

5

Resampling Methods

Resampling methods are an indispensable tool in modern statistics. They involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model. For example, in order to estimate the variability of a linear regression fit, we can repeatedly draw different samples from the training data, fit a linear regression to each new sample, and then examine the extent to which the resulting fits differ. Such an approach may allow us to obtain information that would not be available from fitting the model only once using the original training sample.

Resampling approaches can be computationally expensive, because they involve fitting the same statistical method multiple times using different subsets of the training data. However, due to recent advances in computing power, the computational requirements of resampling methods generally are not prohibitive. In this chapter, we discuss two of the most commonly used resampling methods, *cross-validation* and the *bootstrap*. