increase in Y associated with a one-unit increase in X. The error term is a catch-all for what we miss with this simple model: the true relationship is probably not linear, there may be other variables that cause variation in Y, and there may be measurement error. We typically assume that the error term is independent of X.

The model given by (3.5) defines the *population regression line*, which is the best linear approximation to the true relationship between X and Y.[1] The least squares regression coefficient estimates (3.4) characterize the *least squares line* (3.2). The left-hand panel of Figure 3.3 displays these

population regression line

least squares line

[1] The assumption of linearity is often a useful working model. However, despite what many textbooks might tell us, we seldom believe that the true relationship is linear.

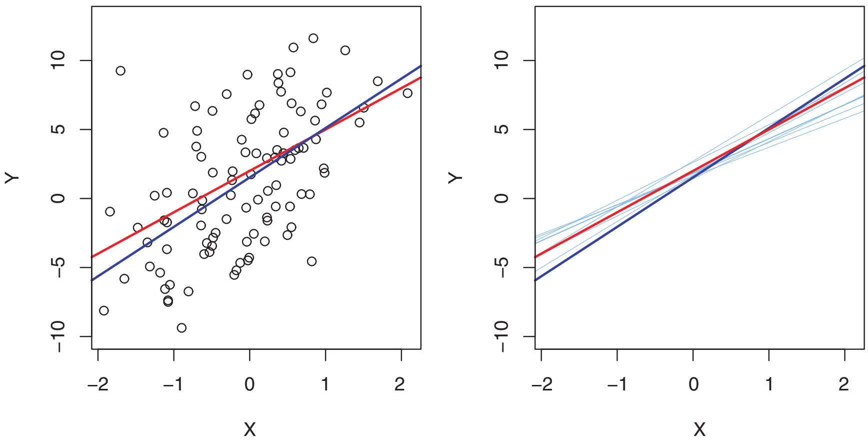

FIGURE 3.3. *A simulated data set.* Left: *The red line represents the true relationship, $f(X) = 2 + 3X$, which is known as the population regression line. The blue line is the least squares line; it is the least squares estimate for $f(X)$ based on the observed data, shown in black.* Right: *The population regression line is again shown in red, and the least squares line in dark blue. In light blue, ten least squares lines are shown, each computed on the basis of a separate random set of observations. Each least squares line is different, but on average, the least squares lines are quite close to the population regression line.*

two lines in a simple simulated example. We created 100 random Xs, and generated 100 corresponding Ys from the model

$$Y = 2 + 3X + \epsilon, \tag{3.6}$$

where   was generated from a normal distribution with mean zero. The red line in the left-hand panel of Figure 3.3 displays the *true* relationship, $f(X) = 2 + 3X$, while the blue line is the least squares estimate based on the observed data. The true relationship is generally not known for real data, but the least squares line can always be computed using the coefficient estimates given in (3.4). In other words, in real applications, we have access to a set of observations from which we can compute the least squares line; however, the population regression line is unobserved. In the right-hand panel of Figure 3.3 we have generated ten different data sets from the model given by (3.6) and plotted the corresponding ten least squares lines. Notice that different data sets generated from the same true model result in slightly different least squares lines, but the unobserved population regression line does not change.

At first glance, the difference between the population regression line and the least squares line may seem subtle and confusing. We only have one data set, and so what does it mean that two different lines describe the relationship between the predictor and the response? Fundamentally, the concept of these two lines is a natural extension of the standard statistical approach of using information from a sample to estimate characteristics of a large population. For example, suppose that we are interested in knowing the population mean μ of some random variable Y. Unfortunately, μ is unknown, but we do have access to n observations from Y, which we can write as $y_1, \ldots, y_n$, and which we can use to estimate μ. A reasonable estimate is $\hat{\mu} = \bar{y}$, where $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ is the sample mean. The sample mean and the population mean are different, but in general the sample mean will provide a good estimate of the population mean. In the same way, the unknown coefficients β_0 and β_1 in linear regression define the population regression line. We seek to estimate these unknown coefficients using $\hat{\beta}_0$ and $\hat{\beta}_1$ given in (3.4). These coefficient estimates define the least squares line.

The analogy between linear regression and estimation of the mean of a random variable is an apt one based on the concept of *bias*. If we use the sample mean $\hat{\mu}$ to estimate μ, this estimate is *unbiased*, in the sense that on average, we expect $\hat{\mu}$ to equal μ. What exactly does this mean? It means that on the basis of one particular set of observations $y_1, \ldots, y_n$, $\hat{\mu}$ might overestimate μ, and on the basis of another set of observations, $\hat{\mu}$ might underestimate μ. But if we could average a huge number of estimates of μ obtained from a huge number of sets of observations, then this average would *exactly* equal μ. Hence, an unbiased estimator does not *systematically* over- or under-estimate the true parameter. The property of unbiasedness holds for the least squares coefficient estimates given by (3.4) as well: if we estimate β_0 and β_1 on the basis of a particular data set, then our estimates won't be exactly equal to β_0 and β_1. But if we could average the estimates obtained over a huge number of data sets, then the average of these estimates would be spot on! In fact, we can see from the right-hand panel of Figure 3.3 that the average of many least squares lines, each estimated from a separate data set, is pretty close to the true population regression line.

bias

unbiased

We continue the analogy with the estimation of the population mean μ of a random variable Y. A natural question is as follows: how accurate is the sample mean $\hat{\mu}$ as an estimate of μ? We have established that the average of $\hat{\mu}$'s over many data sets will be very close to μ, but that a single estimate $\hat{\mu}$ may be a substantial underestimate or overestimate of μ. How far off will that single estimate of $\hat{\mu}$ be? In general, we answer this question by computing the *standard error* of $\hat{\mu}$, written as $\mathrm{SE}(\hat{\mu})$. We have the well-known formula

standard error

$$\mathrm{Var}(\hat{\mu}) = \mathrm{SE}(\hat{\mu})^2 = \frac{\sigma^2}{n}, \tag{3.7}$$

where σ is the standard deviation of each of the realizations y_i of Y.[2] Roughly speaking, the standard error tells us the average amount that this estimate $\hat{\mu}$ differs from the actual value of μ. Equation 3.7 also tells us how this deviation shrinks with n—the more observations we have, the smaller the standard error of $\hat{\mu}$. In a similar vein, we can wonder how close $\hat{\beta}_0$ and $\hat{\beta}_1$ are to the true values β_0 and β_1. To compute the standard errors associated with $\hat{\beta}_0$ and $\hat{\beta}_1$, we use the following formulas:

$$\mathrm{SE}(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right], \quad \mathrm{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{3.8}$$

where $\sigma^2 = \mathrm{Var}(\epsilon)$. For these formulas to be strictly valid, we need to assume that the errors ϵ_i for each observation are uncorrelated with common variance σ^2. This is clearly not true in Figure 3.1, but the formula still turns out to be a good approximation. Notice in the formula that $\mathrm{SE}(\hat{\beta}_1)$ is smaller when the x_i are more spread out; intuitively we have more *leverage* to estimate a slope when this is the case. We also see that $\mathrm{SE}(\hat{\beta}_0)$ would be the same as $\mathrm{SE}(\hat{\mu})$ if $\bar{x}$ were zero (in which case $\hat{\beta}_0$ would be equal to $\bar{y}$). In general, σ^2 is not known, but can be estimated from the data. This estimate is known as the *residual standard error*, and is given by the formula $\mathrm{RSE} = \sqrt{\mathrm{RSS}/(n-2)}$. Strictly speaking, when σ^2 is estimated from the data we should write $\widehat{\mathrm{SE}}(\hat{\beta}_1)$ to indicate that an estimate has been made, but for simplicity of notation we will drop this extra "hat".

residual standard error

Standard errors can be used to compute *confidence intervals*. A 95 % confidence interval is defined as a range of values such that with 95 % probability, the range will contain the true unknown value of the parameter. The range is defined in terms of lower and upper limits computed from the sample of data. For linear regression, the 95 % confidence interval for β_1 approximately takes the form

confidence interval

$$\hat{\beta}_1 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_1). \tag{3.9}$$

That is, there is approximately a 95 % chance that the interval

$$\left[\hat{\beta}_1 - 2 \cdot \mathrm{SE}(\hat{\beta}_1),\ \hat{\beta}_1 + 2 \cdot \mathrm{SE}(\hat{\beta}_1)\right] \tag{3.10}$$

will contain the true value of β_1.[3] Similarly, a confidence interval for β_0 approximately takes the form

$$\hat{\beta}_0 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_0). \tag{3.11}$$

---

[2] This formula holds provided that the n observations are uncorrelated.

[3] *Approximately* for several reasons. Equation 3.10 relies on the assumption that the errors are Gaussian. Also, the factor of 2 in front of the $\mathrm{SE}(\hat{\beta}_1)$ term will vary slightly depending on the number of observations n in the linear regression. To be precise, rather than the number 2, (3.10) should contain the 97.5 % quantile of a t-distribution with $n-2$ degrees of freedom. Details of how to compute the 95 % confidence interval precisely in `R` will be provided later in this chapter.

In the case of the advertising data, the 95 % confidence interval for β_0 is $[6.130, 7.935]$ and the 95 % confidence interval for β_1 is $[0.042, 0.053]$. Therefore, we can conclude that in the absence of any advertising, sales will, on average, fall somewhere between 6,130 and 7,940 units. Furthermore, for each \$1,000 increase in television advertising, there will be an average increase in sales of between 42 and 53 units.

Standard errors can also be used to perform *hypothesis tests* on the coefficients. The most common hypothesis test involves testing the *null hypothesis* of

hypothesis test

null hypothesis

$$H_0 : \text{There is no relationship between } X \text{ and } Y \tag{3.12}$$

versus the *alternative hypothesis*

alternative hypothesis

$$H_a : \text{There is some relationship between } X \text{ and } Y. \tag{3.13}$$

Mathematically, this corresponds to testing

$$H_0 : \beta_1 = 0$$

versus

$$H_a : \beta_1 = 0,$$

since if $\beta_1 = 0$ then the model (3.5) reduces to $Y = \beta_0 + \;$, and X is not associated with Y. To test the null hypothesis, we need to determine whether $\hat{\beta}_1$, our estimate for β_1, is sufficiently far from zero that we can be confident that β_1 is non-zero. How far is far enough? This of course depends on the accuracy of $\hat{\beta}_1$—that is, it depends on $\mathrm{SE}(\hat{\beta}_1)$. If $\mathrm{SE}(\hat{\beta}_1)$ is small, then even relatively small values of $\hat{\beta}_1$ may provide strong evidence that $\beta_1 = 0$, and hence that there is a relationship between X and Y. In contrast, if $\mathrm{SE}(\hat{\beta}_1)$ is large, then $\hat{\beta}_1$ must be large in absolute value in order for us to reject the null hypothesis. In practice, we compute a *t-statistic*, given by

t-statistic

$$t = \frac{\hat{\beta}_1 - 0}{\mathrm{SE}(\hat{\beta}_1)}, \tag{3.14}$$

which measures the number of standard deviations that $\hat{\beta}_1$ is away from 0. If there really is no relationship between X and Y, then we expect that (3.14) will have a t-distribution with $n-2$ degrees of freedom. The t-distribution has a bell shape and for values of n greater than approximately 30 it is quite similar to the normal distribution. Consequently, it is a simple matter to compute the probability of observing any value equal to $|t|$ or larger, assuming $\beta_1 = 0$. We call this probability the *p-value*. Roughly speaking, we interpret the p-value as follows: a small p-value indicates that it is unlikely to observe such a substantial association between the predictor and the response due to chance, in the absence of any real association between the predictor and the response. Hence, if we see a small p-value,

p-value

then we can infer that there is an association between the predictor and the response. We *reject the null hypothesis*—that is, we declare a relationship to exist between X and Y—if the p-value is small enough. Typical p-value cutoffs for rejecting the null hypothesis are 5 or 1 %. When $n = 30$, these correspond to t-statistics (3.14) of around 2 and 2.75, respectively.

	Coefficient	Std. error	t-statistic	p-value
`Intercept`	7.0325	0.4578	15.36	< 0.0001
`TV`	0.0475	0.0027	17.67	< 0.0001

TABLE 3.1. *For the* `Advertising` *data, coefficients of the least squares model for the regression of number of units sold on TV advertising budget. An increase of* \$1,000 *in the TV advertising budget is associated with an increase in sales by around 50 units (Recall that the* `sales` *variable is in thousands of units, and the* `TV` *variable is in thousands of dollars).*

Table 3.1 provides details of the least squares model for the regression of number of units sold on TV advertising budget for the `Advertising` data. Notice that the coefficients for $\hat{\beta}_0$ and $\hat{\beta}_1$ are very large relative to their standard errors, so the t-statistics are also large; the probabilities of seeing such values if H_0 is true are virtually zero. Hence we can conclude that $\beta_0 = 0$ and $\beta_1 = 0$.[4]

3.1.3 Assessing the Accuracy of the Model

Once we have rejected the null hypothesis (3.12) in favor of the alternative hypothesis (3.13), it is natural to want to quantify *the extent to which the model fits the data*. The quality of a linear regression fit is typically assessed using two related quantities: the *residual standard error* (RSE) and the R^2 statistic. R^2

Table 3.2 displays the RSE, the R^2 statistic, and the F-statistic (to be described in Section 3.2.2) for the linear regression of number of units sold on TV advertising budget.

Residual Standard Error

Recall from the model (3.5) that associated with each observation is an error term . Due to the presence of these error terms, even if we knew the true regression line (i.e. even if β_0 and β_1 were known), we would not be able to perfectly predict Y from X. The RSE is an estimate of the standard

[4]In Table 3.1, a small p-value for the intercept indicates that we can reject the null hypothesis that $\beta_0 = 0$, and a small p-value for `TV` indicates that we can reject the null hypothesis that $\beta_1 = 0$. Rejecting the latter null hypothesis allows us to conclude that there is a relationship between `TV` and `sales`. Rejecting the former allows us to conclude that in the absence of `TV` expenditure, `sales` are non-zero.

Quantity	Value
Residual standard error	3.26
R^2	0.612
F-statistic	312.1

TABLE 3.2. *For the* `Advertising` *data, more information about the least squares model for the regression of number of units sold on TV advertising budget.*

deviation of . Roughly speaking, it is the average amount that the response will deviate from the true regression line. It is computed using the formula

$$\mathrm{RSE} = \sqrt{\frac{1}{n-2}\mathrm{RSS}} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{3.15}$$

Note that RSS was defined in Section 3.1.1, and is given by the formula

$$\mathrm{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3.16}$$

In the case of the advertising data, we see from the linear regression output in Table 3.2 that the RSE is 3.26. In other words, actual sales in each market deviate from the true regression line by approximately 3,260 units, on average. Another way to think about this is that even if the model were correct and the true values of the unknown coefficients β_0 and β_1 were known exactly, any prediction of sales on the basis of TV advertising would still be off by about 3,260 units on average. Of course, whether or not 3,260 units is an acceptable prediction error depends on the problem context. In the advertising data set, the mean value of `sales` over all markets is approximately 14,000 units, and so the percentage error is $3{,}260/14{,}000 = 23\,\%$.

The RSE is considered a measure of the *lack of fit* of the model (3.5) to the data. If the predictions obtained using the model are very close to the true outcome values—that is, if $\hat{y}_i \approx y_i$ for $i = 1, \ldots, n$—then (3.15) will be small, and we can conclude that the model fits the data very well. On the other hand, if $\hat{y}_i$ is very far from y_i for one or more observations, then the RSE may be quite large, indicating that the model doesn't fit the data well.

R^2 Statistic

The RSE provides an absolute measure of lack of fit of the model (3.5) to the data. But since it is measured in the units of Y, it is not always clear what constitutes a good RSE. The R^2 statistic provides an alternative measure of fit. It takes the form of a *proportion*—the proportion of variance explained—and so it always takes on a value between 0 and 1, and is independent of the scale of Y.

To calculate R^2, we use the formula

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} \tag{3.17}$$

where $\text{TSS} = \sum (y_i - \bar{y})^2$ is the *total sum of squares*, and RSS is defined in (3.16). TSS measures the total variance in the response Y, and can be thought of as the amount of variability inherent in the response before the regression is performed. In contrast, RSS measures the amount of variability that is left unexplained after performing the regression. Hence, TSS − RSS measures the amount of variability in the response that is explained (or removed) by performing the regression, and R^2 measures the *proportion of variability in Y that can be explained using X*. An R^2 statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression. A number near 0 indicates that the regression did not explain much of the variability in the response; this might occur because the linear model is wrong, or the inherent error σ^2 is high, or both. In Table 3.2, the R^2 was 0.61, and so just under two-thirds of the variability in `sales` is explained by a linear regression on `TV`.

total sum of squares

The R^2 statistic (3.17) has an interpretational advantage over the RSE (3.15), since unlike the RSE, it always lies between 0 and 1. However, it can still be challenging to determine what is a *good* R^2 value, and in general, this will depend on the application. For instance, in certain problems in physics, we may know that the data truly comes from a linear model with a small residual error. In this case, we would expect to see an R^2 value that is extremely close to 1, and a substantially smaller R^2 value might indicate a serious problem with the experiment in which the data were generated. On the other hand, in typical applications in biology, psychology, marketing, and other domains, the linear model (3.5) is at best an extremely rough approximation to the data, and residual errors due to other unmeasured factors are often very large. In this setting, we would expect only a very small proportion of the variance in the response to be explained by the predictor, and an R^2 value well below 0.1 might be more realistic!

The R^2 statistic is a measure of the linear relationship between X and Y. Recall that *correlation*, defined as

correlation

$$\text{Cor}(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}, \tag{3.18}$$

is also a measure of the linear relationship between X and Y.[5] This suggests that we might be able to use $r = \text{Cor}(X, Y)$ instead of R^2 in order to assess the fit of the linear model. In fact, it can be shown that in the simple linear regression setting, $R^2 = r^2$. In other words, the squared correlation

[5]We note that in fact, the right-hand side of (3.18) is the sample correlation; thus, it would be more correct to write $\widehat{\text{Cor}}(X, Y)$; however, we omit the "hat" for ease of notation.

and the R^2 statistic are identical. However, in the next section we will discuss the multiple linear regression problem, in which we use several predictors simultaneously to predict the response. The concept of correlation between the predictors and the response does not extend automatically to this setting, since correlation quantifies the association between a single pair of variables rather than between a larger number of variables. We will see that R^2 fills this role.

3.2 Multiple Linear Regression

Simple linear regression is a useful approach for predicting a response on the basis of a single predictor variable. However, in practice we often have more than one predictor. For example, in the `Advertising` data, we have examined the relationship between sales and TV advertising. We also have data for the amount of money spent advertising on the radio and in newspapers, and we may want to know whether either of these two media is associated with sales. How can we extend our analysis of the advertising data in order to accommodate these two additional predictors?

One option is to run three separate simple linear regressions, each of which uses a different advertising medium as a predictor. For instance, we can fit a simple linear regression to predict sales on the basis of the amount spent on radio advertisements. Results are shown in Table 3.3 (top table). We find that a \$1,000 increase in spending on radio advertising is associated with an increase in sales by around 203 units. Table 3.3 (bottom table) contains the least squares coefficients for a simple linear regression of sales onto newspaper advertising budget. A \$1,000 increase in newspaper advertising budget is associated with an increase in sales by approximately 55 units.

However, the approach of fitting a separate simple linear regression model for each predictor is not entirely satisfactory. First of all, it is unclear how to make a single prediction of sales given levels of the three advertising media budgets, since each of the budgets is associated with a separate regression equation. Second, each of the three regression equations ignores the other two media in forming estimates for the regression coefficients. We will see shortly that if the media budgets are correlated with each other in the 200 markets that constitute our data set, then this can lead to very misleading estimates of the individual media effects on sales.

Instead of fitting a separate simple linear regression model for each predictor, a better approach is to extend the simple linear regression model (3.5) so that it can directly accommodate multiple predictors. We can do this by giving each predictor a separate slope coefficient in a single model. In general, suppose that we have p distinct predictors. Then the multiple linear regression model takes the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon, \tag{3.19}$$

Simple regression of `sales` on `radio`

	Coefficient	Std. error	t-statistic	p-value
`Intercept`	9.312	0.563	16.54	< 0.0001
`radio`	0.203	0.020	9.92	< 0.0001

Simple regression of `sales` on `newspaper`

	Coefficient	Std. error	t-statistic	p-value
`Intercept`	12.351	0.621	19.88	< 0.0001
`newspaper`	0.055	0.017	3.30	< 0.0001

TABLE 3.3. *More simple linear regression models for the* `Advertising` *data. Coefficients of the simple linear regression model for number of units sold on* Top: *radio advertising budget and* Bottom: *newspaper advertising budget. A \$1,000 increase in spending on radio advertising is associated with an average increase in sales by around 203 units, while the same increase in spending on newspaper advertising is associated with an average increase in sales by around 55 units (Note that the* `sales` *variable is in thousands of units, and the* `radio` *and* `newspaper` *variables are in thousands of dollars).*

where X_j represents the jth predictor and β_j quantifies the association between that variable and the response. We interpret β_j as the *average* effect on Y of a one unit increase in X_j, *holding all other predictors fixed.* In the advertising example, (3.19) becomes

$$\texttt{sales} = \beta_0 + \beta_1 \times \texttt{TV} + \beta_2 \times \texttt{radio} + \beta_3 \times \texttt{newspaper} + \epsilon. \tag{3.20}$$

3.2.1 Estimating the Regression Coefficients

As was the case in the simple linear regression setting, the regression coefficients $\beta_0, \beta_1, \ldots, \beta_p$ in (3.19) are unknown, and must be estimated. Given estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$, we can make predictions using the formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p. \tag{3.21}$$

The parameters are estimated using the same least squares approach that we saw in the context of simple linear regression. We choose $\beta_0, \beta_1, \ldots, \beta_p$ to minimize the sum of squared residuals

$$\begin{aligned} \text{RSS} &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2. \end{aligned} \tag{3.22}$$

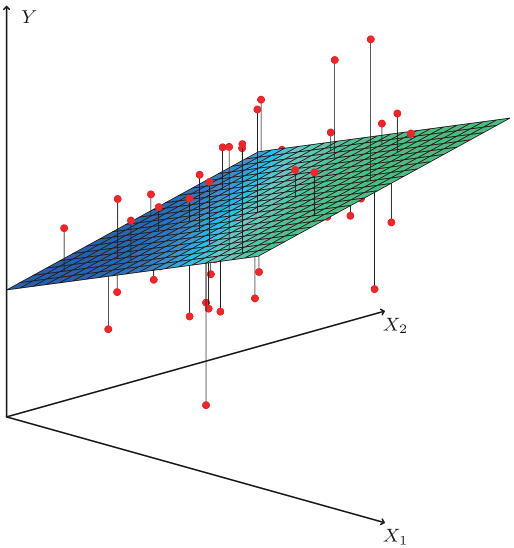

FIGURE 3.4. *In a three-dimensional setting, with two predictors and one response, the least squares regression line becomes a plane. The plane is chosen to minimize the sum of the squared vertical distances between each observation (shown in red) and the plane.*

The values $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ that minimize (3.22) are the multiple least squares regression coefficient estimates. Unlike the simple linear regression estimates given in (3.4), the multiple regression coefficient estimates have somewhat complicated forms that are most easily represented using matrix algebra. For this reason, we do not provide them here. Any statistical software package can be used to compute these coefficient estimates, and later in this chapter we will show how this can be done in `R`. Figure 3.4 illustrates an example of the least squares fit to a toy data set with $p = 2$ predictors.

Table 3.4 displays the multiple regression coefficient estimates when TV, radio, and newspaper advertising budgets are used to predict product sales using the `Advertising` data. We interpret these results as follows: for a given amount of TV and newspaper advertising, spending an additional \$1,000 on radio advertising leads to an increase in sales by approximately 189 units. Comparing these coefficient estimates to those displayed in Tables 3.1 and 3.3, we notice that the multiple regression coefficient estimates for `TV` and `radio` are pretty similar to the simple linear regression coefficient estimates. However, while the `newspaper` regression coefficient estimate in Table 3.3 was significantly non-zero, the coefficient estimate for `newspaper` in the multiple regression model is close to zero, and the corresponding p-value is no longer significant, with a value around 0.86. This illustrates

	Coefficient	Std. error	t-statistic	p-value
`Intercept`	2.939	0.3119	9.42	< 0.0001
`TV`	0.046	0.0014	32.81	< 0.0001
`radio`	0.189	0.0086	21.89	< 0.0001
`newspaper`	-0.001	0.0059	-0.18	0.8599

TABLE 3.4. *For the* `Advertising` *data, least squares coefficient estimates of the multiple linear regression of number of units sold on radio, TV, and newspaper advertising budgets.*

that the simple and multiple regression coefficients can be quite different. This difference stems from the fact that in the simple regression case, the slope term represents the average effect of a $1,000 increase in newspaper advertising, ignoring other predictors such as `TV` and `radio`. In contrast, in the multiple regression setting, the coefficient for `newspaper` represents the average effect of increasing newspaper spending by $1,000 while holding `TV` and `radio` fixed.

Does it make sense for the multiple regression to suggest no relationship between `sales` and `newspaper` while the simple linear regression implies the opposite? In fact it does. Consider the correlation matrix for the three predictor variables and response variable, displayed in Table 3.5. Notice that the correlation between `radio` and `newspaper` is 0.35. This reveals a tendency to spend more on newspaper advertising in markets where more is spent on radio advertising. Now suppose that the multiple regression is correct and newspaper advertising has no direct impact on sales, but radio advertising does increase sales. Then in markets where we spend more on radio our sales will tend to be higher, and as our correlation matrix shows, we also tend to spend more on newspaper advertising in those same markets. Hence, in a simple linear regression which only examines `sales` versus `newspaper`, we will observe that higher values of `newspaper` tend to be associated with higher values of `sales`, even though newspaper advertising does not actually affect sales. So `newspaper` sales are a surrogate for `radio` advertising; `newspaper` gets "credit" for the effect of `radio` on `sales`.

This slightly counterintuitive result is very common in many real life situations. Consider an absurd example to illustrate the point. Running a regression of shark attacks versus ice cream sales for data collected at a given beach community over a period of time would show a positive relationship, similar to that seen between `sales` and `newspaper`. Of course no one (yet) has suggested that ice creams should be banned at beaches to reduce shark attacks. In reality, higher temperatures cause more people to visit the beach, which in turn results in more ice cream sales and more shark attacks. A multiple regression of attacks versus ice cream sales and temperature reveals that, as intuition implies, the former predictor is no longer significant after adjusting for temperature.

	TV	radio	newspaper	sales
TV	1.0000	0.0548	0.0567	0.7822
radio		1.0000	0.3541	0.5762
newspaper			1.0000	0.2283
sales				1.0000

TABLE 3.5. *Correlation matrix for* `TV`, `radio`, `newspaper`, *and* `sales` *for the* `Advertising` *data.*

3.2.2 Some Important Questions

When we perform multiple linear regression, we usually are interested in answering a few important questions.

1. *Is at least one of the predictors $X_1, X_2, \dots, X_p$ useful in predicting the response?*

2. *Do all the predictors help to explain Y, or is only a subset of the predictors useful?*

3. *How well does the model fit the data?*

4. *Given a set of predictor values, what response value should we predict, and how accurate is our prediction?*

We now address each of these questions in turn.

One: Is There a Relationship Between the Response and Predictors?

Recall that in the simple linear regression setting, in order to determine whether there is a relationship between the response and the predictor we can simply check whether $\beta_1 = 0$. In the multiple regression setting with p predictors, we need to ask whether all of the regression coefficients are zero, i.e. whether $\beta_1 = \beta_2 = \cdots = \beta_p = 0$. As in the simple linear regression setting, we use a hypothesis test to answer this question. We test the null hypothesis,

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

versus the alternative

$$H_a : \text{ at least one } \beta_j \text{ is non-zero.}$$

This hypothesis test is performed by computing the *F-statistic*, F-statistic

$$F = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n-p-1)}, \tag{3.23}$$

Quantity	Value
Residual standard error	1.69
R^2	0.897
F-statistic	570

TABLE 3.6. *More information about the least squares model for the regression of number of units sold on TV, newspaper, and radio advertising budgets in the* `Advertising` *data. Other information about this model was displayed in Table 3.4.*

where, as with simple linear regression, TSS $=$ $(y_i - \bar{y})^2$ and RSS $=$ $(y_i - \hat{y}_i)^2$. If the linear model assumptions are correct, one can show that

$$E\{\text{RSS}/(n-p-1)\} = \sigma^2$$

and that, provided H_0 is true,

$$E\{(\text{TSS} - \text{RSS})/p\} = \sigma^2.$$

Hence, when there is no relationship between the response and predictors, one would expect the F-statistic to take on a value close to 1. On the other hand, if H_a is true, then $E\{(\text{TSS} - \text{RSS})/p\} > \sigma^2$, so we expect F to be greater than 1.

The F-statistic for the multiple linear regression model obtained by regressing `sales` onto `radio`, `TV`, and `newspaper` is shown in Table 3.6. In this example the F-statistic is 570. Since this is far larger than 1, it provides compelling evidence against the null hypothesis H_0. In other words, the large F-statistic suggests that at least one of the advertising media must be related to `sales`. However, what if the F-statistic had been closer to 1? How large does the F-statistic need to be before we can reject H_0 and conclude that there is a relationship? It turns out that the answer depends on the values of n and p. When n is large, an F-statistic that is just a little larger than 1 might still provide evidence against H_0. In contrast, a larger F-statistic is needed to reject H_0 if n is small. When H_0 is true and the errors $_i$ have a normal distribution, the F-statistic follows an F-distribution.[6] For any given value of n and p, any statistical software package can be used to compute the p-value associated with the F-statistic using this distribution. Based on this p-value, we can determine whether or not to reject H_0. For the advertising data, the p-value associated with the F-statistic in Table 3.6 is essentially zero, so we have extremely strong evidence that at least one of the media is associated with increased `sales`.

In (3.23) we are testing H_0 that all the coefficients are zero. Sometimes we want to test that a particular subset of q of the coefficients are zero. This corresponds to a null hypothesis

$$H_0: \quad \beta_{p-q+1} = \beta_{p-q+2} = \ldots = \beta_p = 0,$$

[6]Even if the errors are not normally-distributed, the F-statistic approximately follows an F-distribution provided that the sample size n is large.

where for convenience we have put the variables chosen for omission at the end of the list. In this case we fit a second model that uses all the variables *except* those last q. Suppose that the residual sum of squares for that model is RSS_0. Then the appropriate F-statistic is

$$F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n-p-1)}. \tag{3.24}$$

Notice that in Table 3.4, for each individual predictor a t-statistic and a p-value were reported. These provide information about whether each individual predictor is related to the response, after adjusting for the other predictors. It turns out that each of these are exactly equivalent[7] to the F-test that omits that single variable from the model, leaving all the others in—i.e. q=1 in (3.24). So it reports the *partial effect* of adding that variable to the model. For instance, as we discussed earlier, these p-values indicate that `TV` and `radio` are related to `sales`, but that there is no evidence that `newspaper` is associated with `sales`, in the presence of these two.

Given these individual p-values for each variable, why do we need to look at the overall F-statistic? After all, it seems likely that if any one of the p-values for the individual variables is very small, then *at least one of the predictors is related to the response*. However, this logic is flawed, especially when the number of predictors p is large.

For instance, consider an example in which $p = 100$ and $H_0 : \beta_1 = \beta_2 = \ldots = \beta_p = 0$ is true, so no variable is truly associated with the response. In this situation, about 5 % of the p-values associated with each variable (of the type shown in Table 3.4) will be below 0.05 by chance. In other words, we expect to see approximately five *small* p-values even in the absence of any true association between the predictors and the response. In fact, we are almost guaranteed that we will observe at least one p-value below 0.05 by chance! Hence, if we use the individual t-statistics and associated p-values in order to decide whether or not there is any association between the variables and the response, there is a very high chance that we will incorrectly conclude that there is a relationship. However, the F-statistic does not suffer from this problem because it adjusts for the number of predictors. Hence, if H_0 is true, there is only a 5 % chance that the F-statistic will result in a p-value below 0.05, regardless of the number of predictors or the number of observations.

The approach of using an F-statistic to test for any association between the predictors and the response works when p is relatively small, and certainly small compared to n. However, sometimes we have a very large number of variables. If $p > n$ then there are more coefficients β_j to estimate than observations from which to estimate them. In this case we cannot even fit the multiple linear regression model using least squares, so the

[7]The square of each t-statistic is the corresponding F-statistic.

F-statistic cannot be used, and neither can most of the other concepts that we have seen so far in this chapter. When p is large, some of the approaches discussed in the next section, such as *forward selection*, can be used. This *high-dimensional* setting is discussed in greater detail in Chapter 6.

high-dimensional

Two: Deciding on Important Variables

As discussed in the previous section, the first step in a multiple regression analysis is to compute the F-statistic and to examine the associated p-value. If we conclude on the basis of that p-value that at least one of the predictors is related to the response, then it is natural to wonder *which* are the guilty ones! We could look at the individual p-values as in Table 3.4, but as discussed, if p is large we are likely to make some false discoveries.

It is possible that all of the predictors are associated with the response, but it is more often the case that the response is only related to a subset of the predictors. The task of determining which predictors are associated with the response, in order to fit a single model involving only those predictors, is referred to as *variable selection*. The variable selection problem is studied extensively in Chapter 6, and so here we will provide only a brief outline of some classical approaches.

variable selection

Ideally, we would like to perform variable selection by trying out a lot of different models, each containing a different subset of the predictors. For instance, if $p = 2$, then we can consider four models: (1) a model containing no variables, (2) a model containing X_1 only, (3) a model containing X_2 only, and (4) a model containing both X_1 and X_2. We can then select the *best* model out of all of the models that we have considered. How do we determine which model is best? Various statistics can be used to judge the quality of a model. These include *Mallow's C_p*, *Akaike information criterion* (AIC), *Bayesian information criterion* (BIC), and *adjusted R^2*. These are discussed in more detail in Chapter 6. We can also determine which model is best by plotting various model outputs, such as the residuals, in order to search for patterns.

Mallow's C_p

Akaike information criterion

Bayesian information criterion

adjusted R^2

Unfortunately, there are a total of 2^p models that contain subsets of p variables. This means that even for moderate p, trying out every possible subset of the predictors is infeasible. For instance, we saw that if $p = 2$, then there are $2^2 = 4$ models to consider. But if $p = 30$, then we must consider $2^{30} = 1{,}073{,}741{,}824$ models! This is not practical. Therefore, unless p is very small, we cannot consider all 2^p models, and instead we need an automated and efficient approach to choose a smaller set of models to consider. There are three classical approaches for this task:

- *Forward selection*. We begin with the *null model*—a model that contains an intercept but no predictors. We then fit p simple linear regressions and add to the null model the variable that results in the lowest RSS. We then add to that model the variable that results

forward selection

null model

in the lowest RSS for the new two-variable model. This approach is continued until some stopping rule is satisfied.

- *Backward selection*. We start with all variables in the model, and remove the variable with the largest p-value—that is, the variable that is the least statistically significant. The new $(p-1)$-variable model is fit, and the variable with the largest p-value is removed. This procedure continues until a stopping rule is reached. For instance, we may stop when all remaining variables have a p-value below some threshold. backward selection

- *Mixed selection*. This is a combination of forward and backward selection. We start with no variables in the model, and as with forward selection, we add the variable that provides the best fit. We continue to add variables one-by-one. Of course, as we noted with the `Advertising` example, the p-values for variables can become larger as new predictors are added to the model. Hence, if at any point the p-value for one of the variables in the model rises above a certain threshold, then we remove that variable from the model. We continue to perform these forward and backward steps until all variables in the model have a sufficiently low p-value, and all variables outside the model would have a large p-value if added to the model. mixed selection

Backward selection cannot be used if $p > n$, while forward selection can always be used. Forward selection is a greedy approach, and might include variables early that later become redundant. Mixed selection can remedy this.

Three: Model Fit

Two of the most common numerical measures of model fit are the RSE and R^2, the fraction of variance explained. These quantities are computed and interpreted in the same fashion as for simple linear regression.

Recall that in simple regression, R^2 is the square of the correlation of the response and the variable. In multiple linear regression, it turns out that it equals $\mathrm{Cor}(Y, \hat{Y})^2$, the square of the correlation between the response and the fitted linear model; in fact one property of the fitted linear model is that it maximizes this correlation among all possible linear models.

An R^2 value close to 1 indicates that the model explains a large portion of the variance in the response variable. As an example, we saw in Table 3.6 that for the `Advertising` data, the model that uses all three advertising media to predict `sales` has an R^2 of 0.8972. On the other hand, the model that uses only `TV` and `radio` to predict `sales` has an R^2 value of 0.89719. In other words, there is a *small* increase in R^2 if we include newspaper advertising in the model that already contains TV and radio advertising, even though we saw earlier that the p-value for newspaper advertising in Table 3.4 is not significant. It turns out that R^2 will always increase when more variables are added to the model, even if those variables are only weakly associated with the response. This is due to the fact that adding another variable to the least squares equations must allow us to fit the training data (though not necessarily the testing data) more accurately. Thus, the R^2 statistic, which is also computed on the training data, must increase. The fact that adding newspaper advertising to the model containing only TV and radio advertising leads to just a tiny increase in R^2 provides additional evidence that `newspaper` can be dropped from the model. Essentially, `newspaper` provides no real improvement in the model fit to the training samples, and its inclusion will likely lead to poor results on independent test samples due to overfitting.

In contrast, the model containing only `TV` as a predictor had an R^2 of 0.61 (Table 3.2). Adding `radio` to the model leads to a substantial improvement in R^2. This implies that a model that uses TV and radio expenditures to predict sales is substantially better than one that uses only TV advertising. We could further quantify this improvement by looking at the p-value for the `radio` coefficient in a model that contains only `TV` and `radio` as predictors.

The model that contains only `TV` and `radio` as predictors has an RSE of 1.681, and the model that also contains `newspaper` as a predictor has an RSE of 1.686 (Table 3.6). In contrast, the model that contains only `TV` has an RSE of 3.26 (Table 3.2). This corroborates our previous conclusion that a model that uses TV and radio expenditures to predict sales is much more accurate (on the training data) than one that only uses TV spending. Furthermore, given that TV and radio expenditures are used as predictors, there is no point in also using newspaper spending as a predictor in the model. The observant reader may wonder how RSE can increase when `newspaper` is added to the model given that RSS must decrease. In general RSE is defined as

$$\text{RSE} = \sqrt{\frac{1}{n-p-1}\text{RSS}}, \tag{3.25}$$

which simplifies to (3.15) for a simple linear regression. Thus, models with more variables can have higher RSE if the decrease in RSS is small relative to the increase in p.

In addition to looking at the RSE and R^2 statistics just discussed, it can be useful to plot the data. Graphical summaries can reveal problems with a model that are not visible from numerical statistics. For example, Figure 3.5 displays a three-dimensional plot of `TV` and `radio` versus `sales`. We see that some observations lie above and some observations lie below the least squares regression plane. Notice that there is a clear pattern of negative residuals, followed by positive residuals, followed by negative residuals. In particular, the linear model seems to overestimate `sales` for instances in which most of the advertising money was spent exclusively on either

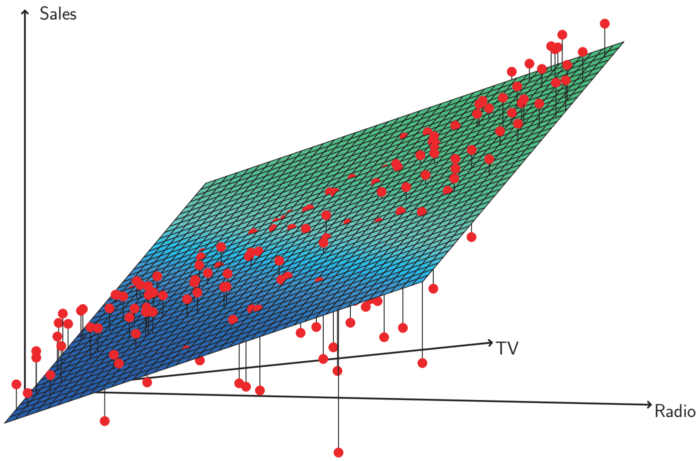

FIGURE 3.5. *For the* `Advertising` *data, a linear regression fit to* `sales` *using* `TV` *and* `radio` *as predictors. From the pattern of the residuals, we can see that there is a pronounced non-linear relationship in the data.*

`TV` or `radio`. It underestimates `sales` for instances where the budget was split between the two media. This pronounced non-linear pattern cannot be modeled accurately using linear regression. It suggests a *synergy* or *interaction* effect between the advertising media, whereby combining the media together results in a bigger boost to sales than using any single medium. In Section 3.3.2, we will discuss extending the linear model to accommodate such synergistic effects through the use of interaction terms.

Four: Predictions

Once we have fit the multiple regression model, it is straightforward to apply (3.21) in order to predict the response Y on the basis of a set of values for the predictors $X_1, X_2, \ldots, X_p$. However, there are three sorts of uncertainty associated with this prediction.

1. The coefficient estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ are estimates for $\beta_0, \beta_1, \ldots, \beta_p$. That is, the *least squares plane*

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p$$

is only an estimate for the *true population regression plane*

$$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

The inaccuracy in the coefficient estimates is related to the *reducible error* from Chapter 2. We can compute a *confidence interval* in order to determine how close $\hat{Y}$ will be to $f(X)$.

2. Of course, in practice assuming a linear model for $f(X)$ is almost always an approximation of reality, so there is an additional source of potentially reducible error which we call *model bias*. So when we use a linear model, we are in fact estimating the best linear approximation to the true surface. However, here we will ignore this discrepancy, and operate as if the linear model were correct.

3. Even if we knew $f(X)$—that is, even if we knew the true values for $\beta_0, \beta_1, \ldots, \beta_p$—the response value cannot be predicted perfectly because of the random error in the model (3.21). In Chapter 2, we referred to this as the *irreducible error*. How much will Y vary from $\hat{Y}$? We use *prediction intervals* to answer this question. Prediction intervals are always wider than confidence intervals, because they incorporate both the error in the estimate for $f(X)$ (the reducible error) and the uncertainty as to how much an individual point will differ from the population regression plane (the irreducible error).

We use a *confidence interval* to quantify the uncertainty surrounding the *average* `sales` over a large number of cities. For example, given that \$100,000 is spent on `TV` advertising and \$20,000 is spent on `radio` advertising in each city, the 95 % confidence interval is [10,985, 11,528]. We interpret this to mean that 95 % of intervals of this form will contain the true value of $f(X)$.[8] On the other hand, a *prediction interval* can be used to quantify the uncertainty surrounding `sales` for a *particular* city. Given that \$100,000 is spent on `TV` advertising and \$20,000 is spent on `radio` advertising in that city the 95 % prediction interval is [7,930, 14,580]. We interpret this to mean that 95 % of intervals of this form will contain the true value of Y for this city. Note that both intervals are centered at 11,256, but that the prediction interval is substantially wider than the confidence interval, reflecting the increased uncertainty about `sales` for a given city in comparison to the average `sales` over many locations.

confidence interval

prediction interval

3.3 Other Considerations in the Regression Model

3.3.1 Qualitative Predictors

In our discussion so far, we have assumed that all variables in our linear regression model are *quantitative*. But in practice, this is not necessarily the case; often some predictors are *qualitative*.

---

[8] In other words, if we collect a large number of data sets like the `Advertising` data set, and we construct a confidence interval for the average `sales` on the basis of each data set (given \$100,000 in `TV` and \$20,000 in `radio` advertising), then 95 % of these confidence intervals will contain the true value of average `sales`.

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: binary

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

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 $\Pr(\texttt{drug overdose}|X)$ 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.

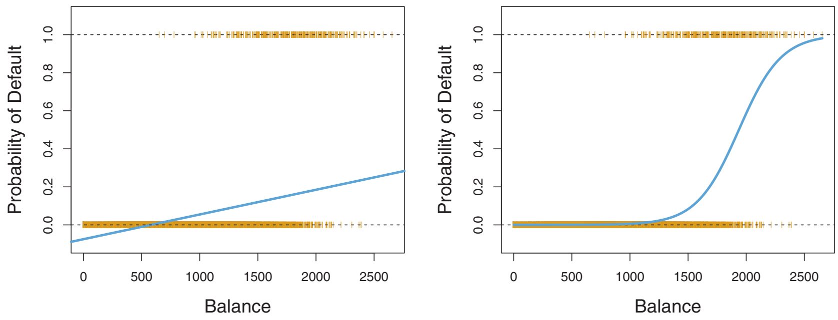

FIGURE 4.2. *Classification using the* `Default` *data.* Left: *Estimated probability of* `default` *using linear regression. Some estimated probabilities are negative! The orange ticks indicate the 0/1 values coded for* `default`*(*`No` *or* `Yes`*).* Right: *Predicted probabilities of* `default` *using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

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

The values of $\Pr(\texttt{default} = \texttt{Yes}|\texttt{balance})$, which we abbreviate $p(\texttt{balance})$, 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 $p(\texttt{balance}) > 0.5$. 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 $p(\texttt{balance}) > 0.1$.

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:

$$p(X) = \beta_0 + \beta_1 X. \tag{4.1}$$

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*,

logistic function

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

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

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

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

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

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*:

likelihood function

$$(\beta_0, \beta_1) = \underset{i:y_i=1}{} p(x_i) \underset{i\ :y_i\ =0}{} (1 - p(x_i\ )). \tag{4.5}$$

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:

$$\Pr(\texttt{default}=\texttt{Yes}|\texttt{student}=\texttt{Yes}) = \frac{e^{-3.5041+0.4049\times 1}}{1+e^{-3.5041+0.4049\times 1}} = 0.0431,$$
$$\Pr(\texttt{default}=\texttt{Yes}|\texttt{student}=\texttt{No}) = \frac{e^{-3.5041+0.4049\times 0}}{1+e^{-3.5041+0.4049\times 0}} = 0.0292.$$

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:

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

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

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

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

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

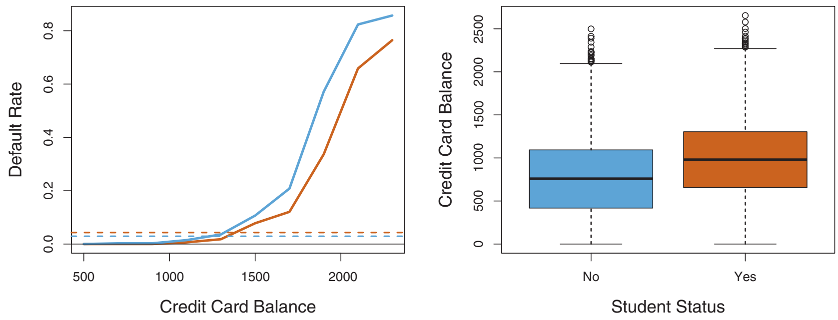

FIGURE 4.3. *Confounding in the* `Default` *data.* Left: *Default rates are shown for students (orange) and non-students (blue). The solid lines display default rate as a function of* `balance`*, while the horizontal broken lines display the overall default rates.* Right: *Boxplots of* `balance` *for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

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

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

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

(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 $\Pr(Y = \texttt{stroke}|X)$ and $\Pr(Y = \texttt{drug overdose}|X)$, with the remaining $\Pr(Y = \texttt{epileptic seizure}|X) = 1 - \Pr(Y = \texttt{stroke}|X) - \Pr(Y = \texttt{drug overdose}|X)$. 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 kth class; this is the probability that a given observation is associated with the kth 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 kth class. In other words, $f_k(x)$ is relatively large if there is a high probability that an observation in the kth class has $X \approx x$, and $f_k(x)$ is small if it is very

prior

density function

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

Bayes' theorem

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

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 Ys from the population: we simply compute the fraction of the training observations that belong to the kth 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 kth class. That is, it is the probability that the observation belongs to the kth 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

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

where μ_k and σ_k^2 are the mean and variance parameters for the kth class. For now, let us further assume that $\sigma_1^2 = \ldots = \sigma_K^2$: 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

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

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

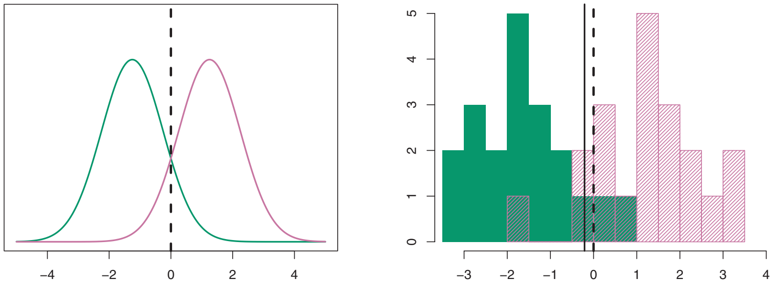

FIGURE 4.4. Left: *Two one-dimensional normal density functions are shown. The dashed vertical line represents the Bayes decision boundary.* Right: *20 observations were drawn from each of the two classes, and are shown as histograms. The Bayes decision boundary is again shown as a dashed vertical line.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

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

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

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

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 $\mu_1, \ldots, \mu_K$, $\pi_1, \ldots, \pi_K$, 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

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

where n is the total number of training observations, and n_k is the number of training observations in the kth class. The estimate for μ_k is simply the average of all the training observations from the kth 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 $\pi_1, \ldots, \pi_K$, 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 kth class. In other words,

$$\hat{\pi}_k = n_k/n. \tag{4.16}$$

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

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

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.

