2
# Statistical Learning

2.1 What Is Statistical Learning?

In order to motivate our study of statistical learning, we begin with a simple example. Suppose that we are statistical consultants hired by a client to investigate the association between advertising and sales of a particular product. The `Advertising` data set consists of the `sales` of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: `TV`, `radio`, and `newspaper`. The data are displayed in Figure 2.1. It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.

In this setting, the advertising budgets are *input variables* while `sales` is an *output variable*. The input variables are typically denoted using the symbol X, with a subscript to distinguish them. So X_1 might be the `TV` budget, X_2 the `radio` budget, and X_3 the `newspaper` budget. The inputs go by different names, such as *predictors*, *independent variables*, *features*, or sometimes just *variables*. The output variable—in this case, `sales`—is often called the *response* or *dependent variable*, and is typically denoted using the symbol Y. Throughout this book, we will use all of these terms interchangeably.

input variable
output variable

predictor
independent variable
feature
variable
response
dependent variable
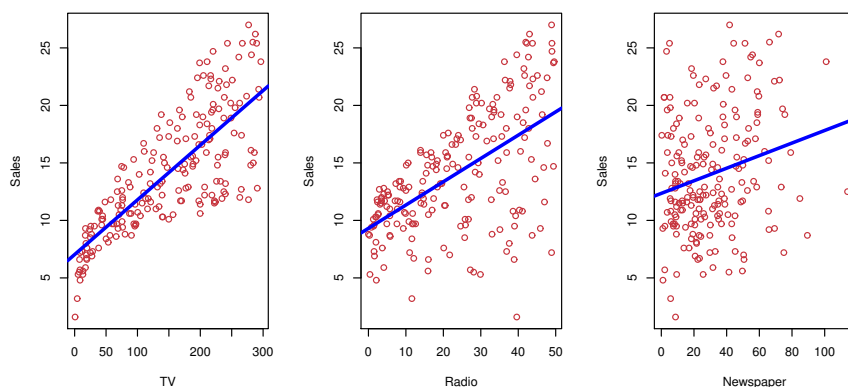
FIGURE 2.1. *The* `Advertising` *data set. The plot displays* `sales`*, in thousands of units, as a function of* `TV`*,* `radio`*, and* `newspaper` *budgets, in thousands of dollars, for* 200 *different markets. In each plot we show the simple least squares fit of* `sales` *to that variable, as described in Chapter 3. In other words, each blue line represents a simple model that can be used to predict* `sales` *using* `TV`*,* `radio`*, and* `newspaper`*, respectively.*

More generally, suppose that we observe a quantitative response Y and p different predictors, $X_1, X_2, \ldots, X_p$. We assume that there is some relationship between Y and $X = (X_1, X_2, \ldots, X_p)$, which can be written in the very general form

$$Y = f(X) + \epsilon. \tag{2.1}$$

Here f is some fixed but unknown function of $X_1, \ldots, X_p$, and ϵ is a random *error term*, which is independent of X and has mean zero. In this formulation, f represents the *systematic* information that X provides about Y.

error term

systematic

As another example, consider the left-hand panel of Figure 2.2, a plot of `income` versus `years of education` for 30 individuals in the `Income` data set. The plot suggests that one might be able to predict `income` using `years of education`. However, the function f that connects the input variable to the output variable is in general unknown. In this situation one must estimate f based on the observed points. Since `Income` is a simulated data set, f is known and is shown by the blue curve in the right-hand panel of Figure 2.2. The vertical lines represent the error terms ϵ. We note that some of the 30 observations lie above the blue curve and some lie below it; overall, the errors have approximately mean zero.

In general, the function f may involve more than one input variable. In Figure 2.3 we plot `income` as a function of `years of education` and `seniority`. Here f is a two-dimensional surface that must be estimated based on the observed data.
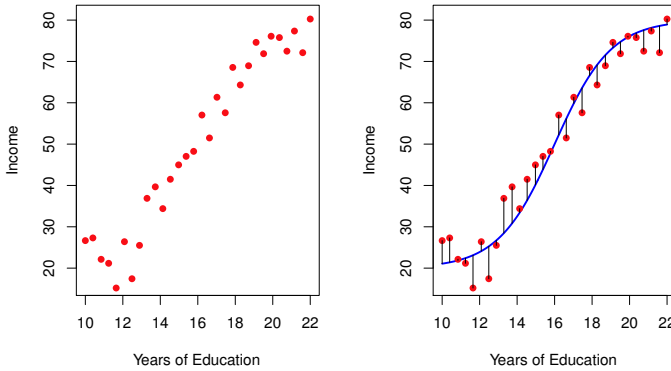
FIGURE 2.2. *The* `Income` *data set. Left: The red dots are the observed values of* `income` *(in thousands of dollars) and* `years of education` *for* 30 *individuals. Right: The blue curve represents the true underlying relationship between* `income` *and* `years of education`, *which is generally unknown (but is known in this case because the data were simulated). The black lines represent the error associated with each observation. Note that some errors are positive (if an observation lies above the blue curve) and some are negative (if an observation lies below the curve). Overall, these errors have approximately mean zero.*

In essence, statistical learning refers to a set of approaches for estimating f. In this chapter we outline some of the key theoretical concepts that arise in estimating f, as well as tools for evaluating the estimates obtained.

2.1.1   Why Estimate f?

There are two main reasons that we may wish to estimate f: *prediction* and *inference*. We discuss each in turn.

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can predict Y using

$$\hat{Y} = \hat{f}(X), \tag{2.2}$$

where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y. In this setting, $\hat{f}$ is often treated as a *black box*, in the sense that one is not typically concerned with the exact form of $\hat{f}$, provided that it yields accurate predictions for Y.

As an example, suppose that $X_1, \ldots, X_p$ are characteristics of a patient's blood sample that can be easily measured in a lab, and Y is a variable encoding the patient's risk for a severe adverse reaction to a particular
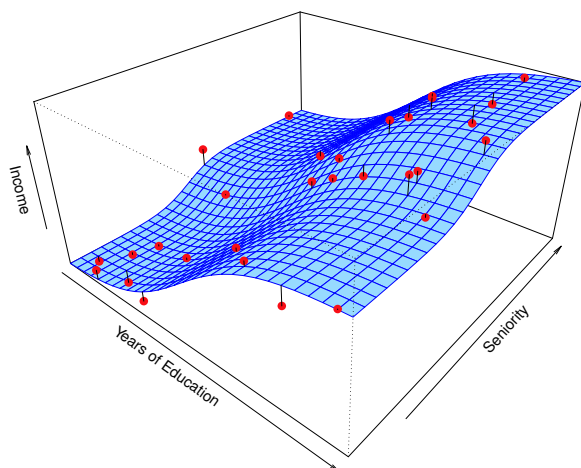
FIGURE 2.3. *The plot displays* `income` *as a function of* `years of education` *and* `seniority` *in the* `Income` *data set. The blue surface represents the true underlying relationship between* `income` *and* `years of education` *and* `seniority`, *which is known since the data are simulated. The red dots indicate the observed values of these quantities for* 30 *individuals.*

drug. It is natural to seek to predict Y using X, since we can then avoid giving the drug in question to patients who are at high risk of an adverse reaction—that is, patients for whom the estimate of Y is high.

The accuracy of $\hat{Y}$ as a prediction for Y depends on two quantities, which we will call the *reducible error* and the *irreducible error*. In general, $\hat{f}$ will not be a perfect estimate for f, and this inaccuracy will introduce some error. This error is *reducible* because we can potentially improve the accuracy of $\hat{f}$ by using the most appropriate statistical learning technique to estimate f. However, even if it were possible to form a perfect estimate for f, so that our estimated response took the form $\hat{Y} = f(X)$, our prediction would still have some error in it! This is because Y is also a function of ϵ, which, by definition, cannot be predicted using X. Therefore, variability associated with ϵ also affects the accuracy of our predictions. This is known as the *irreducible* error, because no matter how well we estimate f, we cannot reduce the error introduced by ϵ.

Why is the irreducible error larger than zero? The quantity ϵ may contain unmeasured variables that are useful in predicting Y: since we don't measure them, f cannot use them for its prediction. The quantity ϵ may also contain unmeasurable variation. For example, the risk of an adverse reaction might vary for a given patient on a given day, depending on manufacturing variation in the drug itself or the patient's general feeling of well-being on that day.

reducible
error
irreducible
error

Consider a given estimate $\hat{f}$ and a set of predictors X, which yields the prediction $\hat{Y} = \hat{f}(X)$. Assume for a moment that both $\hat{f}$ and X are fixed, so that the only variability comes from ϵ. Then, it is easy to show that

$$
\begin{aligned}
E(Y - \hat{Y})^2 &= E[f(X) + \epsilon - \hat{f}(X)]^2 \\
&= \underbrace{[f(X) - \hat{f}(X)]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}} ,
\end{aligned} \tag{2.3}
$$

where $E(Y - \hat{Y})^2$ represents the average, or *expected value*, of the squared difference between the predicted and actual value of Y, and $\text{Var}(\epsilon)$ represents the *variance* associated with the error term ϵ.

expected value

variance

The focus of this book is on techniques for estimating f with the aim of minimizing the reducible error. It is important to keep in mind that the irreducible error will always provide an upper bound on the accuracy of our prediction for Y. This bound is almost always unknown in practice.

Inference

We are often interested in understanding the association between Y and $X_1, \ldots, X_p$. In this situation we wish to estimate f, but our goal is not necessarily to make predictions for Y. Now $\hat{f}$ cannot be treated as a black box, because we need to know its exact form. In this setting, one may be interested in answering the following questions:

- *Which predictors are associated with the response?* It is often the case that only a small fraction of the available predictors are substantially associated with Y. Identifying the few *important* predictors among a large set of possible variables can be extremely useful, depending on the application.

- *What is the relationship between the response and each predictor?* Some predictors may have a positive relationship with Y, in the sense that larger values of the predictor are associated with larger values of Y. Other predictors may have the opposite relationship. Depending on the complexity of f, the relationship between the response and a given predictor may also depend on the values of the other predictors.

- *Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?* Historically, most methods for estimating f have taken a linear form. In some situations, such an assumption is reasonable or even desirable. But often the true relationship is more complicated, in which case a linear model may not provide an accurate representation of the relationship between the input and output variables.

In this book, we will see a number of examples that fall into the prediction setting, the inference setting, or a combination of the two.

For instance, consider a company that is interested in conducting a direct-marketing campaign. The goal is to identify individuals who are likely to respond positively to a mailing, based on observations of demographic variables measured on each individual. In this case, the demographic variables serve as predictors, and response to the marketing campaign (either positive or negative) serves as the outcome. The company is not interested in obtaining a deep understanding of the relationships between each individual predictor and the response; instead, the company simply wants to accurately predict the response using the predictors. This is an example of modeling for prediction.

In contrast, consider the `Advertising` data illustrated in Figure 2.1. One may be interested in answering questions such as:

- *Which media are associated with sales?*

- *Which media generate the biggest boost in sales?* or

- *How large of an increase in sales is associated with a given increase in TV advertising?*

This situation falls into the inference paradigm. Another example involves modeling the brand of a product that a customer might purchase based on variables such as price, store location, discount levels, competition price, and so forth. In this situation one might really be most interested in the association between each variable and the probability of purchase. For instance, *to what extent is the product's price associated with sales?* This is an example of modeling for inference.

Finally, some modeling could be conducted both for prediction and inference. For example, in a real estate setting, one may seek to relate values of homes to inputs such as crime rate, zoning, distance from a river, air quality, schools, income level of community, size of houses, and so forth. In this case one might be interested in the association between each individual input variable and housing price—for instance, *how much extra will a house be worth if it has a view of the river?* This is an inference problem. Alternatively, one may simply be interested in predicting the value of a home given its characteristics: *is this house under- or over-valued?* This is a prediction problem.

Depending on whether our ultimate goal is prediction, inference, or a combination of the two, different methods for estimating f may be appropriate. For example, *linear models* allow for relatively simple and interpretable inference, but may not yield as accurate predictions as some other approaches. In contrast, some of the highly non-linear approaches that we discuss in the later chapters of this book can potentially provide quite accurate predictions for Y, but this comes at the expense of a less interpretable model for which inference is more challenging.

linear model

2.1.2   How Do We Estimate f?

Throughout this book, we explore many linear and non-linear approaches for estimating f. However, these methods generally share certain characteristics. We provide an overview of these shared characteristics in this section. We will always assume that we have observed a set of n different data points. For example in Figure 2.2 we observed $n = 30$ data points. These observations are called the *training data* because we will use these observations to train, or teach, our method how to estimate f. Let x_{ij} represent the value of the jth predictor, or input, for observation i, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$. Correspondingly, let y_i represent the response variable for the ith observation. Then our training data consist of $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$.

training data

Our goal is to apply a statistical learning method to the training data in order to estimate the unknown function f. In other words, we want to find a function $\hat{f}$ such that $Y \approx \hat{f}(X)$ for any observation (X, Y). Broadly speaking, most statistical learning methods for this task can be characterized as either *parametric* or *non-parametric*. We now briefly discuss these two types of approaches.

parametric
non-parametric

Parametric Methods

Parametric methods involve a two-step model-based approach.

1. First, we make an assumption about the functional form, or shape, of f. For example, one very simple assumption is that f is linear in X:
$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p. \qquad (2.4)$$

   This is a *linear model*, which will be discussed extensively in Chapter 3. Once we have assumed that f is linear, the problem of estimating f is greatly simplified. Instead of having to estimate an entirely arbitrary p-dimensional function $f(X)$, one only needs to estimate the $p + 1$ coefficients $\beta_0, \beta_1, \ldots, \beta_p$.

2. After a model has been selected, we need a procedure that uses the training data to *fit* or *train* the model. In the case of the linear model (2.4), we need to estimate the parameters $\beta_0, \beta_1, \ldots, \beta_p$. That is, we want to find values of these parameters such that

fit
train

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p.$$

   The most common approach to fitting the model (2.4) is referred to as *(ordinary) least squares*, which we discuss in Chapter 3. However, least squares is one of many possible ways to fit the linear model. In Chapter 6, we discuss other approaches for estimating the parameters in (2.4).

least squares
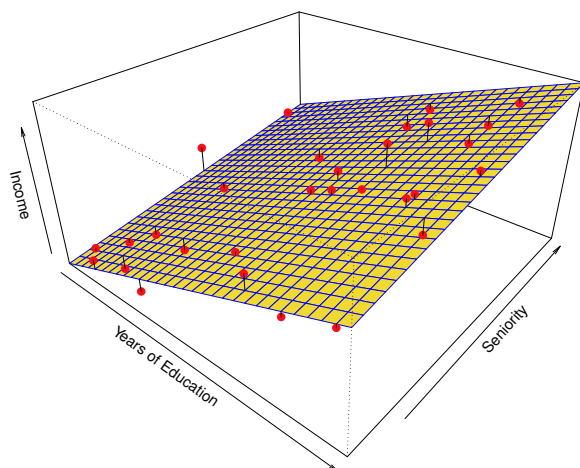
FIGURE 2.4. *A linear model fit by least squares to the* `Income` *data from Figure 2.3. The observations are shown in red, and the yellow plane indicates the least squares fit to the data.*

The model-based approach just described is referred to as *parametric*; it reduces the problem of estimating f down to one of estimating a set of parameters. Assuming a parametric form for f simplifies the problem of estimating f because it is generally much easier to estimate a set of parameters, such as $\beta_0, \beta_1, \ldots, \beta_p$ in the linear model (2.4), than it is to fit an entirely arbitrary function f. The potential disadvantage of a parametric approach is that the model we choose will usually not match the true unknown form of f. If the chosen model is too far from the true f, then our estimate will be poor. We can try to address this problem by choosing *flexible* models that can fit many different possible functional forms for f. But in general, fitting a more flexible model requires estimating a greater number of parameters. These more complex models can lead to a phenomenon known as *overfitting* the data, which essentially means they follow the errors, or *noise*, too closely. These issues are discussed throughout this book.
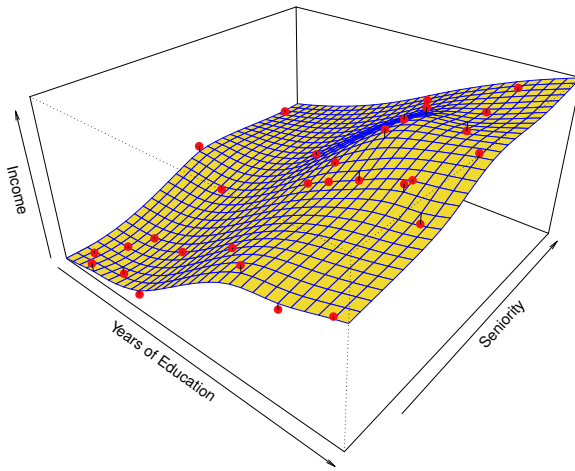
flexible

overfitting
noise

Figure 2.4 shows an example of the parametric approach applied to the `Income` data from Figure 2.3. We have fit a linear model of the form

$$\texttt{income} \approx \beta_0 + \beta_1 \times \texttt{education} + \beta_2 \times \texttt{seniority}.$$

Since we have assumed a linear relationship between the response and the two predictors, the entire fitting problem reduces to estimating β_0, β_1, and β_2, which we do using least squares linear regression. Comparing Figure 2.3 to Figure 2.4, we can see that the linear fit given in Figure 2.4 is not quite right: the true f has some curvature that is not captured in the linear fit. However, the linear fit still appears to do a reasonable job of capturing the

FIGURE 2.5. *A smooth thin-plate spline fit to the* `Income` *data from Figure 2.3 is shown in yellow; the observations are displayed in red. Splines are discussed in Chapter 7.*

positive relationship between `years of education` and `income`, as well as the slightly less positive relationship between `seniority` and `income`. It may be that with such a small number of observations, this is the best we can do.

Non-Parametric Methods

Non-parametric methods do not make explicit assumptions about the functional form of f. Instead they seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly. Such approaches can have a major advantage over parametric approaches: by avoiding the assumption of a particular functional form for f, they have the potential to accurately fit a wider range of possible shapes for f. Any parametric approach brings with it the possibility that the functional form used to estimate f is very different from the true f, in which case the resulting model will not fit the data well. In contrast, non-parametric approaches completely avoid this danger, since essentially no assumption about the form of f is made. But non-parametric approaches do suffer from a major disadvantage: since they do not reduce the problem of estimating f to a small number of parameters, a very large number of observations (far more than is typically needed for a parametric approach) is required in order to obtain an accurate estimate for f.

An example of a non-parametric approach to fitting the `Income` data is shown in Figure 2.5. A *thin-plate spline* is used to estimate f. This approach does not impose any pre-specified model on f. It instead attempts to produce an estimate for f that is as close as possible to the observed data, subject to the fit—that is, the yellow surface in Figure 2.5—being

thin-plate spline
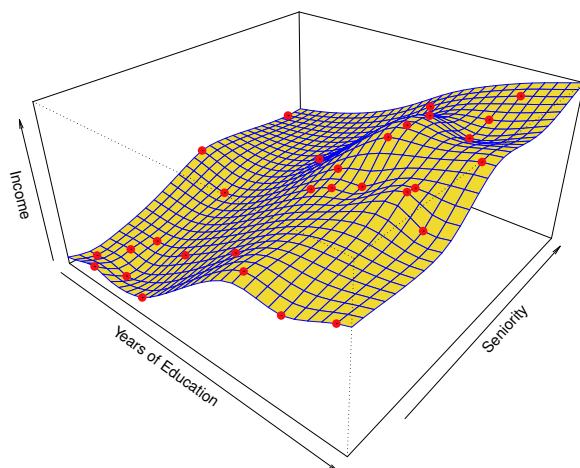
FIGURE 2.6. *A rough thin-plate spline fit to the* `Income` *data from Figure 2.3. This fit makes zero errors on the training data.*

smooth. In this case, the non-parametric fit has produced a remarkably accurate estimate of the true f shown in Figure 2.3. In order to fit a thin-plate spline, the data analyst must select a level of smoothness. Figure 2.6 shows the same thin-plate spline fit using a lower level of smoothness, allowing for a rougher fit. The resulting estimate fits the observed data perfectly! However, the spline fit shown in Figure 2.6 is far more variable than the true function f, from Figure 2.3. This is an example of overfitting the data, which we discussed previously. It is an undesirable situation because the fit obtained will not yield accurate estimates of the response on new observations that were not part of the original training data set. We discuss methods for choosing the *correct* amount of smoothness in Chapter 5. Splines are discussed in Chapter 7.

   As we have seen, there are advantages and disadvantages to parametric and non-parametric methods for statistical learning. We explore both types of methods throughout this book.

2.1.3   The Trade-Off Between Prediction Accuracy and Model Interpretability

Of the many methods that we examine in this book, some are less flexible, or more restrictive, in the sense that they can produce just a relatively small range of shapes to estimate f. For example, linear regression is a relatively inflexible approach, because it can only generate linear functions such as the lines shown in Figure 2.1 or the plane shown in Figure 2.4. Other methods, such as the thin plate splines shown in Figures 2.5 and 2.6,
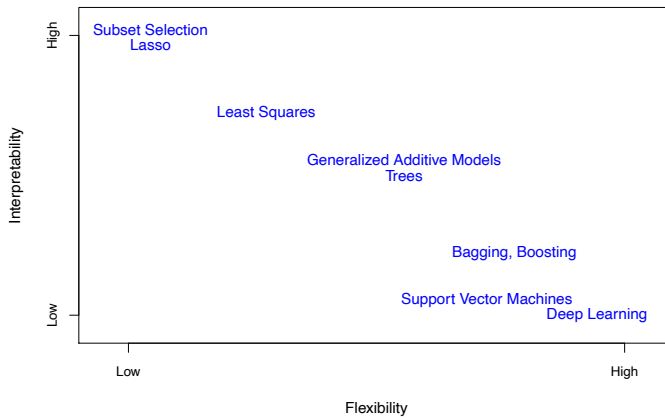
FIGURE 2.7. *A representation of the tradeoff between flexibility and inter-pretability, using different statistical learning methods. In general, as the flexibil-ity of a method increases, its interpretability decreases.*

are considerably more flexible because they can generate a much wider range of possible shapes to estimate f.

One might reasonably ask the following question: *why would we ever choose to use a more restrictive method instead of a very flexible approach?* There are several reasons that we might prefer a more restrictive model. If we are mainly interested in inference, then restrictive models are much more interpretable. For instance, when inference is the goal, the linear model may be a good choice since it will be quite easy to understand the relationship between Y and $X_1, X_2, \ldots, X_p$. In contrast, very flexible approaches, such as the splines discussed in Chapter 7 and displayed in Figures 2.5 and 2.6, and the boosting methods discussed in Chapter 8, can lead to such complicated estimates of f that it is difficult to understand how any individual predictor is associated with the response.

Figure 2.7 provides an illustration of the trade-off between flexibility and interpretability for some of the methods that we cover in this book. Least squares linear regression, discussed in Chapter 3, is relatively inflexible but is quite interpretable. The *lasso*, discussed in Chapter 6, relies upon the linear model (2.4) but uses an alternative fitting procedure for estimating the coefficients $\beta_0, \beta_1, \ldots, \beta_p$. The new procedure is more restrictive in estimating the coefficients, and sets a number of them to exactly zero. Hence in this sense the lasso is a less flexible approach than linear regression. It is also more interpretable than linear regression, because in the final model the response variable will only be related to a small subset of the predictors—namely, those with nonzero coefficient estimates. *Generalized additive models* (GAMs), discussed in Chapter 7, instead extend the linear model (2.4) to allow for certain non-linear relationships. Consequently,

lasso

generalized additive model

GAMs are more flexible than linear regression. They are also somewhat less interpretable than linear regression, because the relationship between each predictor and the response is now modeled using a curve. Finally, fully non-linear methods such as *bagging*, *boosting*, *support vector machines* with non-linear kernels, and *neural networks* (deep learning), discussed in Chapters 8, 9, and 10, are highly flexible approaches that are harder to interpret.

bagging
boosting
support vector machine

We have established that when inference is the goal, there are clear advantages to using simple and relatively inflexible statistical learning methods. In some settings, however, we are only interested in prediction, and the interpretability of the predictive model is simply not of interest. For instance, if we seek to develop an algorithm to predict the price of a stock, our sole requirement for the algorithm is that it predict accurately—interpretability is not a concern. In this setting, we might expect that it will be best to use the most flexible model available. Surprisingly, this is not always the case! We will often obtain more accurate predictions using a less flexible method. This phenomenon, which may seem counterintuitive at first glance, has to do with the potential for overfitting in highly flexible methods. We saw an example of overfitting in Figure 2.6. We will discuss this very important concept further in Section 2.2 and throughout this book.

2.1.4   Supervised Versus Unsupervised Learning

Most statistical learning problems fall into one of two categories: *supervised* or *unsupervised*. The examples that we have discussed so far in this chapter all fall into the supervised learning domain. For each observation of the predictor measurement(s) x_i, $i = 1, \ldots, n$ there is an associated response measurement y_i. We wish to fit a model that relates the response to the predictors, with the aim of accurately predicting the response for future observations (prediction) or better understanding the relationship between the response and the predictors (inference). Many classical statistical learning methods such as linear regression and *logistic regression* (Chapter 4), as well as more modern approaches such as GAM, boosting, and support vector machines, operate in the supervised learning domain. The vast majority of this book is devoted to this setting.

supervised
unsupervised

logistic regression

By contrast, unsupervised learning describes the somewhat more challenging situation in which for every observation $i = 1, \ldots, n$, we observe a vector of measurements x_i but no associated response y_i. It is not possible to fit a linear regression model, since there is no response variable to predict. In this setting, we are in some sense working blind; the situation is referred to as *unsupervised* because we lack a response variable that can supervise our analysis. What sort of statistical analysis is possible? We can seek to understand the relationships between the variables or between the observations. One statistical learning tool that we may use
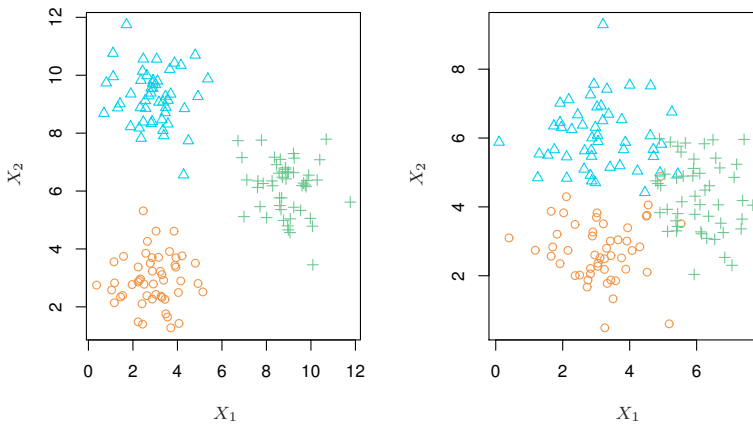
FIGURE 2.8. *A clustering data set involving three groups. Each group is shown using a different colored symbol.* Left: *The three groups are well-separated. In this setting, a clustering approach should successfully identify the three groups.* Right: *There is some overlap among the groups. Now the clustering task is more challenging.*

in this setting is *cluster analysis*, or clustering. The goal of cluster analysis is to ascertain, on the basis of $x_1, \ldots, x_n$, whether the observations fall into relatively distinct groups. For example, in a market segmentation study we might observe multiple characteristics (variables) for potential customers, such as zip code, family income, and shopping habits. We might believe that the customers fall into different groups, such as big spenders versus low spenders. If the information about each customer's spending patterns were available, then a supervised analysis would be possible. However, this information is not available—that is, we do not know whether each potential customer is a big spender or not. In this setting, we can try to cluster the customers on the basis of the variables measured, in order to identify distinct groups of potential customers. Identifying such groups can be of interest because it might be that the groups differ with respect to some property of interest, such as spending habits.

cluster
analysis

Figure 2.8 provides a simple illustration of the clustering problem. We have plotted 150 observations with measurements on two variables, X_1 and X_2. Each observation corresponds to one of three distinct groups. For illustrative purposes, we have plotted the members of each group using different colors and symbols. However, in practice the group memberships are unknown, and the goal is to determine the group to which each observation belongs. In the left-hand panel of Figure 2.8, this is a relatively easy task because the groups are well-separated. By contrast, the right-hand panel illustrates a more challenging setting in which there is some overlap

between the groups. A clustering method could not be expected to assign all of the overlapping points to their correct group (blue, green, or orange).

In the examples shown in Figure 2.8, there are only two variables, and so one can simply visually inspect the scatterplots of the observations in order to identify clusters. However, in practice, we often encounter data sets that contain many more than two variables. In this case, we cannot easily plot the observations. For instance, if there are p variables in our data set, then $p(p-1)/2$ distinct scatterplots can be made, and visual inspection is simply not a viable way to identify clusters. For this reason, automated clustering methods are important. We discuss clustering and other unsupervised learning approaches in Chapter 12.

Many problems fall naturally into the supervised or unsupervised learning paradigms. However, sometimes the question of whether an analysis should be considered supervised or unsupervised is less clear-cut. For instance, suppose that we have a set of n observations. For m of the observations, where $m < n$, we have both predictor measurements and a response measurement. For the remaining $n - m$ observations, we have predictor measurements but no response measurement. Such a scenario can arise if the predictors can be measured relatively cheaply but the corresponding responses are much more expensive to collect. We refer to this setting as a *semi-supervised learning* problem. In this setting, we wish to use a statistical learning method that can incorporate the m observations for which response measurements are available as well as the $n - m$ observations for which they are not. Although this is an interesting topic, it is beyond the scope of this book.

semi-supervised learning

2.1.5 *Regression Versus Classification Problems*

Variables can be characterized as either *quantitative* or *qualitative* (also known as *categorical*). Quantitative variables take on numerical values. Examples include a person's age, height, or income, the value of a house, and the price of a stock. In contrast, qualitative variables take on values in one of K different *classes*, or categories. Examples of qualitative variables include a person's marital status (married or not), the brand of product purchased (brand A, B, or C), whether a person defaults on a debt (yes or no), or a cancer diagnosis (Acute Myelogenous Leukemia, Acute Lymphoblastic Leukemia, or No Leukemia). We tend to refer to problems with a quantitative response as *regression* problems, while those involving a qualitative response are often referred to as *classification* problems. However, the distinction is not always that crisp. Least squares linear regression (Chapter 3) is used with a quantitative response, whereas logistic regression (Chapter 4) is typically used with a qualitative (two-class, or *binary*) response. Thus, despite its name, logistic regression is a classification method. But since it estimates class probabilities, it can be thought of as a regression method as well. Some statistical methods, such as K-nearest neighbors

quantitative
qualitative
categorical

class

regression
classification

binary

(Chapters 2 and 4) and boosting (Chapter 8), can be used in the case of either quantitative or qualitative responses.

We tend to select statistical learning methods on the basis of whether the response is quantitative or qualitative; i.e. we might use linear regression when quantitative and logistic regression when qualitative. However, whether the *predictors* are qualitative or quantitative is generally considered less important. Most of the statistical learning methods discussed in this book can be applied regardless of the predictor variable type, provided that any qualitative predictors are properly *coded* before the analysis is performed. This is discussed in Chapter 3.

2.2   Assessing Model Accuracy

One of the key aims of this book is to introduce the reader to a wide range of statistical learning methods that extend far beyond the standard linear regression approach. Why is it necessary to introduce so many different statistical learning approaches, rather than just a single *best* method? *There is no free lunch in statistics:* no one method dominates all others over all possible data sets. On a particular data set, one specific method may work best, but some other method may work better on a similar but different data set. Hence it is an important task to decide for any given set of data which method produces the best results. Selecting the best approach can be one of the most challenging parts of performing statistical learning in practice.

In this section, we discuss some of the most important concepts that arise in selecting a statistical learning procedure for a specific data set. As the book progresses, we will explain how the concepts presented here can be applied in practice.

2.2.1   *Measuring the Quality of Fit*

In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data. That is, we need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation. In the regression setting, the most commonly-used measure is the *mean squared error* (MSE), given by

mean squared error

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2, \qquad (2.5)$$

where $\hat{f}(x_i)$ is the prediction that $\hat{f}$ gives for the ith observation. The MSE will be small if the predicted responses are very close to the true responses,

and will be large if for some of the observations, the predicted and true responses differ substantially.

The MSE in (2.5) is computed using the training data that was used to fit the model, and so should more accurately be referred to as the *training MSE*. But in general, we do not really care how well the method works on the training data. Rather, *we are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen test data.* Why is this what we care about? Suppose that we are interested in developing an algorithm to predict a stock's price based on previous stock returns. We can train the method using stock returns from the past 6 months. But we don't really care how well our method predicts last week's stock price. We instead care about how well it will predict tomorrow's price or next month's price. On a similar note, suppose that we have clinical measurements (e.g. weight, blood pressure, height, age, family history of disease) for a number of patients, as well as information about whether each patient has diabetes. We can use these patients to train a statistical learning method to predict risk of diabetes based on clinical measurements. In practice, we want this method to accurately predict diabetes risk for *future patients* based on their clinical measurements. We are not very interested in whether or not the method accurately predicts diabetes risk for patients used to train the model, since we already know which of those patients have diabetes.

To state it more mathematically, suppose that we fit our statistical learning method on our training observations $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, and we obtain the estimate $\hat{f}$. We can then compute $\hat{f}(x_1), \hat{f}(x_2), \ldots, \hat{f}(x_n)$. If these are approximately equal to $y_1, y_2, \ldots, y_n$, then the training MSE given by (2.5) is small. However, we are really not interested in whether $\hat{f}(x_i) \approx y_i$; instead, we want to know whether $\hat{f}(x_0)$ is approximately equal to y_0, where (x_0, y_0) is a *previously unseen test observation not used to train the statistical learning method.* We want to choose the method that gives the lowest *test MSE*, as opposed to the lowest training MSE. In other words, if we had a large number of test observations, we could compute

$$\text{Ave}(y_0 - \hat{f}(x_0))^2, \tag{2.6}$$

the average squared prediction error for these test observations (x_0, y_0). We'd like to select the model for which this quantity is as small as possible.

How can we go about trying to select a method that minimizes the test MSE? In some settings, we may have a test data set available—that is, we may have access to a set of observations that were not used to train the statistical learning method. We can then simply evaluate (2.6) on the test observations, and select the learning method for which the test MSE is smallest. But what if no test observations are available? In that case, one might imagine simply selecting a statistical learning method that minimizes the training MSE (2.5). This seems like it might be a sensible approach,
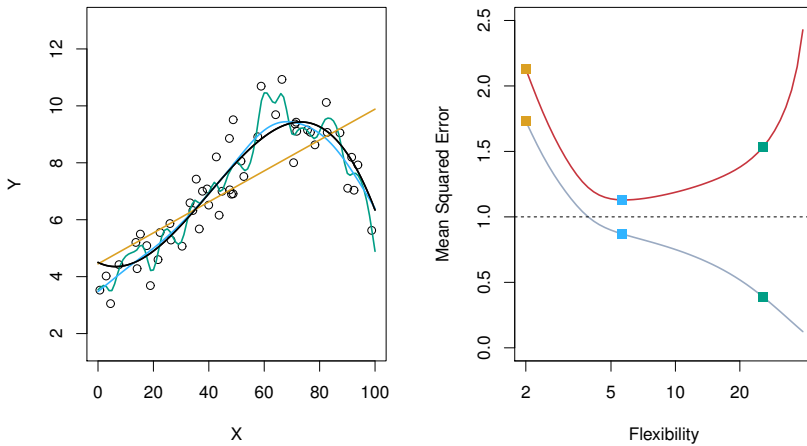
FIGURE 2.9. Left: *Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves).* Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.*

since the training MSE and the test MSE appear to be closely related. Unfortunately, there is a fundamental problem with this strategy: there is no guarantee that the method with the lowest training MSE will also have the lowest test MSE. Roughly speaking, the problem is that many statistical methods specifically estimate coefficients so as to minimize the training set MSE. For these methods, the training set MSE can be quite small, but the test MSE is often much larger.

Figure 2.9 illustrates this phenomenon on a simple example. In the left-hand panel of Figure 2.9, we have generated observations from (2.1) with the true f given by the black curve. The orange, blue and green curves illustrate three possible estimates for f obtained using methods with increasing levels of flexibility. The orange line is the linear regression fit, which is relatively inflexible. The blue and green curves were produced using *smoothing splines*, discussed in Chapter 7, with different levels of smoothness. It is clear that as the level of flexibility increases, the curves fit the observed data more closely. The green curve is the most flexible and matches the data very well; however, we observe that it fits the true f (shown in black) poorly because it is too wiggly. By adjusting the level of flexibility of the smoothing spline fit, we can produce many different fits to this data.