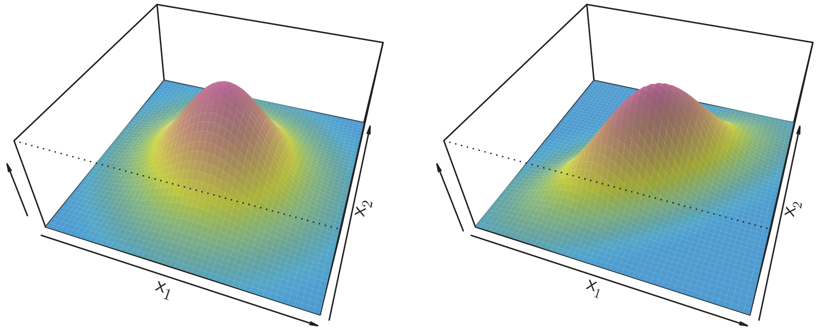

FIGURE 4.5. *Two multivariate Gaussian density functions are shown, with* $p = 2$. Left: *The two predictors are uncorrelated.* Right: *The two variables have a correlation of* 0.7.

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 kth 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 $X = (X_1, X_2, \ldots, X_p)$ 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 $\mathrm{Var}(X_1) = \mathrm{Var}(X_2)$ and $\mathrm{Cor}(X_1, X_2) = 0$; 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,

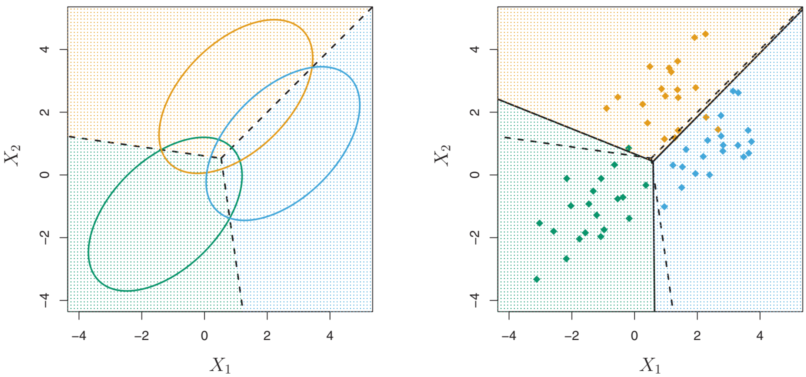

FIGURE 4.6. *An example with three classes. The observations from each class are drawn from a multivariate Gaussian distribution with $p = 2$, with a class-specific mean vector and a common covariance matrix.* Left: *Ellipses that contain 95 % of the probability for each of the three classes are shown. 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 $X \sim N(\mu, \mathbf{\Sigma})$. Here $E(X) = \mu$ is the mean of X (a vector with p components), and $\text{Cov}(X) = \mathbf{\Sigma}$ is the $p \times p$ covariance matrix of X. Formally, the multivariate Gaussian density is defined as

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

In the case of $p > 1$ predictors, the LDA classifier assumes that the observations in the kth class are drawn from a multivariate Gaussian distribution $N(\mu_k, \mathbf{\Sigma})$, where μ_k is a class-specific mean vector, and $\mathbf{\Sigma}$ is a covariance matrix that is common to all K classes. Plugging the density function for the kth 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

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

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 $\delta_k(x) = \delta\ (x)$; i.e.

$$x^T \boldsymbol{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^T \boldsymbol{\Sigma}^{-1} \mu_k = x^T \boldsymbol{\Sigma}^{-1} \mu_l - \frac{1}{2} \mu_l^T \boldsymbol{\Sigma}^{-1} \mu_l \tag{4.20}$$

for $k = l$. (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 $\mu_1, \dots, \mu_K$, $\pi_1, \dots, \pi_K$, and $\boldsymbol{\Sigma}$; 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	No	9,644	252	9,896
default status	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. null

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.

confusion matrix

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 $(1 - 23/9{,}667) \times 100 = 99.8\,\%$.

sensitivity specificity

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

		True default status		
		No	Yes	Total
Predicted	No	9,432	138	9,570
default status	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

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

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

$$P(\texttt{default} = \texttt{Yes}|X = x) > 0.2. \tag{4.22}$$

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

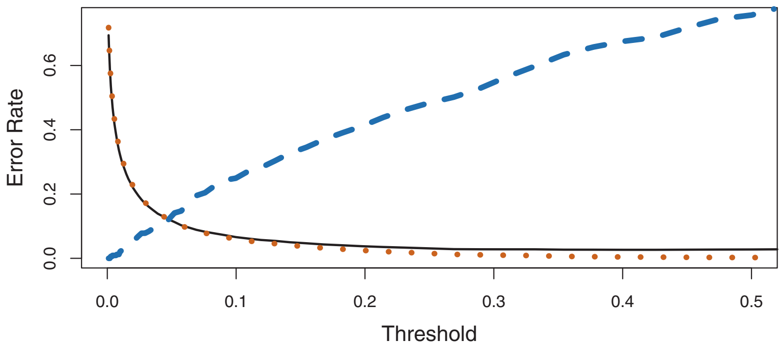

FIGURE 4.7. *For the* `Default` *data set, error rates are shown as a function of the threshold value for the posterior probability that is used to perform the assignment. The black solid line displays the overall error rate. The blue dashed line represents the fraction of defaulting customers that are 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.

ROC curve

area under the (ROC) curve

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

sensitivity

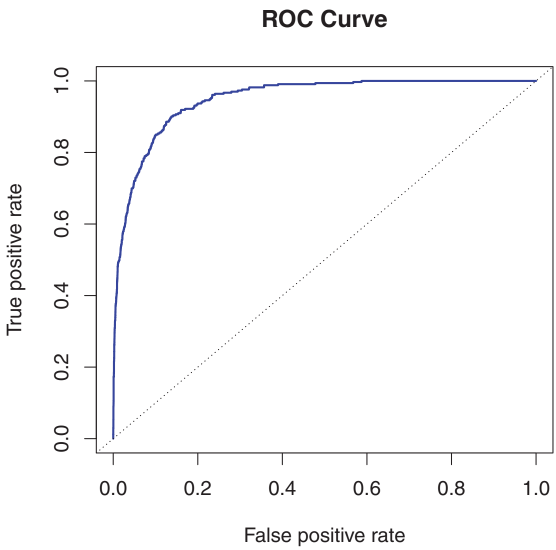

FIGURE 4.8. *A ROC curve for the LDA classifier on the* `Default` *data. It traces out two types of error as we vary the threshold value for the posterior probability of default. The actual thresholds are not shown. The true positive rate is the sensitivity: the fraction of defaulters that are correctly 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	− or Null	True Neg. (TN)	False Pos. (FP)	N
class	+ 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 specificity 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 kth class is of the form $X \sim N(\mu_k, \mathbf{\Sigma}_k)$, where $\mathbf{\Sigma}_k$ is a covariance matrix for the kth class. Under this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

quadratic discriminant analysis

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

is largest. So the QDA classifier involves plugging estimates for $\mathbf{\Sigma}_k$, 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

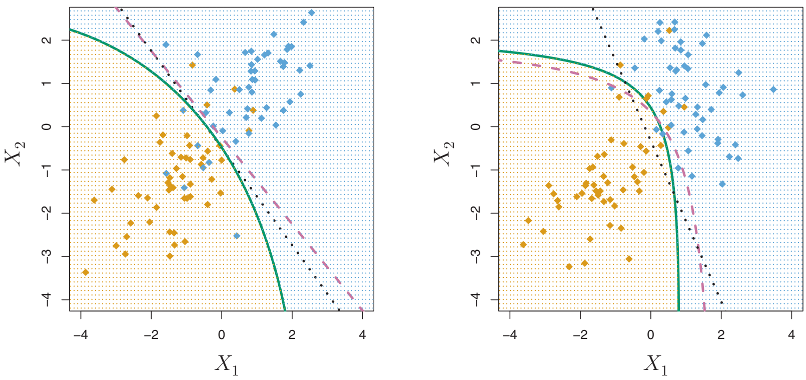

FIGURE 4.9. Left: *The Bayes (purple dashed), LDA (black dotted), and QDA (green solid) decision boundaries for a two-class problem with* $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2$*. The shading indicates the QDA decision rule. Since the Bayes decision boundary is linear, it is more accurately approximated by LDA than by QDA.* Right: *Details are as given in the left-hand panel, except that* $\mathbf{\Sigma}_1 \neq \mathbf{\Sigma}_2$*. 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.

6.1 Subset Selection

In this section we consider some methods for selecting subsets of predictors. These include best subset and stepwise model selection procedures.

6.1.1 Best Subset Selection

To perform *best subset selection*, we fit a separate least squares regression for each possible combination of the p predictors. That is, we fit all p models that contain exactly one predictor, all $\binom{p}{2} = p(p-1)/2$ models that contain exactly two predictors, and so forth. We then look at all of the resulting models, with the goal of identifying the one that is *best*.

best subset selection

The problem of selecting the *best model* from among the 2^p possibilities considered by best subset selection is not trivial. This is usually broken up into two stages, as described in Algorithm 6.1.

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

In Algorithm 6.1, Step 2 identifies the best model (on the training data) for each subset size, in order to reduce the problem from one of 2^p possible models to one of $p + 1$ possible models. In Figure 6.1, these models form the lower frontier depicted in red.

Now in order to select a single best model, we must simply choose among these $p + 1$ options. This task must be performed with care, because the RSS of these $p + 1$ models decreases monotonically, and the R^2 increases monotonically, as the number of features included in the models increases. Therefore, if we use these statistics to select the best model, then we will always end up with a model involving all of the variables. The problem is that a low RSS or a high R^2 indicates a model with a low *training* error, whereas we wish to choose a model that has a low *test* error. (As shown in Chapter 2 in Figures 2.9–2.11, training error tends to be quite a bit smaller than test error, and a low training error by no means guarantees a low test error.) Therefore, in Step 3, we use cross-validated prediction

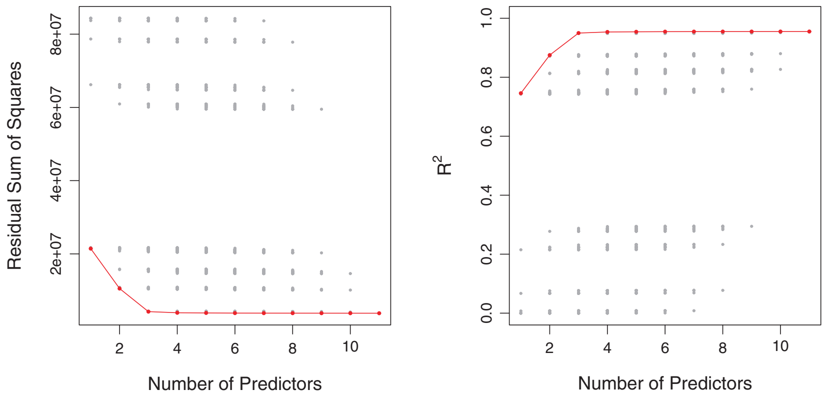

FIGURE 6.1. *For each possible model containing a subset of the ten predictors in the* `Credit` *data set, the RSS and* R^2 *are displayed. The red frontier tracks the* best *model for a given number of predictors, according to RSS and* R^2*. Though the data set contains only ten predictors, the* x*-axis ranges from* 1 *to* 11*, since one of the variables is categorical and takes on three values, leading to the creation of two dummy variables.*

error, C_p, BIC, or adjusted R^2 in order to select among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$. These approaches are discussed in Section 6.1.3.

An application of best subset selection is shown in Figure 6.1. Each plotted point corresponds to a least squares regression model fit using a different subset of the 11 predictors in the `Credit` data set, discussed in Chapter 3. Here the variable `ethnicity` is a three-level qualitative variable, and so is represented by two dummy variables, which are selected separately in this case. We have plotted the RSS and R^2 statistics for each model, as a function of the number of variables. The red curves connect the best models for each model size, according to RSS or R^2. The figure shows that, as expected, these quantities improve as the number of variables increases; however, from the three-variable model on, there is little improvement in RSS and R^2 as a result of including additional predictors.

Although we have presented best subset selection here for least squares regression, the same ideas apply to other types of models, such as logistic regression. In the case of logistic regression, instead of ordering models by RSS in Step 2 of Algorithm 6.1, we instead use the *deviance*, a measure deviance that plays the role of RSS for a broader class of models. The deviance is negative two times the maximized log-likelihood; the smaller the deviance, the better the fit.

While best subset selection is a simple and conceptually appealing approach, it suffers from computational limitations. The number of possible models that must be considered grows rapidly as p increases. In general, there are 2^p models that involve subsets of p predictors. So if $p = 10$, then there are approximately 1,000 possible models to be considered, and if

$p = 20$, then there are over one million possibilities! Consequently, best subset selection becomes computationally infeasible for values of p greater than around 40, even with extremely fast modern computers. There are computational shortcuts—so called branch-and-bound techniques—for eliminating some choices, but these have their limitations as p gets large. They also only work for least squares linear regression. We present computationally efficient alternatives to best subset selection next.

6.1.2 Stepwise Selection

For computational reasons, best subset selection cannot be applied with very large p. Best subset selection may also suffer from statistical problems when p is large. The larger the search space, the higher the chance of finding models that look good on the training data, even though they might not have any predictive power on future data. Thus an enormous search space can lead to overfitting and high variance of the coefficient estimates.

For both of these reasons, *stepwise* methods, which explore a far more restricted set of models, are attractive alternatives to best subset selection.

Forward Stepwise Selection

Forward stepwise selection is a computationally efficient alternative to best subset selection. While the best subset selection procedure considers all 2^p possible models containing subsets of the p predictors, forward stepwise considers a much smaller set of models. Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model. In particular, at each step the variable that gives the greatest *additional* improvement to the fit is added to the model. More formally, the forward stepwise selection procedure is given in Algorithm 6.2.

forward stepwise selection

Algorithm 6.2 *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p - k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p - k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Unlike best subset selection, which involved fitting 2^p models, forward stepwise selection involves fitting one null model, along with $p - k$ models in the kth iteration, for $k = 0, \ldots, p - 1$. This amounts to a total of $1 + {}_{k=0}^{p-1}(p - k) = 1 + p(p+1)/2$ models. This is a substantial difference: when $p = 20$, best subset selection requires fitting 1,048,576 models, whereas forward stepwise selection requires fitting only 211 models.[1]

In Step 2(b) of Algorithm 6.2, we must identify the *best* model from among those $p - k$ that augment $\mathcal{M}_k$ with one additional predictor. We can do this by simply choosing the model with the lowest RSS or the highest R^2. However, in Step 3, we must identify the best model among a set of models with different numbers of variables. This is more challenging, and is discussed in Section 6.1.3.