We now move on to the right-hand panel of Figure 2.9. The grey curve displays the average training MSE as a function of flexibility, or more formally the *degrees of freedom*, for a number of smoothing splines. The degrees of freedom is a quantity that summarizes the flexibility of a curve;

smoothing
spline

degrees of
freedom

it is discussed more fully in Chapter 7. The orange, blue and green squares indicate the MSEs associated with the corresponding curves in the left-hand panel. A more restricted and hence smoother curve has fewer degrees of freedom than a wiggly curve—note that in Figure 2.9, linear regression is at the most restrictive end, with two degrees of freedom. The training MSE declines monotonically as flexibility increases. In this example the true f is non-linear, and so the orange linear fit is not flexible enough to estimate f well. The green curve has the lowest training MSE of all three methods, since it corresponds to the most flexible of the three curves fit in the left-hand panel.

In this example, we know the true function f, and so we can also compute the test MSE over a very large test set, as a function of flexibility. (Of course, in general f is unknown, so this will not be possible.) The test MSE is displayed using the red curve in the right-hand panel of Figure 2.9. As with the training MSE, the test MSE initially declines as the level of flexibility increases. However, at some point the test MSE levels off and then starts to increase again. Consequently, the orange and green curves both have high test MSE. The blue curve minimizes the test MSE, which should not be surprising given that visually it appears to estimate f the best in the left-hand panel of Figure 2.9. The horizontal dashed line indicates $\text{Var}(\epsilon)$, the irreducible error in (2.3), which corresponds to the lowest achievable test MSE among all possible methods. Hence, the smoothing spline represented by the blue curve is close to optimal.

In the right-hand panel of Figure 2.9, as the flexibility of the statistical learning method increases, we observe a monotone decrease in the training MSE and a *U-shape* in the test MSE. This is a fundamental property of statistical learning that holds regardless of the particular data set at hand and regardless of the statistical method being used. As model flexibility increases, the training MSE will decrease, but the test MSE may not. When a given method yields a small training MSE but a large test MSE, we are said to be *overfitting* the data. This happens because our statistical learning procedure is working too hard to find patterns in the training data, and may be picking up some patterns that are just caused by random chance rather than by true properties of the unknown function f. When we overfit the training data, the test MSE will be very large because the supposed patterns that the method found in the training data simply don't exist in the test data. Note that regardless of whether or not overfitting has occurred, we almost always expect the training MSE to be smaller than the test MSE because most statistical learning methods either directly or indirectly seek to minimize the training MSE. Overfitting refers specifically to the case in which a less flexible model would have yielded a smaller test MSE.

Figure 2.10 provides another example in which the true f is approximately linear. Again we observe that the training MSE decreases monotonically as the model flexibility increases, and that there is a U-shape in
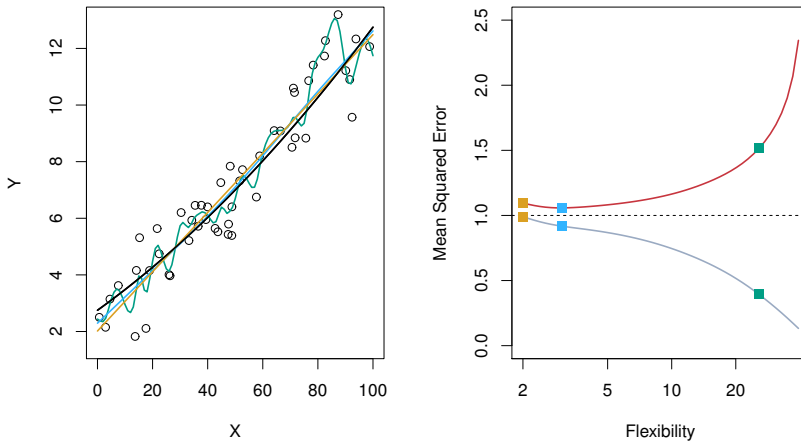
FIGURE 2.10. *Details are as in Figure 2.9, using a different true f that is much closer to linear. In this setting, linear regression provides a very good fit to the data.*

the test MSE. However, because the truth is close to linear, the test MSE only decreases slightly before increasing again, so that the orange least squares fit is substantially better than the highly flexible green curve. Finally, Figure 2.11 displays an example in which f is highly non-linear. The training and test MSE curves still exhibit the same general patterns, but now there is a rapid decrease in both curves before the test MSE starts to increase slowly.

In practice, one can usually compute the training MSE with relative ease, but estimating the test MSE is considerably more difficult because usually no test data are available. As the previous three examples illustrate, the flexibility level corresponding to the model with the minimal test MSE can vary considerably among data sets. Throughout this book, we discuss a variety of approaches that can be used in practice to estimate this minimum point. One important method is *cross-validation* (Chapter 5), which is a method for estimating the test MSE using the training data.

cross-
validation

2.2.2 The Bias-Variance Trade-Off

The U-shape observed in the test MSE curves (Figures 2.9–2.11) turns out to be the result of two competing properties of statistical learning methods. Though the mathematical proof is beyond the scope of this book, it is possible to show that the expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental quantities: the *variance* of $\hat{f}(x_0)$, the squared *bias* of $\hat{f}(x_0)$ and the variance of the error

variance
bias
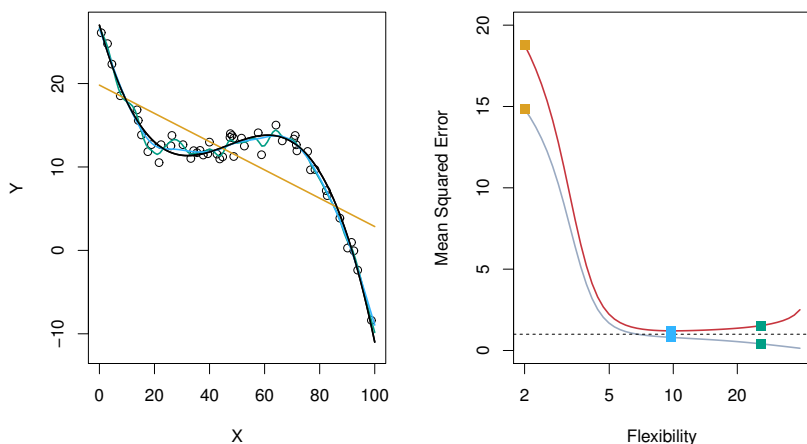
FIGURE 2.11. *Details are as in Figure 2.9, using a different f that is far from linear. In this setting, linear regression provides a very poor fit to the data.*

terms ϵ. That is,

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon). \qquad (2.7)$$

Here the notation $E\left(y_0 - \hat{f}(x_0)\right)^2$ defines the *expected test MSE* at x_0, and refers to the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0. The overall expected test MSE can be computed by averaging $E\left(y_0 - \hat{f}(x_0)\right)^2$ over all possible values of x_0 in the test set.

expected
test MSE

Equation 2.7 tells us that in order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves *low variance* and *low bias*. Note that variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below $\text{Var}(\epsilon)$, the irreducible error from (2.3).

What do we mean by the *variance* and *bias* of a statistical learning method? *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. Since the training data are used to fit the statistical learning method, different training data sets will result in a different $\hat{f}$. But ideally the estimate for f should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in $\hat{f}$. In general, more flexible statistical methods have higher variance. Consider the green and orange curves in Figure 2.9. The flexible green curve is following the observations very closely. It has high variance because changing any

one of these data points may cause the estimate $\hat{f}$ to change considerably. In contrast, the orange least squares line is relatively inflexible and has low variance, because moving any single observation will likely cause only a small shift in the position of the line.

On the other hand, *bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. For example, linear regression assumes that there is a linear relationship between Y and $X_1, X_2, \ldots, X_p$. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f. In Figure 2.11, the true f is substantially non-linear, so no matter how many training observations we are given, it will not be possible to produce an accurate estimate using linear regression. In other words, linear regression results in high bias in this example. However, in Figure 2.10 the true f is very close to linear, and so given enough data, it should be possible for linear regression to produce an accurate estimate. Generally, more flexible methods result in less bias.

As a general rule, as we use more flexible methods, the variance will increase and the bias will decrease. The relative rate of change of these two quantities determines whether the test MSE increases or decreases. As we increase the flexibility of a class of methods, the bias tends to initially decrease faster than the variance increases. Consequently, the expected test MSE declines. However, at some point increasing flexibility has little impact on the bias but starts to significantly increase the variance. When this happens the test MSE increases. Note that we observed this pattern of decreasing test MSE followed by increasing test MSE in the right-hand panels of Figures 2.9–2.11.