Forward stepwise selection's computational advantage over best subset selection is clear. Though forward stepwise tends to do well in practice, it is not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors. For instance, suppose that in a given data set with $p = 3$ predictors, the best possible one-variable model contains X_1, and the best possible two-variable model instead contains X_2 and X_3. Then forward stepwise selection will fail to select the best possible two-variable model, because $\mathcal{M}_1$ will contain X_1, so $\mathcal{M}_2$ must also contain X_1 together with one additional variable.

Table 6.1, which shows the first four selected models for best subset and forward stepwise selection on the `Credit` data set, illustrates this phenomenon. Both best subset selection and forward stepwise selection choose `rating` for the best one-variable model and then include `income` and `student` for the two- and three-variable models. However, best subset selection replaces `rating` by `cards` in the four-variable model, while forward stepwise selection must maintain `rating` in its four-variable model. In this example, Figure 6.1 indicates that there is not much difference between the three- and four-variable models in terms of RSS, so either of the four-variable models will likely be adequate.

Forward stepwise selection can be applied even in the high-dimensional setting where $n < p$; however, in this case, it is possible to construct submodels $\mathcal{M}_0, \ldots, \mathcal{M}_{n-1}$ only, since each submodel is fit using least squares, which will not yield a unique solution if $p \geq n$.

Backward Stepwise Selection

Like forward stepwise selection, *backward stepwise selection* provides an efficient alternative to best subset selection. However, unlike forward

backward stepwise selection

[1]Though forward stepwise selection considers $p(p+1)/2 + 1$ models, it performs a *guided* search over model space, and so the *effective* model space considered contains substantially more than $p(p+1)/2 + 1$ models.

# Variables	Best subset	Forward stepwise
One	`rating`	`rating`
Two	`rating`, `income`	`rating`, `income`
Three	`rating`, `income`, `student`	`rating`, `income`, `student`
Four	`cards`, `income` `student`, `limit`	`rating`, `income`, `student`, `limit`

TABLE 6.1. *The first four selected models for best subset selection and forward stepwise selection on the* `Credit` *data set. The first three models are identical but the fourth models differ.*

stepwise selection, it begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time. Details are given in Algorithm 6.3.

Algorithm 6.3 *Backward stepwise selection*

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Like forward stepwise selection, the backward selection approach searches through only $1+p(p+1)/2$ models, and so can be applied in settings where p is too large to apply best subset selection.[2] Also like forward stepwise selection, backward stepwise selection is not guaranteed to yield the *best* model containing a subset of the p predictors.

Backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit). In contrast, forward stepwise can be used even when $n < p$, and so is the only viable subset method when p is very large.

[2]Like forward stepwise selection, backward stepwise selection performs a *guided* search over model space, and so effectively considers substantially more than $1+p(p+1)/2$ models.

Hybrid Approaches

The best subset, forward stepwise, and backward stepwise selection approaches generally give similar but not identical models. As another alternative, hybrid versions of forward and backward stepwise selection are available, in which variables are added to the model sequentially, in analogy to forward selection. However, after adding each new variable, the method may also remove any variables that no longer provide an improvement in the model fit. Such an approach attempts to more closely mimic best subset selection while retaining the computational advantages of forward and backward stepwise selection.

6.1.3 Choosing the Optimal Model

Best subset selection, forward selection, and backward selection result in the creation of a set of models, each of which contains a subset of the p predictors. In order to implement these methods, we need a way to determine which of these models is *best.* As we discussed in Section 6.1.1, the model containing all of the predictors will always have the smallest RSS and the largest R^2, since these quantities are related to the training error. Instead, we wish to choose a model with a low test error. As is evident here, and as we show in Chapter 2, the training error can be a poor estimate of the test error. Therefore, RSS and R^2 are not suitable for selecting the best model among a collection of models with different numbers of predictors.

In order to select the best model with respect to test error, we need to estimate this test error. There are two common approaches:

1. We can indirectly estimate test error by making an *adjustment* to the training error to account for the bias due to overfitting.

2. We can *directly* estimate the test error, using either a validation set approach or a cross-validation approach, as discussed in Chapter 5.

We consider both of these approaches below.

C_p, AIC, BIC, and Adjusted R^2

We show in Chapter 2 that the training set MSE is generally an underestimate of the test MSE. (Recall that MSE = RSS/n.) This is because when we fit a model to the training data using least squares, we specifically estimate the regression coefficients such that the training RSS (but not the test RSS) is as small as possible. In particular, the training error will decrease as more variables are included in the model, but the test error may not. Therefore, training set RSS and training set R^2 cannot be used to select from among a set of models with different numbers of variables.

However, a number of techniques for *adjusting* the training error for the model size are available. These approaches can be used to select among a set

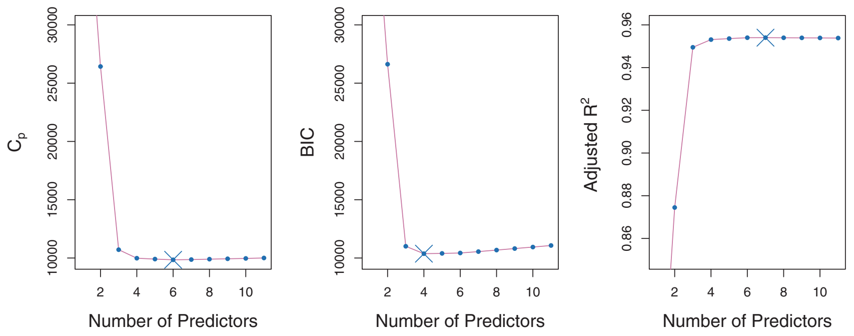

FIGURE 6.2. *C_p, BIC, and adjusted R^2 are shown for the best models of each size for the* `Credit` *data set (the lower frontier in Figure 6.1). C_p and BIC are estimates of test MSE. In the middle plot we see that the BIC estimate of test error shows an increase after four variables are selected. The other two plots are rather flat after four variables are included.*

of models with different numbers of variables. We now consider four such approaches: C_p, *Akaike information criterion* (AIC), *Bayesian information criterion* (BIC), and *adjusted* R^2. Figure 6.2 displays C_p, BIC, and adjusted R^2 for the best model of each size produced by best subset selection on the `Credit` data set.

C_p
Akaike information criterion
Bayesian information criterion
adjusted R^2

For a fitted least squares model containing d predictors, the C_p estimate of test MSE is computed using the equation

$$C_p = \frac{1}{n} \quad \text{RSS} + 2d\hat{\sigma}^2 \quad , \tag{6.2}$$

where $\hat{\sigma}^2$ is an estimate of the variance of the error associated with each response measurement in (6.1).[3] Essentially, the C_p statistic adds a penalty of $2d\hat{\sigma}^2$ to the training RSS in order to adjust for the fact that the training error tends to underestimate the test error. Clearly, the penalty increases as the number of predictors in the model increases; this is intended to adjust for the corresponding decrease in training RSS. Though it is beyond the scope of this book, one can show that if $\hat{\sigma}^2$ is an unbiased estimate of σ^2 in (6.2), then C_p is an unbiased estimate of test MSE. As a consequence, the C_p statistic tends to take on a small value for models with a low test error, so when determining which of a set of models is best, we choose the model with the lowest C_p value. In Figure 6.2, C_p selects the six-variable model containing the predictors `income`, `limit`, `rating`, `cards`, `age` and `student`.

[3]Mallow's C_p is sometimes defined as $C'_p = \text{RSS}/\hat{\sigma}^2 + 2d - n$. This is equivalent to the definition given above in the sense that $C_p = \hat{\sigma}^2(C'_p + n)$, and so the model with smallest C_p also has smallest C'_p.

The AIC criterion is defined for a large class of models fit by maximum likelihood. In the case of the model (6.1) with Gaussian errors, maximum likelihood and least squares are the same thing. In this case AIC is given by

$$\text{AIC} = \frac{1}{n\hat{\sigma}^2} \ \ \text{RSS} + 2d\hat{\sigma}^2 \ \ ,$$

where, for simplicity, we have omitted an additive constant. Hence for least squares models, C_p and AIC are proportional to each other, and so only C_p is displayed in Figure 6.2.

BIC is derived from a Bayesian point of view, but ends up looking similar to C_p (and AIC) as well. For the least squares model with d predictors, the BIC is, up to irrelevant constants, given by

$$\text{BIC} = \frac{1}{n} \ \ \text{RSS} + \log(n)d\hat{\sigma}^2 \ \ . \tag{6.3}$$

Like C_p, the BIC will tend to take on a small value for a model with a low test error, and so generally we select the model that has the lowest BIC value. Notice that BIC replaces the $2d\hat{\sigma}^2$ used by C_p with a $\log(n)d\hat{\sigma}^2$ term, where n is the number of observations. Since $\log n > 2$ for any $n > 7$, the BIC statistic generally places a heavier penalty on models with many variables, and hence results in the selection of smaller models than C_p. In Figure 6.2, we see that this is indeed the case for the `Credit` data set; BIC chooses a model that contains only the four predictors `income`, `limit`, `cards`, and `student`. In this case the curves are very flat and so there does not appear to be much difference in accuracy between the four-variable and six-variable models.

The adjusted R^2 statistic is another popular approach for selecting among a set of models that contain different numbers of variables. Recall from Chapter 3 that the usual R^2 is defined as $1 - \text{RSS}/\text{TSS}$, where TSS = $(y_i - \overline{y})^2$ is the *total sum of squares* for the response. Since RSS always decreases as more variables are added to the model, the R^2 always increases as more variables are added. For a least squares model with d variables, the adjusted R^2 statistic is calculated as

$$\text{Adjusted } R^2 = 1 - \frac{\text{RSS}/(n-d-1)}{\text{TSS}/(n-1)}. \tag{6.4}$$

Unlike C_p, AIC, and BIC, for which a *small* value indicates a model with a low test error, a *large* value of adjusted R^2 indicates a model with a small test error. Maximizing the adjusted R^2 is equivalent to minimizing $\frac{\text{RSS}}{n-d-1}$. While RSS always decreases as the number of variables in the model increases, $\frac{\text{RSS}}{n-d-1}$ may increase or decrease, due to the presence of d in the denominator.

The intuition behind the adjusted R^2 is that once all of the correct variables have been included in the model, adding additional *noise* variables will lead to only a very small decrease in RSS. Since adding noise variables leads to an increase in d, such variables will lead to an increase in $\frac{\text{RSS}}{n-d-1}$, and consequently a decrease in the adjusted R^2. Therefore, in theory, the model with the largest adjusted R^2 will have only correct variables and no noise variables. Unlike the R^2 statistic, the adjusted R^2 statistic *pays a price* for the inclusion of unnecessary variables in the model. Figure 6.2 displays the adjusted R^2 for the `Credit` data set. Using this statistic results in the selection of a model that contains seven variables, adding `gender` to the model selected by C_p and AIC.

C_p, AIC, and BIC all have rigorous theoretical justifications that are beyond the scope of this book. These justifications rely on asymptotic arguments (scenarios where the sample size n is very large). Despite its popularity, and even though it is quite intuitive, the adjusted R^2 is not as well motivated in statistical theory as AIC, BIC, and C_p. All of these measures are simple to use and compute. Here we have presented the formulas for AIC, BIC, and C_p in the case of a linear model fit using least squares; however, these quantities can also be defined for more general types of models.

Validation and Cross-Validation

As an alternative to the approaches just discussed, we can directly estimate the test error using the validation set and cross-validation methods discussed in Chapter 5. We can compute the validation set error or the cross-validation error for each model under consideration, and then select the model for which the resulting estimated test error is smallest. This procedure has an advantage relative to AIC, BIC, C_p, and adjusted R^2, in that it provides a direct estimate of the test error, and makes fewer assumptions about the true underlying model. It can also be used in a wider range of model selection tasks, even in cases where it is hard to pinpoint the model degrees of freedom (e.g. the number of predictors in the model) or hard to estimate the error variance σ^2.

In the past, performing cross-validation was computationally prohibitive for many problems with large p and/or large n, and so AIC, BIC, C_p, and adjusted R^2 were more attractive approaches for choosing among a set of models. However, nowadays with fast computers, the computations required to perform cross-validation are hardly ever an issue. Thus, cross-validation is a very attractive approach for selecting from among a number of models under consideration.

Figure 6.3 displays, as a function of d, the BIC, validation set errors, and cross-validation errors on the `Credit` data, for the best d-variable model. The validation errors were calculated by randomly selecting three-quarters of the observations as the training set, and the remainder as the validation set. The cross-validation errors were computed using $k = 10$ folds. In this case, the validation and cross-validation methods both result in a

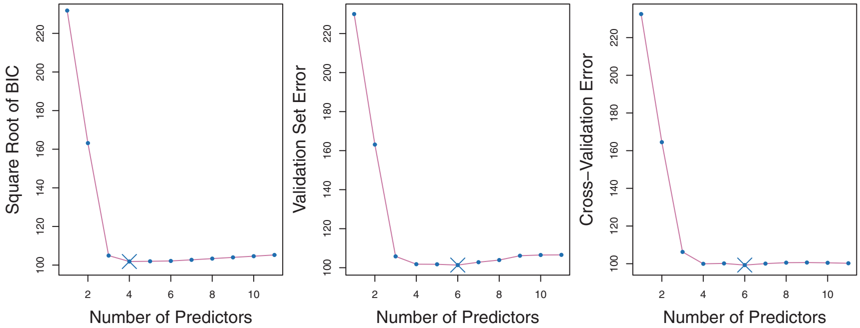

FIGURE 6.3. *For the* `Credit` *data set, three quantities are displayed for the best model containing d predictors, for d ranging from* 1 *to* 11*. The overall* best *model, based on each of these quantities, is shown as a blue cross.* Left: *Square root of BIC.* Center: *Validation set errors.* Right: *Cross-validation errors.*

six-variable model. However, all three approaches suggest that the four-, five-, and six-variable models are roughly equivalent in terms of their test errors.

In fact, the estimated test error curves displayed in the center and right-hand panels of Figure 6.3 are quite flat. While a three-variable model clearly has lower estimated test error than a two-variable model, the estimated test errors of the 3- to 11-variable models are quite similar. Furthermore, if we repeated the validation set approach using a different split of the data into a training set and a validation set, or if we repeated cross-validation using a different set of cross-validation folds, then the precise model with the lowest estimated test error would surely change. In this setting, we can select a model using the *one-standard-error rule*. We first calculate the standard error of the estimated test MSE for each model size, and then select the smallest model for which the estimated test error is within one standard error of the lowest point on the curve. The rationale here is that if a set of models appear to be more or less equally good, then we might as well choose the simplest model—that is, the model with the smallest number of predictors. In this case, applying the one-standard-error rule to the validation set or cross-validation approach leads to selection of the three-variable model.

one-standard-error rule

6.2 Shrinkage Methods

The subset selection methods described in Section 6.1 involve using least squares to fit a linear model that contains a subset of the predictors. As an alternative, we can fit a model containing all p predictors using a technique that *constrains* or *regularizes* the coefficient estimates, or equivalently, that *shrinks* the coefficient estimates towards zero. It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance. The two best-known techniques for shrinking the regression coefficients towards zero are *ridge regression* and the *lasso*.

6.2.1 Ridge Regression

Recall from Chapter 3 that the least squares fitting procedure estimates $\beta_0, \beta_1, \ldots, \beta_p$ using the values that minimize

$$\text{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity. In particular, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

ridge regression

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2, \tag{6.5}$$

where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately. Equation 6.5 trades off two different criteria. As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small. However, the second term, $\lambda \sum_j \beta_j^2$, called a *shrinkage penalty*, is small when $\beta_1, \ldots, \beta_p$ are close to zero, and so it has the effect of *shrinking* the estimates of β_j towards zero. The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates. When $\lambda = 0$, the penalty term has no effect, and ridge regression will produce the least squares estimates. However, as $\lambda \to \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero. Unlike least squares, which generates only one set of coefficient estimates, ridge regression will produce a different set of coefficient estimates, $\hat{\beta}^R_\lambda$, for each value of λ. Selecting a good value for λ is critical; we defer this discussion to Section 6.2.3, where we use cross-validation.

tuning parameter

shrinkage penalty

Note that in (6.5), the shrinkage penalty is applied to $\beta_1, \ldots, \beta_p$, but not to the intercept β_0. We want to shrink the estimated association of each variable with the response; however, we do not want to shrink the intercept, which is simply a measure of the mean value of the response when $x_{i1} = x_{i2} = \ldots = x_{ip} = 0$. If we assume that the variables—that is, the columns of the data matrix **X**—have been centered to have mean zero before ridge regression is performed, then the estimated intercept will take the form $\hat{\beta}_0 = \bar{y} = \sum_{i=1}^{n} y_i/n$.