The three plots in Figure 2.12 illustrate Equation 2.7 for the examples in Figures 2.9–2.11. In each case the blue solid curve represents the squared bias, for different levels of flexibility, while the orange curve corresponds to the variance. The horizontal dashed line represents $\text{Var}(\epsilon)$, the irreducible error. Finally, the red curve, corresponding to the test set MSE, is the sum of these three quantities. In all three cases, the variance increases and the bias decreases as the method's flexibility increases. However, the flexibility level corresponding to the optimal test MSE differs considerably among the three data sets, because the squared bias and variance change at different rates in each of the data sets. In the left-hand panel of Figure 2.12, the bias initially decreases rapidly, resulting in an initial sharp decrease in the expected test MSE. On the other hand, in the center panel of Figure 2.12 the true f is close to linear, so there is only a small decrease in bias as flexibility increases, and the test MSE only declines slightly before increasing rapidly as the variance increases. Finally, in the right-hand panel of Figure 2.12, as flexibility increases, there is a dramatic decline in bias because the true f is very non-linear. There is also very little increase in variance
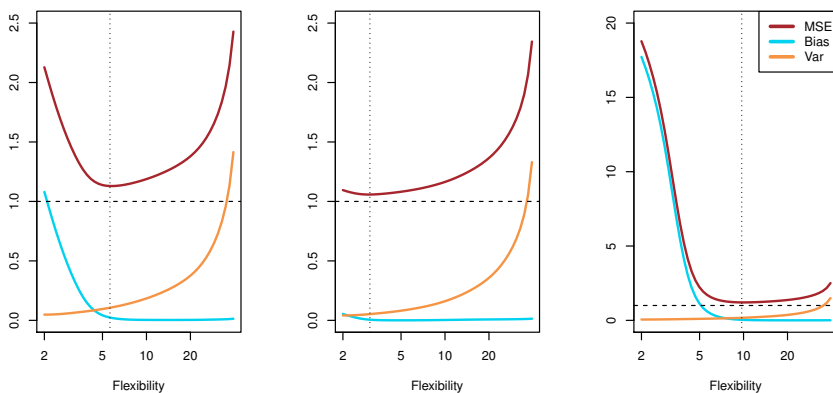
FIGURE 2.12. *Squared bias (blue curve), variance (orange curve), Var(ϵ) (dashed line), and test MSE (red curve) for the three data sets in Figures 2.9–2.11. The vertical dotted line indicates the flexibility level corresponding to the smallest test MSE.*

as flexibility increases. Consequently, the test MSE declines substantially before experiencing a small increase as model flexibility increases.

The relationship between bias, variance, and test set MSE given in Equation 2.7 and displayed in Figure 2.12 is referred to as the *bias-variance trade-off*. Good test set performance of a statistical learning method requires low variance as well as low squared bias. This is referred to as a trade-off because it is easy to obtain a method with extremely low bias but high variance (for instance, by drawing a curve that passes through every single training observation) or a method with very low variance but high bias (by fitting a horizontal line to the data). The challenge lies in finding a method for which both the variance and the squared bias are low. This trade-off is one of the most important recurring themes in this book.

In a real-life situation in which f is unobserved, it is generally not possible to explicitly compute the test MSE, bias, or variance for a statistical learning method. Nevertheless, one should always keep the bias-variance trade-off in mind. In this book we explore methods that are extremely flexible and hence can essentially eliminate bias. However, this does not guarantee that they will outperform a much simpler method such as linear regression. To take an extreme example, suppose that the true f is linear. In this situation linear regression will have no bias, making it very hard for a more flexible method to compete. In contrast, if the true f is highly non-linear and we have an ample number of training observations, then we may do better using a highly flexible approach, as in Figure 2.11. In Chapter 5 we discuss cross-validation, which is a way to estimate the test MSE using the training data.

2.2.3   The Classification Setting

Thus far, our discussion of model accuracy has been focused on the regression setting. But many of the concepts that we have encountered, such as the bias-variance trade-off, transfer over to the classification setting with only some modifications due to the fact that y_i is no longer quantitative. Suppose that we seek to estimate f on the basis of training observations $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where now $y_1, \ldots, y_n$ are qualitative. The most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the training *error rate*, the proportion of mistakes that are made if we apply our estimate $\hat{f}$ to the training observations:

error rate

$$\frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i). \tag{2.8}$$

Here $\hat{y}_i$ is the predicted class label for the ith observation using $\hat{f}$. And $I(y_i \neq \hat{y}_i)$ is an *indicator variable* that equals 1 if $y_i \neq \hat{y}_i$ and zero if $y_i = \hat{y}_i$. If $I(y_i \neq \hat{y}_i) = 0$ then the ith observation was classified correctly by our classification method; otherwise it was misclassified. Hence Equation 2.8 computes the fraction of incorrect classifications.

indicator variable

Equation 2.8 is referred to as the *training error* rate because it is computed based on the data that was used to train our classifier. As in the regression setting, we are most interested in the error rates that result from applying our classifier to test observations that were not used in training. The *test error* rate associated with a set of test observations of the form (x_0, y_0) is given by

training error

test error

$$\text{Ave}\left(I(y_0 \neq \hat{y}_0)\right), \tag{2.9}$$

where $\hat{y}_0$ is the predicted class label that results from applying the classifier to the test observation with predictor x_0. A *good* classifier is one for which the test error (2.9) is smallest.

The Bayes Classifier

It is possible to show (though the proof is outside of the scope of this book) that the test error rate given in (2.9) is minimized, on average, by a very simple classifier that *assigns each observation to the most likely class, given its predictor values.* In other words, we should simply assign a test observation with predictor vector x_0 to the class j for which

$$\Pr(Y = j | X = x_0) \tag{2.10}$$

is largest. Note that (2.10) is a *conditional probability*: it is the probability that $Y = j$, given the observed predictor vector x_0. This very simple classifier is called the *Bayes classifier*. In a two-class problem where there are only two possible response values, say *class 1* or *class 2*, the Bayes classifier

conditional probability

Bayes classifier
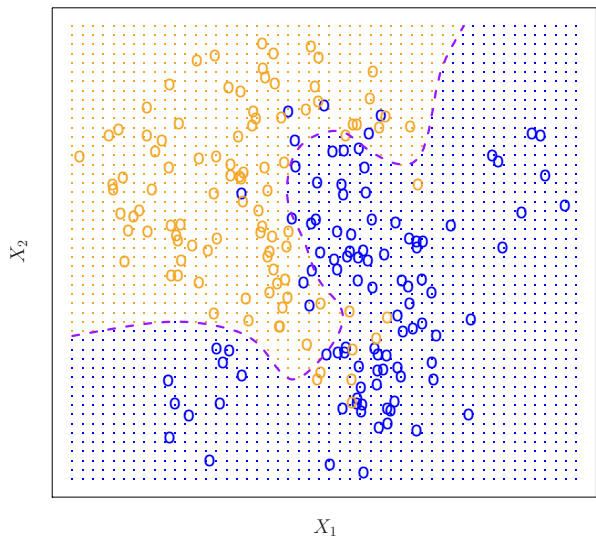
FIGURE 2.13. *A simulated data set consisting of* 100 *observations in each of two groups, indicated in blue and in orange. The purple dashed line represents the Bayes decision boundary. The orange background grid indicates the region in which a test observation will be assigned to the orange class, and the blue background grid indicates the region in which a test observation will be assigned to the blue class.*

corresponds to predicting class one if $\Pr(Y = 1|X = x_0) > 0.5$, and class two otherwise.

Figure 2.13 provides an example using a simulated data set in a two-dimensional space consisting of predictors X_1 and X_2. The orange and blue circles correspond to training observations that belong to two different classes. For each value of X_1 and X_2, there is a different probability of the response being orange or blue. Since this is simulated data, we know how the data were generated and we can calculate the conditional probabilities for each value of X_1 and X_2. The orange shaded region reflects the set of points for which $\Pr(Y = \text{orange}|X)$ is greater than 50 %, while the blue shaded region indicates the set of points for which the probability is below 50 %. The purple dashed line represents the points where the probability is exactly 50 %. This is called the *Bayes decision boundary*. The Bayes classifier's prediction is determined by the Bayes decision boundary; an observation that falls on the orange side of the boundary will be assigned to the orange class, and similarly an observation on the blue side of the boundary will be assigned to the blue class.

The Bayes classifier produces the lowest possible test error rate, called the *Bayes error rate*. Since the Bayes classifier will always choose the class for which (2.10) is largest, the error rate will be $1 - \max_j \Pr(Y = j|X = x_0)$

Bayes decision boundary

Bayes error rate

at $X = x_0$. In general, the overall Bayes error rate is given by

$$1 - E \left(\max_j \Pr(Y = j | X) \right), \tag{2.11}$$

where the expectation averages the probability over all possible values of X. For our simulated data, the Bayes error rate is 0.133. It is greater than zero, because the classes overlap in the true population, which implies that $\max_j \Pr(Y = j | X = x_0) < 1$ for some values of x_0. The Bayes error rate is analogous to the irreducible error, discussed earlier.

K-Nearest Neighbors

In theory we would always like to predict qualitative responses using the Bayes classifier. But for real data, we do not know the conditional distribution of Y given X, and so computing the Bayes classifier is impossible. Therefore, the Bayes classifier serves as an unattainable gold standard against which to compare other methods. Many approaches attempt to estimate the conditional distribution of Y given X, and then classify a given observation to the class with highest *estimated* probability. One such method is the *K-nearest neighbors* (KNN) classifier. Given a positive integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by $\mathcal{N}_0$. It then estimates the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

K-nearest neighbors

$$\Pr(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j). \tag{2.12}$$

Finally, KNN classifies the test observation x_0 to the class with the largest probability from (2.12).

Figure 2.14 provides an illustrative example of the KNN approach. In the left-hand panel, we have plotted a small training data set consisting of six blue and six orange observations. Our goal is to make a prediction for the point labeled by the black cross. Suppose that we choose $K = 3$. Then KNN will first identify the three observations that are closest to the cross. This neighborhood is shown as a circle. It consists of two blue points and one orange point, resulting in estimated probabilities of 2/3 for the blue class and 1/3 for the orange class. Hence KNN will predict that the black cross belongs to the blue class. In the right-hand panel of Figure 2.14 we have applied the KNN approach with $K = 3$ at all of the possible values for X_1 and X_2, and have drawn in the corresponding KNN decision boundary.

Despite the fact that it is a very simple approach, KNN can often produce classifiers that are surprisingly close to the optimal Bayes classifier. Figure 2.15 displays the KNN decision boundary, using $K = 10$, when applied to the larger simulated data set from Figure 2.13. Notice that even
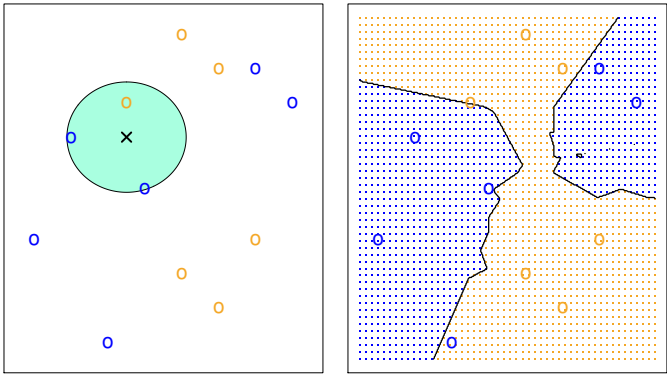
FIGURE 2.14.   *The KNN approach, using $K = 3$, is illustrated in a simple situation with six blue observations and six orange observations.* Left: *a test observation at which a predicted class label is desired is shown as a black cross. The three closest points to the test observation are identified, and it is predicted that the test observation belongs to the most commonly-occurring class, in this case blue.* Right: *The KNN decision boundary for this example is shown in black. The blue grid indicates the region in which a test observation will be assigned to the blue class, and the orange grid indicates the region in which it will be assigned to the orange class.*

KNN: K=10
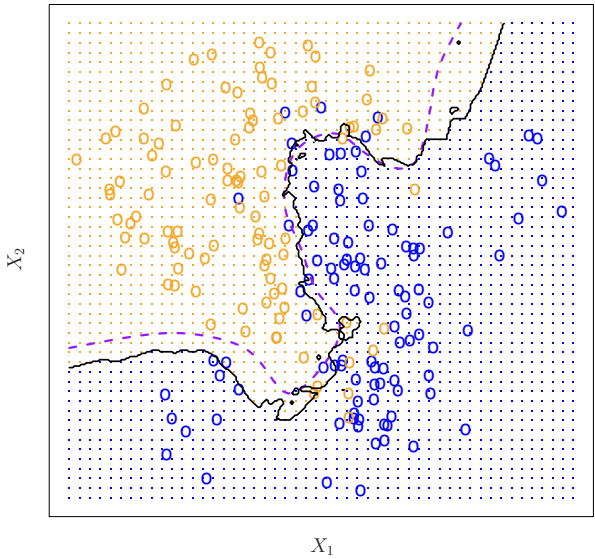
FIGURE 2.15.   *The black curve indicates the KNN decision boundary on the data from Figure 2.13, using $K = 10$. The Bayes decision boundary is shown as a purple dashed line. The KNN and Bayes decision boundaries are very similar.*
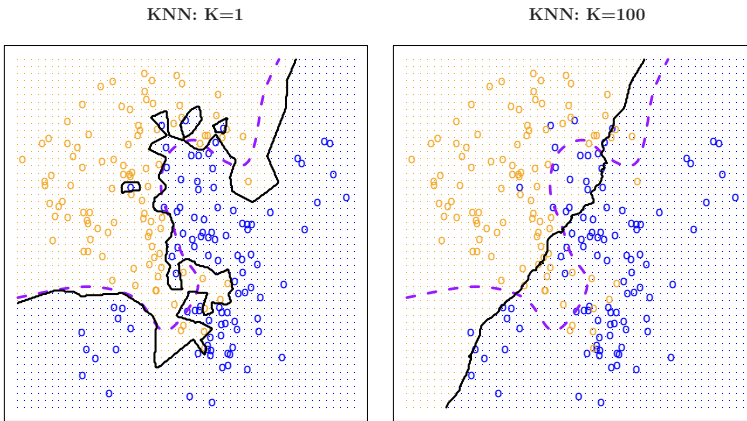
FIGURE 2.16. *A comparison of the KNN decision boundaries (solid black curves) obtained using $K = 1$ and $K = 100$ on the data from Figure 2.13. With $K = 1$, the decision boundary is overly flexible, while with $K = 100$ it is not sufficiently flexible. The Bayes decision boundary is shown as a purple dashed line.*

though the true distribution is not known by the KNN classifier, the KNN decision boundary is very close to that of the Bayes classifier. The test error rate using KNN is 0.1363, which is close to the Bayes error rate of 0.1304.

The choice of K has a drastic effect on the KNN classifier obtained. Figure 2.16 displays two KNN fits to the simulated data from Figure 2.13, using $K = 1$ and $K = 100$. When $K = 1$, the decision boundary is overly flexible and finds patterns in the data that don't correspond to the Bayes decision boundary. This corresponds to a classifier that has low bias but very high variance. As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. This corresponds to a low-variance but high-bias classifier. On this simulated data set, neither $K = 1$ nor $K = 100$ give good predictions: they have test error rates of 0.1695 and 0.1925, respectively.

Just as in the regression setting, there is not a strong relationship between the training error rate and the test error rate. With $K = 1$, the KNN training error rate is 0, but the test error rate may be quite high. In general, as we use more flexible classification methods, the training error rate will decline but the test error rate may not. In Figure 2.17, we have plotted the KNN test and training errors as a function of $1/K$. As $1/K$ increases, the method becomes more flexible. As in the regression setting, the training error rate consistently declines as the flexibility increases. However, the test error exhibits a characteristic U-shape, declining at first (with a minimum at approximately $K = 10$) before increasing again when the method becomes excessively flexible and overfits.
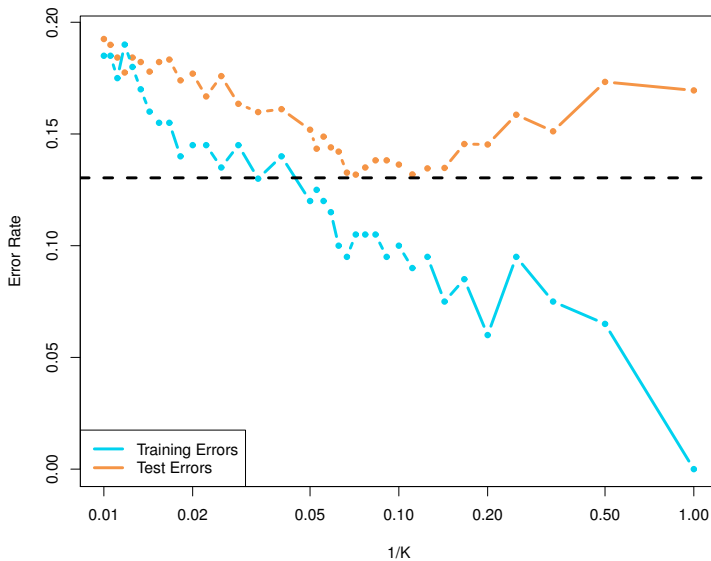
FIGURE 2.17. *The KNN training error rate (blue, 200 observations) and test error rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of flexibility (assessed using $1/K$ on the log scale) increases, or equivalently as the number of neighbors K decreases. The black dashed line 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

<div align="center">

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

</div>

We recommend that you run `R` within an integrated development environment (IDE) such as `RStudio`, which can be freely downloaded from

<div align="center">

http://rstudio.com

</div>

The `RStudio` website also provides a cloud-based version of `R`, which does not require installing any software.

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` 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.

<div style="text-align: right">function
argument

c()

vector</div>

```
> 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.

<div style="text-align: right">length()</div>

```
> 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.

<div style="text-align: right">ls()
rm()</div>

```
> 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 `x2` 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.

```
> 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(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, as new versions of `R` become available, small discrepancies may arise between this 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.

```
> 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`.

```
> 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 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.

`image()`

`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 `ISLR2` library, discussed in Chapter 3. To illustrate the `read.table()` function, we load it now from a text file, `Auto.data`, which you can find on the textbook website. 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*.

data frame

Once the data has been loaded, the `View()` function can be used to view it in a spreadsheet-like window.[1] The `head()` function can also be used to view the first few rows of the data.

```
> Auto <- read.table("Auto.data")
> View(Auto)
> head(Auto)
    V1       V2           V3          V4     V5
1  mpg cylinders displacement horsepower weight
2 18.0         8        307.0      130.0  3504.
3 15.0         8        350.0      165.0  3693.
4 18.0         8        318.0      150.0  3436.
5 16.0         8        304.0      150.0  3433.
6 17.0         8        302.0      140.0  3449.
           V6   V7    V8                            V9
1 acceleration year origin                        name
2         12.0   70     1 chevrolet chevelle malibu
3         11.5   70     1           buick skylark 320
4         11.0   70     1           plymouth satellite
5         12.0   70     1                amc rebel sst
6         10.5   70     1                  ford torino
```

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 = "?",
    stringsAsFactors = T)
> View(Auto)
```

The `stringsAsFactors = T` argument tells `R` that any variable containing character strings should be interpreted as a qualitative variable, and that each distinct character string represents a distinct level for that qualitative variable. An easy way to load data from Excel into `R` is to save it as a csv (comma-separated values) file, and then use the `read.csv()` function.