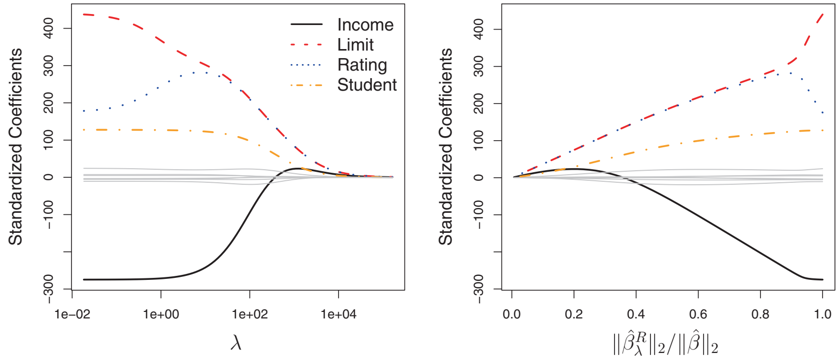

FIGURE 6.4. *The standardized ridge regression coefficients are displayed for the* `Credit` *data set, as a function of* λ *and* $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$.

An Application to the Credit Data

In Figure 6.4, the ridge regression coefficient estimates for the `Credit` data set are displayed. In the left-hand panel, each curve corresponds to the ridge regression coefficient estimate for one of the ten variables, plotted as a function of λ. For example, the black solid line represents the ridge regression estimate for the `income` coefficient, as λ is varied. At the extreme left-hand side of the plot, λ is essentially zero, and so the corresponding ridge coefficient estimates are the same as the usual least squares estimates. But as λ increases, the ridge coefficient estimates shrink towards zero. When λ is extremely large, then all of the ridge coefficient estimates are basically zero; this corresponds to the *null model* that contains no predictors. In this plot, the `income`, `limit`, `rating`, and `student` variables are displayed in distinct colors, since these variables tend to have by far the largest coefficient estimates. While the ridge coefficient estimates tend to decrease in aggregate as λ increases, individual coefficients, such as `rating` and `income`, may occasionally increase as λ increases.

The right-hand panel of Figure 6.4 displays the same ridge coefficient estimates as the left-hand panel, but instead of displaying λ on the x-axis, we now display $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$, where $\hat{\beta}$ denotes the vector of least squares coefficient estimates. The notation $\|\beta\|_2$ denotes the ℓ_2 *norm* (pronounced "ell 2") of a vector, and is defined as $\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} {\beta_j}^2}$. It measures the distance of β from zero. As λ increases, the ℓ_2 norm of $\hat{\beta}^R_\lambda$ will *always* decrease, and so will $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$. The latter quantity ranges from 1 (when $\lambda = 0$, in which case the ridge regression coefficient estimate is the same as the least squares estimate, and so their ℓ_2 norms are the same) to 0 (when $\lambda = \infty$, in which case the ridge regression coefficient estimate is a vector of zeros, with ℓ_2 norm equal to zero). Therefore, we can think of the x-axis in the right-hand panel of Figure 6.4 as the amount that the ridge

ℓ_2 norm

regression coefficient estimates have been shrunken towards zero; a small value indicates that they have been shrunken very close to zero.

The standard least squares coefficient estimates discussed in Chapter 3 are *scale invariant*: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same. In contrast, the ridge regression coefficient estimates can change *substantially* when multiplying a given predictor by a constant. For instance, consider the `income` variable, which is measured in dollars. One could reasonably have measured income in thousands of dollars, which would result in a reduction in the observed values of `income` by a factor of 1,000. Now due to the sum of squared coefficients term in the ridge regression formulation (6.5), such a change in scale will not simply cause the ridge regression coefficient estimate for `income` to change by a factor of 1,000. In other words, $X_j\hat{\beta}^R_{j,\lambda}$ will depend not only on the value of λ, but also on the scaling of the jth predictor. In fact, the value of $X_j\hat{\beta}^R_{j,\lambda}$ may even depend on the scaling of the *other* predictors! Therefore, it is best to apply ridge regression after *standardizing the predictors*, using the formula

scale invariant

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}}, \qquad (6.6)$$

so that they are all on the same scale. In (6.6), the denominator is the estimated standard deviation of the jth predictor. Consequently, all of the standardized predictors will have a standard deviation of one. As a result the final fit will not depend on the scale on which the predictors are measured. In Figure 6.4, the y-axis displays the standardized ridge regression coefficient estimates—that is, the coefficient estimates that result from performing ridge regression using standardized predictors.

Why Does Ridge Regression Improve Over Least Squares?

Ridge regression's advantage over least squares is rooted in the *bias-variance trade-off*. As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias. This is illustrated in the left-hand panel of Figure 6.5, using a simulated data set containing $p = 45$ predictors and $n = 50$ observations. The green curve in the left-hand panel of Figure 6.5 displays the variance of the ridge regression predictions as a function of λ. At the least squares coefficient estimates, which correspond to ridge regression with $\lambda = 0$, the variance is high but there is no bias. But as λ increases, the shrinkage of the ridge coefficient estimates leads to a substantial reduction in the variance of the predictions, at the expense of a slight increase in bias. Recall that the test mean squared error (MSE), plotted in purple, is a function of the variance plus the squared bias. For values

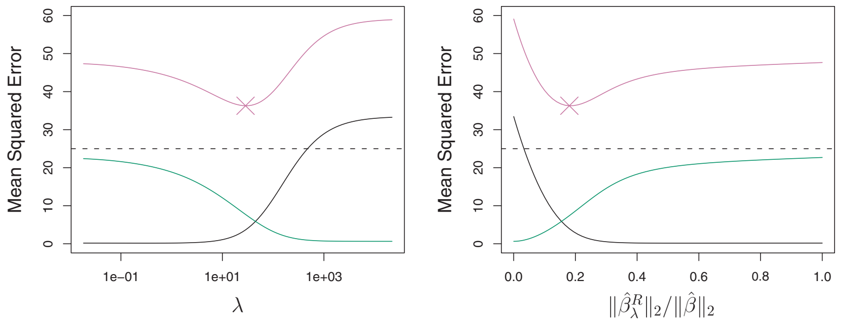

FIGURE 6.5. *Squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set, as a function of λ and $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$. The horizontal dashed lines indicate the minimum possible MSE. The purple crosses indicate the ridge regression models for which the MSE is smallest.*

of λ up to about 10, the variance decreases rapidly, with very little increase in bias, plotted in black. Consequently, the MSE drops considerably as λ increases from 0 to 10. Beyond this point, the decrease in variance due to increasing λ slows, and the shrinkage on the coefficients causes them to be significantly underestimated, resulting in a large increase in the bias. The minimum MSE is achieved at approximately $\lambda = 30$. Interestingly, because of its high variance, the MSE associated with the least squares fit, when $\lambda = 0$, is almost as high as that of the null model for which all coefficient estimates are zero, when $\lambda = \infty$. However, for an intermediate value of λ, the MSE is considerably lower.

The right-hand panel of Figure 6.5 displays the same curves as the left-hand panel, this time plotted against the ℓ_2 norm of the ridge regression coefficient estimates divided by the ℓ_2 norm of the least squares estimates. Now as we move from left to right, the fits become more flexible, and so the bias decreases and the variance increases.

In general, in situations where the relationship between the response and the predictors is close to linear, the least squares estimates will have low bias but may have high variance. This means that a small change in the training data can cause a large change in the least squares coefficient estimates. In particular, when the number of variables p is almost as large as the number of observations n, as in the example in Figure 6.5, the least squares estimates will be extremely variable. And if $p > n$, then the least squares estimates do not even have a unique solution, whereas ridge regression can still perform well by trading off a small increase in bias for a large decrease in variance. Hence, ridge regression works best in situations where the least squares estimates have high variance.

Ridge regression also has substantial computational advantages over best subset selection, which requires searching through 2^p models. As we discussed previously, even for moderate values of p, such a search can be computationally infeasible. In contrast, for any fixed value of λ, ridge regression only fits a single model, and the model-fitting procedure can be performed quite quickly. In fact, one can show that the computations required to solve (6.5), *simultaneously for all values of* λ, are almost identical to those for fitting a model using least squares.

6.2.2 The Lasso

Ridge regression does have one obvious disadvantage. Unlike best subset, forward stepwise, and backward stepwise selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model. The penalty $\lambda \sum \beta_j^2$ in (6.5) will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero (unless $\lambda = \infty$). This may not be a problem for prediction accuracy, but it can create a challenge in model interpretation in settings in which the number of variables p is quite large. For example, in the `Credit` data set, it appears that the most important variables are `income`, `limit`, `rating`, and `student`. So we might wish to build a model including just these predictors. However, ridge regression will always generate a model involving all ten predictors. Increasing the value of λ will tend to reduce the magnitudes of the coefficients, but will not result in exclusion of any of the variables.

The *lasso* is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients, $\hat{\beta}_\lambda^L$, minimize the quantity lasso

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|. \qquad (6.7)$$

Comparing (6.7) to (6.5), we see that the lasso and ridge regression have similar formulations. The only difference is that the β_j^2 term in the ridge regression penalty (6.5) has been replaced by $|\beta_j|$ in the lasso penalty (6.7). In statistical parlance, the lasso uses an ℓ_1 (pronounced "ell 1") penalty instead of an ℓ_2 penalty. The ℓ_1 norm of a coefficient vector β is given by $\|\beta\|_1 = \sum |\beta_j|$.

As with ridge regression, the lasso shrinks the coefficient estimates towards zero. However, in the case of the lasso, the ℓ_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large. Hence, much like best subset selection, the lasso performs *variable selection*. As a result, models generated from the lasso are generally much easier to interpret than those produced by ridge regression. We say that the lasso yields *sparse* models—that is, sparse models that involve only a subset of the variables. As in ridge regression, selecting a good value of λ for the lasso is critical; we defer this discussion to Section 6.2.3, where we use cross-validation.

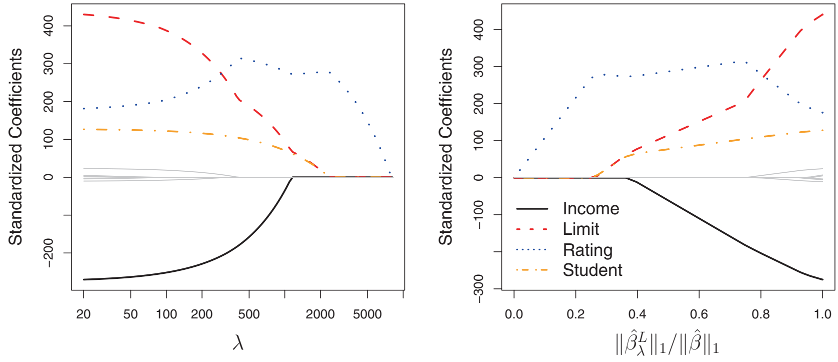

FIGURE 6.6. *The standardized lasso coefficients on the* `Credit` *data set are shown as a function of λ and $\|\hat{\beta}^L_\lambda\|_1/\|\hat{\beta}\|_1$.*

As an example, consider the coefficient plots in Figure 6.6, which are generated from applying the lasso to the `Credit` data set. When $\lambda = 0$, then the lasso simply gives the least squares fit, and when λ becomes sufficiently large, the lasso gives the null model in which all coefficient estimates equal zero. However, in between these two extremes, the ridge regression and lasso models are quite different from each other. Moving from left to right in the right-hand panel of Figure 6.6, we observe that at first the lasso results in a model that contains only the `rating` predictor. Then `student` and `limit` enter the model almost simultaneously, shortly followed by `income`. Eventually, the remaining variables enter the model. Hence, depending on the value of λ, the lasso can produce a model involving any number of variables. In contrast, ridge regression will always include all of the variables in the model, although the magnitude of the coefficient estimates will depend on λ.

Another Formulation for Ridge Regression and the Lasso

One can show that the lasso and ridge regression coefficient estimates solve the problems

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq s \tag{6.8}$$

and

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq s, \tag{6.9}$$

respectively. In other words, for every value of λ, there is some s such that the Equations (6.7) and (6.8) will give the same lasso coefficient estimates. Similarly, for every value of λ there is a corresponding s such that Equations (6.5) and (6.9) will give the same ridge regression coefficient estimates. When $p = 2$, then (6.8) indicates that the lasso coefficient estimates have the smallest RSS out of all points that lie within the diamond defined by $|\beta_1| + |\beta_2| \leq s$. Similarly, the ridge regression estimates have the smallest RSS out of all points that lie within the circle defined by $\beta_1^2 + \beta_2^2 \leq s$.

We can think of (6.8) as follows. When we perform the lasso we are trying to find the set of coefficient estimates that lead to the smallest RSS, subject to the constraint that there is a *budget* s for how large $\sum_{j=1}^{p} |\beta_j|$ can be. When s is extremely large, then this budget is not very restrictive, and so the coefficient estimates can be large. In fact, if s is large enough that the least squares solution falls within the budget, then (6.8) will simply yield the least squares solution. In contrast, if s is small, then $\sum_{j=1}^{p} |\beta_j|$ must be small in order to avoid violating the budget. Similarly, (6.9) indicates that when we perform ridge regression, we seek a set of coefficient estimates such that the RSS is as small as possible, subject to the requirement that $\sum_{j=1}^{p} \beta_j^2$ not exceed the budget s.

The formulations (6.8) and (6.9) reveal a close connection between the lasso, ridge regression, and best subset selection. Consider the problem

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} I(\beta_j \neq 0) \leq s. \tag{6.10}$$

Here $I(\beta_j \neq 0)$ is an indicator variable: it takes on a value of 1 if $\beta_j \neq 0$, and equals zero otherwise. Then (6.10) amounts to finding a set of coefficient estimates such that RSS is as small as possible, subject to the constraint that no more than s coefficients can be nonzero. The problem (6.10) is equivalent to best subset selection. Unfortunately, solving (6.10) is computationally infeasible when p is large, since it requires considering all $\binom{p}{s}$ models containing s predictors. Therefore, we can interpret ridge regression and the lasso as computationally feasible alternatives to best subset selection that replace the intractable form of the budget in (6.10) with forms that are much easier to solve. Of course, the lasso is much more closely related to best subset selection, since only the lasso performs feature selection for s sufficiently small in (6.8).

The Variable Selection Property of the Lasso

Why is it that the lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero? The formulations (6.8) and (6.9) can be used to shed light on the issue. Figure 6.7 illustrates the situation. The least squares solution is marked as $\hat{\beta}$, while the blue diamond and

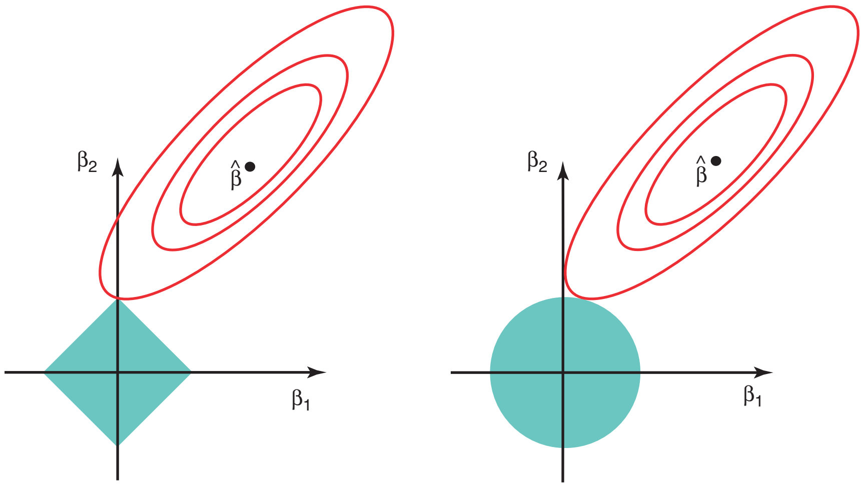

FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.*

circle represent the lasso and ridge regression constraints in (6.8) and (6.9), respectively. If s is sufficiently large, then the constraint regions will contain $\hat{\beta}$, and so the ridge regression and lasso estimates will be the same as the least squares estimates. (Such a large value of s corresponds to $\lambda = 0$ in (6.5) and (6.7).) However, in Figure 6.7 the least squares estimates lie outside of the diamond and the circle, and so the least squares estimates are not the same as the lasso and ridge regression estimates.

The ellipses that are centered around $\hat{\beta}$ represent regions of constant RSS. In other words, all of the points on a given ellipse share a common value of the RSS. As the ellipses expand away from the least squares coefficient estimates, the RSS increases. Equations (6.8) and (6.9) indicate that the lasso and ridge regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region. Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero. However, the lasso constraint has *corners* at each of the axes, and so the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero. In higher dimensions, many of the coefficient estimates may equal zero simultaneously. In Figure 6.7, the intersection occurs at $\beta_1 = 0$, and so the resulting model will only include β_2.

In Figure 6.7, we considered the simple case of $p = 2$. When $p = 3$, then the constraint region for ridge regression becomes a sphere, and the constraint region for the lasso becomes a polyhedron. When $p > 3$, the

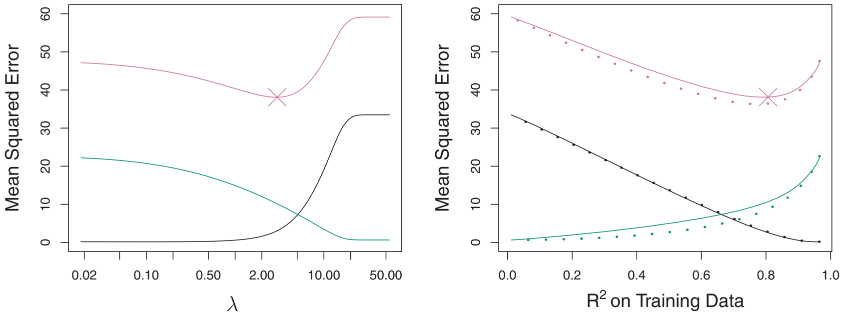

FIGURE 6.8. Left: *Plots of squared bias (black), variance (green), and test MSE (purple) for the lasso on a simulated data set.* Right: *Comparison of squared bias, variance and test MSE between lasso (solid) and ridge (dashed). Both are plotted against their R^2 on the training data, as a common form of indexing. The crosses in both plots indicate the lasso model for which the MSE is smallest.*

constraint for ridge regression becomes a hypersphere, and the constraint for the lasso becomes a polytope. However, the key ideas depicted in Figure 6.7 still hold. In particular, the lasso leads to feature selection when $p > 2$ due to the sharp corners of the polyhedron or polytope.

Comparing the Lasso and Ridge Regression

It is clear that the lasso has a major advantage over ridge regression, in that it produces simpler and more interpretable models that involve only a subset of the predictors. However, which method leads to better prediction accuracy? Figure 6.8 displays the variance, squared bias, and test MSE of the lasso applied to the same simulated data as in Figure 6.5. Clearly the lasso leads to qualitatively similar behavior to ridge regression, in that as λ increases, the variance decreases and the bias increases. In the right-hand panel of Figure 6.8, the dotted lines represent the ridge regression fits. Here we plot both against their R^2 on the training data. This is another useful way to index models, and can be used to compare models with different types of regularization, as is the case here. In this example, the lasso and ridge regression result in almost identical biases. However, the variance of ridge regression is slightly lower than the variance of the lasso. Consequently, the minimum MSE of ridge regression is slightly smaller than that of the lasso.

However, the data in Figure 6.8 were generated in such a way that all 45 predictors were related to the response—that is, none of the true coefficients $\beta_1, \dots, \beta_{45}$ equaled zero. The lasso implicitly assumes that a number of the coefficients truly equal zero. Consequently, it is not surprising that ridge regression outperforms the lasso in terms of prediction error in this setting. Figure 6.9 illustrates a similar situation, except that now the response is a

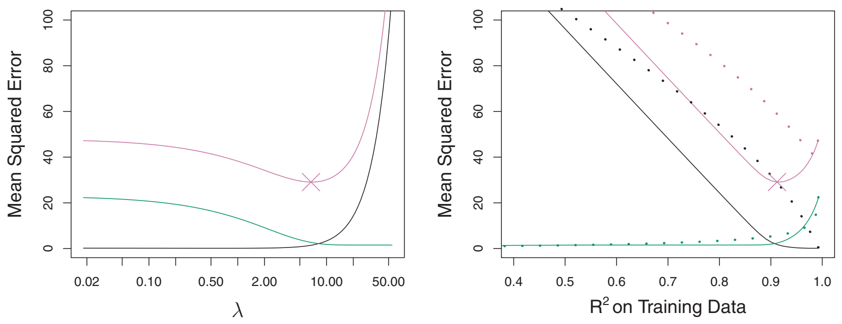

FIGURE 6.9. Left: *Plots of squared bias (black), variance (green), and test MSE (purple) for the lasso. The simulated data is similar to that in Figure 6.8, except that now only two predictors are related to the response.* Right: *Comparison of squared bias, variance and test MSE between lasso (solid) and ridge (dashed). Both are plotted against their R^2 on the training data, as a common form of indexing. The crosses in both plots indicate the lasso model for which the MSE is smallest.*

function of only 2 out of 45 predictors. Now the lasso tends to outperform ridge regression in terms of bias, variance, and MSE.

These two examples illustrate that neither ridge regression nor the lasso will universally dominate the other. In general, one might expect the lasso to perform better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero. Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size. However, the number of predictors that is related to the response is never known *a priori* for real data sets. A technique such as cross-validation can be used in order to determine which approach is better on a particular data set.

As with ridge regression, when the least squares estimates have excessively high variance, the lasso solution can yield a reduction in variance at the expense of a small increase in bias, and consequently can generate more accurate predictions. Unlike ridge regression, the lasso performs variable selection, and hence results in models that are easier to interpret.

There are very efficient algorithms for fitting both ridge and lasso models; in both cases the entire coefficient paths can be computed with about the same amount of work as a single least squares fit. We will explore this further in the lab at the end of this chapter.

A Simple Special Case for Ridge Regression and the Lasso

In order to obtain a better intuition about the behavior of ridge regression and the lasso, consider a simple special case with $n = p$, and $\mathbf{X}$ a diagonal matrix with 1's on the diagonal and 0's in all off-diagonal elements.

To simplify the problem further, assume also that we are performing regression without an intercept. With these assumptions, the usual least squares problem simplifies to finding $\beta_1, \ldots, \beta_p$ that minimize

$$\sum_{j=1}^{p}(y_j - \beta_j)^2. \tag{6.11}$$

In this case, the least squares solution is given by

$$\hat{\beta}_j = y_j.$$

And in this setting, ridge regression amounts to finding $\beta_1, \ldots, \beta_p$ such that

$$\sum_{j=1}^{p}(y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{6.12}$$

is minimized, and the lasso amounts to finding the coefficients such that

$$\sum_{j=1}^{p}(y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{6.13}$$

is minimized. One can show that in this setting, the ridge regression estimates take the form

$$\hat{\beta}_j^R = y_j/(1 + \lambda), \tag{6.14}$$

and the lasso estimates take the form

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & \text{if } y_j > \lambda/2; \\ y_j + \lambda/2 & \text{if } y_j < -\lambda/2; \\ 0 & \text{if } |y_j| \leq \lambda/2. \end{cases} \tag{6.15}$$

Figure 6.10 displays the situation. We can see that ridge regression and the lasso perform two very different types of shrinkage. In ridge regression, each least squares coefficient estimate is shrunken by the same proportion. In contrast, the lasso shrinks each least squares coefficient towards zero by a constant amount, $\lambda/2$; the least squares coefficients that are less than $\lambda/2$ in absolute value are shrunken entirely to zero. The type of shrinkage performed by the lasso in this simple setting (6.15) is known as *soft-thresholding*. The fact that some lasso coefficients are shrunken entirely to zero explains why the lasso performs feature selection. soft-thresholding

In the case of a more general data matrix **X**, the story is a little more complicated than what is depicted in Figure 6.10, but the main ideas still hold approximately: ridge regression more or less shrinks every dimension of the data by the same proportion, whereas the lasso more or less shrinks all coefficients toward zero by a similar amount, and sufficiently small coefficients are shrunken all the way to zero.

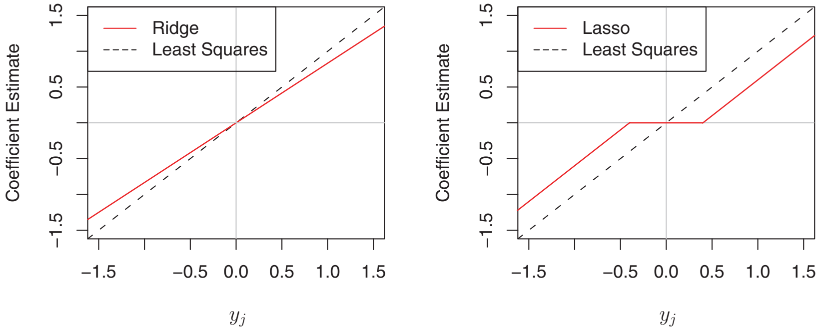

FIGURE 6.10. *The ridge regression and lasso coefficient estimates for a simple setting with $n = p$ and $\mathbf{X}$ a diagonal matrix with 1's on the diagonal.* Left: *The ridge regression coefficient estimates are shrunken proportionally towards zero, relative to the least squares estimates.* Right: *The lasso coefficient estimates are soft-thresholded towards zero.*

Bayesian Interpretation for Ridge Regression and the Lasso

We now show that one can view ridge regression and the lasso through a Bayesian lens. A Bayesian viewpoint for regression assumes that the coefficient vector β has some *prior* distribution, say $p(\beta)$, where $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^T$. The likelihood of the data can be written as $f(Y|X, \beta)$, where $X = (X_1, \ldots, X_p)$. Multiplying the prior distribution by the likelihood gives us (up to a proportionality constant) the *posterior distribution*, which takes the form

posterior distribution

$$p(\beta|X, Y) \propto f(Y|X, \beta)p(\beta|X) = f(Y|X, \beta)p(\beta),$$

where the proportionality above follows from Bayes' theorem, and the equality above follows from the assumption that X is fixed.

We assume the usual linear model,

$$Y = \beta_0 + X_1\beta_1 + \ldots + X_p\beta_p + \epsilon,$$

and suppose that the errors are independent and drawn from a normal distribution. Furthermore, assume that $p(\beta) = {}_{j=1}^{p} g(\beta_j)$, for some density function g. It turns out that ridge regression and the lasso follow naturally from two special cases of g:

- If g is a Gaussian distribution with mean zero and standard deviation a function of λ, then it follows that the *posterior mode* for β—that is, the most likely value for β, given the data—is given by the ridge regression solution. (In fact, the ridge regression solution is also the posterior mean.)

posterior mode

8
Tree-Based Methods

In this chapter, we describe *tree-based* methods for regression and classification. These involve *stratifying* or *segmenting* the predictor space into a number of simple regions. In order to make a prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision tree* methods.

decision tree

Tree-based methods are simple and useful for interpretation. However, they typically are not competitive with the best supervised learning approaches, such as those seen in Chapters 6 and 7, in terms of prediction accuracy. Hence in this chapter we also introduce *bagging*, *random forests*, and *boosting*. Each of these approaches involves producing multiple trees which are then combined to yield a single consensus prediction. We will see that combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation.

8.1 The Basics of Decision Trees

Decision trees can be applied to both regression and classification problems. We first consider regression problems, and then move on to classification.

G. James et al., *An Introduction to Statistical Learning: with Applications in R*, Springer Texts in Statistics 103, DOI 10.1007/978-1-4614-7138-7_8,

FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* `Years<4.5`*, and the right-hand branch corresponds to* `Years>=4.5`*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

8.1.1 Regression Trees

In order to motivate *regression trees*, we begin with a simple example. regression tree

Predicting Baseball Players' Salaries Using Regression Trees

We use the `Hitters` data set to predict a baseball player's `Salary` based on `Years` (the number of years that he has played in the major leagues) and `Hits` (the number of hits that he made in the previous year). We first remove observations that are missing `Salary` values, and log-transform `Salary` so that its distribution has more of a typical bell-shape. (Recall that `Salary` is measured in thousands of dollars.)

Figure 8.1 shows a regression tree fit to this data. It consists of a series of splitting rules, starting at the top of the tree. The top split assigns observations having `Years<4.5` to the left branch.[1] The predicted salary

[1] Both `Years` and `Hits` are integers in these data; the `tree()` function in `R` labels the splits at the midpoint between two adjacent values.

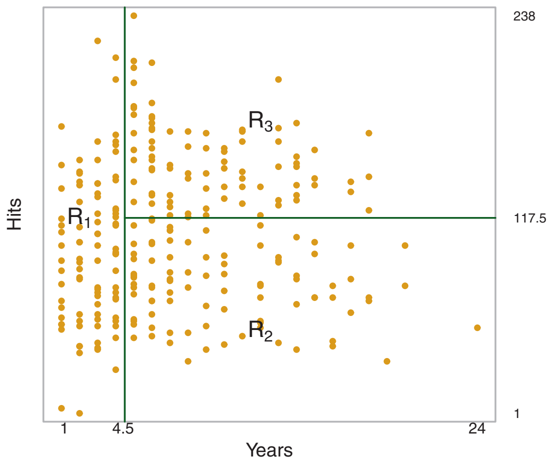

FIGURE 8.2. *The three-region partition for the* `Hitters` *data set from the regression tree illustrated in Figure 8.1.*

for these players is given by the mean response value for the players in the data set with `Years<4.5`. For such players, the mean log salary is 5.107, and so we make a prediction of $e^{5.107}$ thousands of dollars, i.e. \$165,174, for these players. Players with `Years>=4.5` are assigned to the right branch, and then that group is further subdivided by `Hits`. Overall, the tree stratifies or segments the players into three regions of predictor space: players who have played for four or fewer years, players who have played for five or more years and who made fewer than 118 hits last year, and players who have played for five or more years and who made at least 118 hits last year. These three regions can be written as $R_1 = \{X \mid$ `Years<4.5`$\}$, $R_2 = \{X \mid$ `Years>=4.5`, `Hits<117.5`$\}$, and $R_3 = \{X \mid$ `Years>=4.5`, `Hits>=117.5`$\}$. Figure 8.2 illustrates the regions as a function of `Years` and `Hits`. The predicted salaries for these three groups are $\$1{,}000 \times e^{5.107} = \$165{,}174$, $\$1{,}000 \times e^{5.999} = \$402{,}834$, and $\$1{,}000 \times e^{6.740} = \$845{,}346$ respectively.

In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes* or *leaves* of the tree. As is the case for Figure 8.1, decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree. The points along the tree where the predictor space is split are referred to as *internal nodes*. In Figure 8.1, the two internal nodes are indicated by the text `Years<4.5` and `Hits<117.5`. We refer to the segments of the trees that connect the nodes as *branches*.

terminal node

leaf

internal node

branch

We might interpret the regression tree displayed in Figure 8.1 as follows: `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players. Given that a player is less experienced, the number of hits that he made in the previous year seems to play little role in his salary. But among players who have been in the major leagues for five or more years, the number of hits made in the previous year does affect salary, and players who made more hits last year tend to have higher salaries. The regression tree shown in Figure 8.1 is likely an over-simplification of the true relationship between `Hits`, `Years`, and `Salary`. However, it has advantages over other types of regression models (such as those seen in Chapters 3 and 6): it is easier to interpret, and has a nice graphical representation.