```
> Auto <- read.csv("Auto.csv", na.strings = "?", stringsAsFactors =
    T)
```

---

[1]This function can sometimes be a bit finicky. If you have trouble using it, then try the `head()` function instead.

```
> View(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 qualitative, 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. We can also produce scatterplots for just a subset of the variables.

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

In conjunction with the `plot()` function, `identify()` provides a useful interactive method for identifying the value of 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 one or more points in the plot and hitting Escape will cause R to print the values of the variable of interest. 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, if we wish.

`q()`
workspace

`savehistory()`
`loadhistory()`

2.4   Exercises

Conceptual

1. For each of parts (a) through (d), indicate whether we would generally expect the performance of a flexible statistical learning method to be better or worse than an inflexible method. Justify your answer.

   (a) The sample size n is extremely large, and the number of predictors p is small.

   (b) The number of predictors p is extremely large, and the number of observations n is small.

   (c) The relationship between the predictors and response is highly non-linear.

   (d) The variance of the error terms, i.e. $\sigma^2 = \text{Var}(\epsilon)$, is extremely high.

2. Explain whether each scenario is a classification or regression problem, and indicate whether we are most interested in inference or prediction. Finally, provide n and p.

(a) We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.

(b) We are considering launching a new product and wish to know whether it will be a *success* or a *failure*. We collect data on 20 similar products that were previously launched. For each product we have recorded whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.

(c) We are interested in predicting the % change in the USD/Euro exchange rate in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012. For each week we record the % change in the USD/Euro, the % change in the US market, the % change in the British market, and the % change in the German market.

3. We now revisit the bias-variance decomposition.

(a) Provide a sketch of typical (squared) bias, variance, training error, test error, and Bayes (or irreducible) error curves, on a single plot, as we go from less flexible statistical learning methods towards more flexible approaches. The x-axis should represent the amount of flexibility in the method, and the y-axis should represent the values for each curve. There should be five curves. Make sure to label each one.

(b) Explain why each of the five curves has the shape displayed in part (a).

4. You will now think of some real-life applications for statistical learning.

(a) Describe three real-life applications in which *classification* might be useful. Describe the response, as well as the predictors. Is the goal of each application inference or prediction? Explain your answer.

(b) Describe three real-life applications in which *regression* might be useful. Describe the response, as well as the predictors. Is the goal of each application inference or prediction? Explain your answer.

(c) Describe three real-life applications in which *cluster analysis* might be useful.

5. What are the advantages and disadvantages of a very flexible (versus a less flexible) approach for regression or classification? Under what

circumstances might a more flexible approach be preferred to a less flexible approach? When might a less flexible approach be preferred?

6. Describe the differences between a parametric and a non-parametric statistical learning approach. What are the advantages of a parametric approach to regression or classification (as opposed to a non-parametric approach)? What are its disadvantages?

7. The table below provides a training data set containing six observations, three predictors, and one qualitative response variable.

Obs.	X_1	X_2	X_3	Y
1	0	3	0	Red
2	2	0	0	Red
3	0	1	3	Red
4	0	1	2	Green
5	−1	0	1	Green
6	1	1	1	Red

Suppose we wish to use this data set to make a prediction for Y when $X_1 = X_2 = X_3 = 0$ using K-nearest neighbors.

(a) Compute the Euclidean distance between each observation and the test point, $X_1 = X_2 = X_3 = 0$.

(b) What is our prediction with $K = 1$? Why?

(c) What is our prediction with $K = 3$? Why?

(d) If the Bayes decision boundary in this problem is highly non-linear, then would we expect the *best* value for K to be large or small? Why?

Applied

8. This exercise relates to the `College` data set, which can be found in the file `College.csv` on the book website. It contains a number of variables for 777 different universities and colleges in the US. The variables are

- `Private` : Public/private indicator
- `Apps` : Number of applications received
- `Accept` : Number of applicants accepted
- `Enroll` : Number of new students enrolled
- `Top10perc` : New students from top 10 % of high school class
- `Top25perc` : New students from top 25 % of high school class

- `F.Undergrad` : Number of full-time undergraduates
- `P.Undergrad` : Number of part-time undergraduates
- `Outstate` : Out-of-state tuition
- `Room.Board` : Room and board costs
- `Books` : Estimated book costs
- `Personal` : Estimated personal spending
- `PhD` : Percent of faculty with Ph.D.'s
- `Terminal` : Percent of faculty with terminal degree
- `S.F.Ratio` : Student/faculty ratio
- `perc.alumni` : Percent of alumni who donate
- `Expend` : Instructional expenditure per student
- `Grad.Rate` : Graduation rate

Before reading the data into `R`, it can be viewed in Excel or a text editor.

(a) Use the `read.csv()` function to read the data into `R`. Call the loaded data `college`. Make sure that you have the directory set to the correct location for the data.

(b) Look at the data using the `View()` function. You should notice that the first column is just the name of each university. We don't really want `R` to treat this as data. However, it may be handy to have these names for later. Try the following commands:

```
> rownames(college) <- college[, 1]
> View(college)
```

You should see that there is now a `row.names` column with the name of each university recorded. This means that `R` has given each row a name corresponding to the appropriate university. `R` will not try to perform calculations on the row names. However, we still need to eliminate the first column in the data where the names are stored. Try

```
> college <- college[, -1]
> View(college)
```

Now you should see that the first data column is `Private`. Note that another column labeled `row.names` now appears before the `Private` column. However, this is not a data column but rather the name that `R` is giving to each row.

(c)   i. Use the `summary()` function to produce a numerical summary of the variables in the data set.

      ii. Use the `pairs()` function to produce a scatterplot matrix of the first ten columns or variables of the data. Recall that you can reference the first ten columns of a matrix `A` using `A[,1:10]`.

     iii. Use the `plot()` function to produce side-by-side boxplots of `Outstate` versus `Private`.

     iv. Create a new qualitative variable, called `Elite`, by *binning* the `Top10perc` variable. We are going to divide universities into two groups based on whether or not the proportion of students coming from the top 10 % of their high school classes exceeds 50 %.

```
> Elite <- rep("No", nrow(college))
> Elite[college$Top10perc > 50] <- "Yes"
> Elite <- as.factor(Elite)
> college <- data.frame(college, Elite)
```

      Use the `summary()` function to see how many elite universities there are. Now use the `plot()` function to produce side-by-side boxplots of `Outstate` versus `Elite`.

      v. Use the `hist()` function to produce some histograms with differing numbers of bins for a few of the quantitative variables. You may find the command `par(mfrow = c(2, 2))` useful: it will divide the print window into four regions so that four plots can be made simultaneously. Modifying the arguments to this function will divide the screen in other ways.

     vi. Continue exploring the data, and provide a brief summary of what you discover.

9. This exercise involves the `Auto` data set studied in the lab. Make sure that the missing values have been removed from the data.

   (a) Which of the predictors are quantitative, and which are qualitative?

   (b) What is the *range* of each quantitative predictor? You can answer this using the `range()` function.

        `range()`

   (c) What is the mean and standard deviation of each quantitative predictor?

   (d) Now remove the 10th through 85th observations. What is the range, mean, and standard deviation of each predictor in the subset of the data that remains?

   (e) Using the full data set, investigate the predictors graphically, using scatterplots or other tools of your choice. Create some plots highlighting the relationships among the predictors. Comment on your findings.

(f) Suppose that we wish to predict gas mileage (`mpg`) on the basis of the other variables. Do your plots suggest that any of the other variables might be useful in predicting `mpg`? Justify your answer.

10. This exercise involves the `Boston` housing data set.

   (a) To begin, load in the `Boston` data set. The `Boston` data set is part of the `ISLR2` *library*.

   ```
   > library(ISLR2)
   ```

   Now the data set is contained in the object `Boston`.

   ```
   > Boston
   ```

   Read about the data set:

   ```
   > ?Boston
   ```

   How many rows are in this data set? How many columns? What do the rows and columns represent?

   (b) Make some pairwise scatterplots of the predictors (columns) in this data set. Describe your findings.

   (c) Are any of the predictors associated with per capita crime rate? If so, explain the relationship.

   (d) Do any of the census tracts of Boston appear to have particularly high crime rates? Tax rates? Pupil-teacher ratios? Comment on the range of each predictor.

   (e) How many of the census tracts in this data set bound the Charles river?

   (f) What is the median pupil-teacher ratio among the towns in this data set?

   (g) Which census tract of Boston has lowest median value of owner-occupied homes? What are the values of the other predictors for that census tract, and how do those values compare to the overall ranges for those predictors? Comment on your findings.

   (h) In this data set, how many of the census tracts average more than seven rooms per dwelling? More than eight rooms per dwelling? Comment on the census tracts that average more than eight rooms per dwelling.