Prediction via Stratification of the Feature Space

We now discuss the process of building a regression tree. Roughly speaking, there are two steps.

1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

For instance, suppose that in Step 1 we obtain two regions, R_1 and R_2, and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20. Then for a given observation $X = x$, if $x \in R_1$ we will predict a value of 10, and if $x \in R_2$ we will predict a value of 20.

We now elaborate on Step 1 above. How do we construct the regions $R_1, \ldots, R_J$? In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \tag{8.1}$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box. Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*. The approach is *top-down* because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree. It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

recursive binary splitting

In order to perform recursive binary splitting, we first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS. (The notation $\{X|X_j < s\}$ means *the region of predictor space in which X_j takes on a value less than s*.) That is, we consider all predictors $X_1, \ldots, X_p$, and all possible values of the cutpoint s for each of the predictors, and then choose the predictor and cutpoint such that the resulting tree has the lowest RSS. In greater detail, for any j and s, we define the pair of half-planes

$$R_1(j, s) = \{X|X_j < s\} \text{ and } R_2(j, s) = \{X|X_j \geq s\}, \tag{8.2}$$

and we seek the value of j and s that minimize the equation

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2, \tag{8.3}$$

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j, s)$, and $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j, s)$. Finding the values of j and s that minimize (8.3) can be done quite quickly, especially when the number of features p is not too large.

Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions. However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions. Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Once the regions $R_1, \ldots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

A five-region example of this approach is shown in Figure 8.3.

Tree Pruning

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. This is because the resulting tree might be too complex. A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias. One possible alternative to the process described above is to build the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold. This strategy will result in smaller trees, but is too short-sighted since a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.

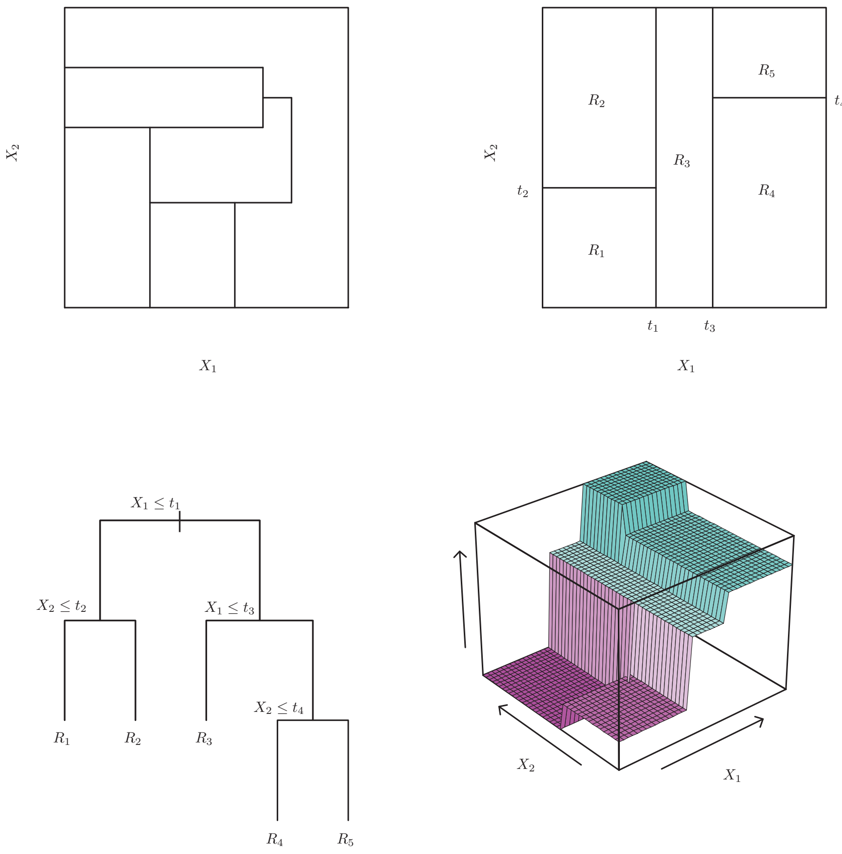

FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree.*

Therefore, a better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*. How do we determine the best way to prune the tree? Intuitively, our goal is to select a subtree that leads to the lowest test error rate. Given a subtree, we can estimate its test error using cross-validation or the validation set approach. However, estimating the cross-validation error for every possible subtree would be too cumbersome, since there is an extremely large number of possible subtrees. Instead, we need a way to select a small set of subtrees for consideration.

prune

subtree

Cost complexity pruning—also known as *weakest link pruning*—gives us a way to do just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.

cost complexity pruning

weakest link pruning

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{8.4}$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the predicted response associated with R_m—that is, the mean of the training observations in R_m. The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0, because then (8.4) just measures the training error. However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity (8.4) will tend to be minimized for a smaller subtree. Equation 8.4 is reminiscent of the lasso (6.7) from Chapter 6, in which a similar formulation was used in order to control the complexity of a linear model.

It turns out that as we increase α from zero in (8.4), branches get pruned from the tree in a nested and predictable fashion, so obtaining the whole sequence of subtrees as a function of α is easy. We can select a value of α using a validation set or using cross-validation. We then return to the full data set and obtain the subtree corresponding to α. This process is summarized in Algorithm 8.1.

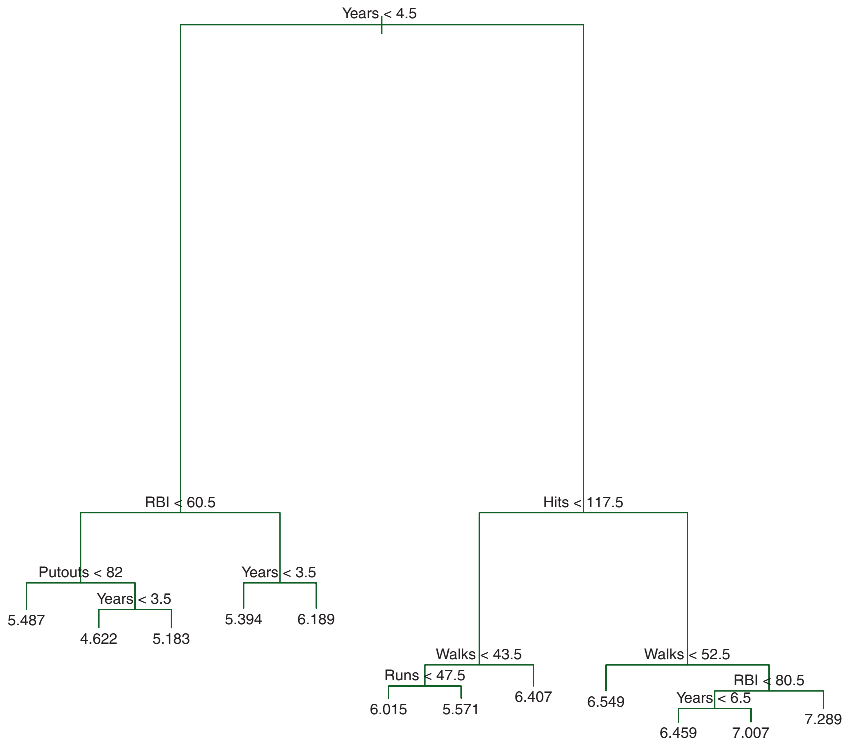

FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

Figures 8.4 and 8.5 display the results of fitting and pruning a regression tree on the `Hitters` data, using nine of the features. First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set. We then built a large regression tree on the training data and varied α in (8.4) in order to create subtrees with different numbers of terminal nodes. Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of α. (We chose to perform six-fold cross-validation because 132 is an exact multiple of six.) The unpruned regression tree is shown in Figure 8.4. The green curve in Figure 8.5 shows the CV error as a function of the number of leaves,[2] while the orange curve indicates the test error. Also shown are standard error bars around the estimated errors. For reference, the training error curve is shown in black. The CV error is a reasonable approximation of the test error: the CV error takes on its

[2] Although CV error is computed as a function of α, it is convenient to display the result as a function of $|T|$, the number of leaves; this is based on the relationship between α and $|T|$ in the original tree grown to all the training data.

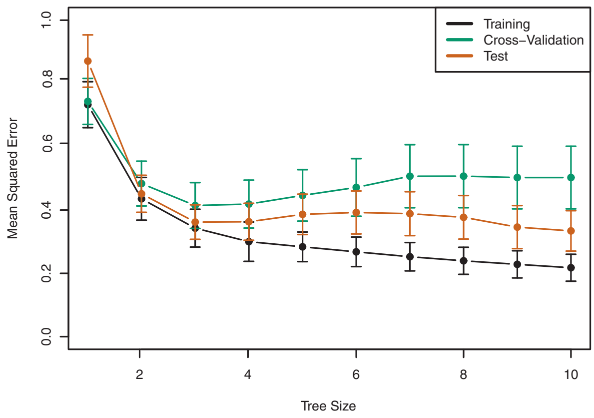

FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

minimum for a three-node tree, while the test error also dips down at the three-node tree (though it takes on its lowest value at the ten-node tree). The pruned tree containing three terminal nodes is shown in Figure 8.1.

8.1.2 Classification Trees

A *classification tree* is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one. Recall that for a regression tree, the predicted response for an observation is given by the mean response of the training observations that belong to the same terminal node. In contrast, for a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs. In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the *class proportions* among the training observations that fall into that region.

classification tree

The task of growing a classification tree is quite similar to the task of growing a regression tree. Just as in the regression setting, we use recursive binary splitting to grow a classification tree. However, in the classification setting, RSS cannot be used as a criterion for making the binary splits. A natural alternative to RSS is the *classification error rate*. Since we plan to assign an observation in a given region to the *most commonly occurring class* of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

classification error rate

$$E = 1 - \max_k(\hat{p}_{mk}). \tag{8.5}$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

The *Gini index* is defined by

Gini index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{8.6}$$

a measure of total variance across the K classes. It is not hard to see that the Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node *purity*—a small value indicates that a node contains predominantly observations from a single class.

An alternative to the Gini index is *cross-entropy*, given by

cross-entropy

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \tag{8.7}$$

Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$. One can show that the cross-entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one. Therefore, like the Gini index, the cross-entropy will take on a small value if the mth node is pure. In fact, it turns out that the Gini index and the cross-entropy are quite similar numerically.

When building a classification tree, either the Gini index or the cross-entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate. Any of these three approaches might be used when *pruning* the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

Figure 8.6 shows an example on the `Heart` data set. These data contain a binary outcome `HD` for 303 patients who presented with chest pain. An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while `No` means no heart disease. There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements. Cross-validation results in a tree with six terminal nodes.

In our discussion thus far, we have assumed that the predictor variables take on continuous values. However, decision trees can be constructed even in the presence of qualitative predictor variables. For instance, in the `Heart` data, some of the predictors, such as `Sex`, `Thal` (Thalium stress test), and `ChestPain`, are qualitative. Therefore, a split on one of these variables amounts to assigning some of the qualitative values to one branch and

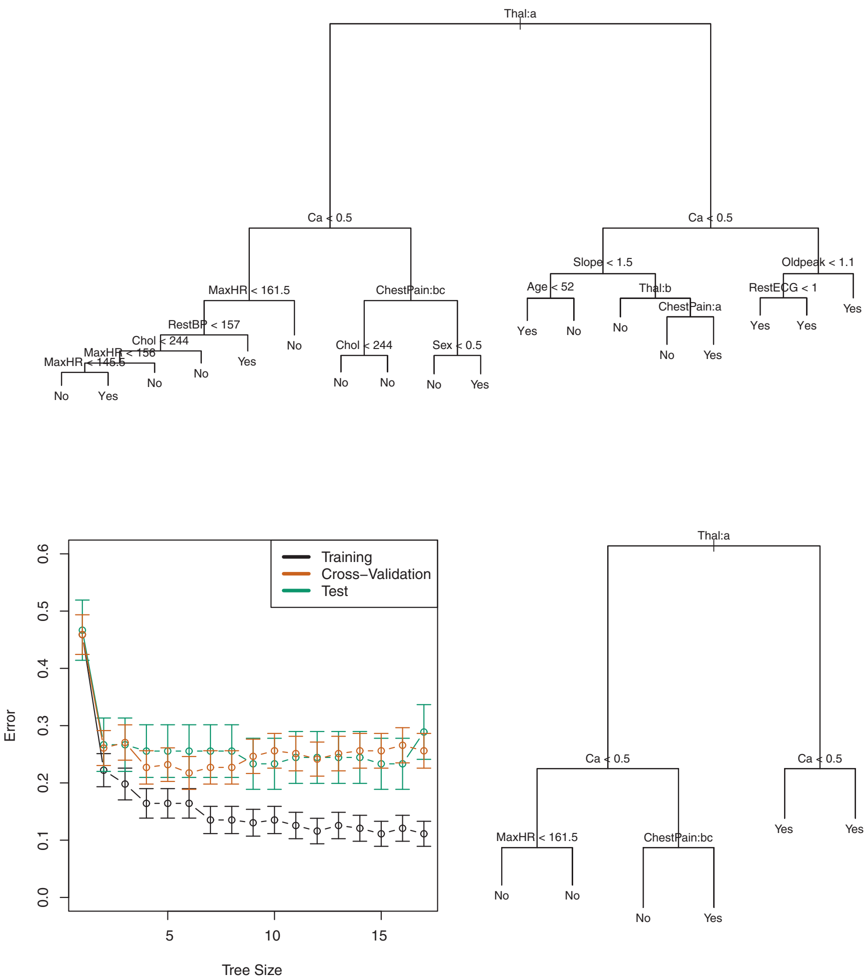

FIGURE 8.6. `Heart` *data.* Top: *The unpruned tree.* Bottom Left: *Cross -validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*

assigning the remaining to the other branch. In Figure 8.6, some of the internal nodes correspond to splitting qualitative variables. For instance, the top internal node corresponds to splitting `Thal`. The text `Thal:a` indicates that the left-hand branch coming out of that node consists of observations with the first value of the `Thal` variable (normal), and the right-hand node consists of the remaining observations (fixed or reversible defects). The text `ChestPain:bc` two splits down the tree on the left indicates that the left-hand branch coming out of that node consists of observations with the second and third values of the `ChestPain` variable, where the possible values are typical angina, atypical angina, non-anginal pain, and asymptomatic.

Figure 8.6 has a surprising characteristic: some of the splits yield two terminal nodes that have the *same predicted value*. For instance, consider the split `RestECG<1` near the bottom right of the unpruned tree. Regardless of the value of `RestECG`, a response value of `Yes` is predicted for those observations. Why, then, is the split performed at all? The split is performed because it leads to increased *node purity*. That is, all 9 of the observations corresponding to the right-hand leaf have a response value of `Yes`, whereas 7/11 of those corresponding to the left-hand leaf have a response value of `Yes`. Why is node purity important? Suppose that we have a test observation that belongs to the region given by that right-hand leaf. Then we can be pretty certain that its response value is `Yes`. In contrast, if a test observation belongs to the region given by the left-hand leaf, then its response value is probably `Yes`, but we are much less certain. Even though the split `RestECG<1` does not reduce the classification error, it improves the Gini index and the cross-entropy, which are more sensitive to node purity.

8.1.3 Trees Versus Linear Models

Regression and classification trees have a very different flavor from the more classical approaches for regression and classification presented in Chapters 3 and 4. In particular, linear regression assumes a model of the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \tag{8.8}$$

whereas regression trees assume a model of the form

$$f(X) = \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)} \tag{8.9}$$

where $R_1, \ldots, R_M$ represent a partition of feature space, as in Figure 8.3.

Which model is better? It depends on the problem at hand. If the relationship between the features and the response is well approximated by a linear model as in (8.8), then an approach such as linear regression will likely work well, and will outperform a method such as a regression tree that does not exploit this linear structure. If instead there is a highly non-linear and complex relationship between the features and the response as indicated by model (8.9), then decision trees may outperform classical approaches. An illustrative example is displayed in Figure 8.7. The relative performances of tree-based and classical approaches can be assessed by estimating the test error, using either cross-validation or the validation set approach (Chapter 5).

Of course, other considerations beyond simply test error may come into play in selecting a statistical learning method; for instance, in certain settings, prediction using a tree may be preferred for the sake of interpretability and visualization.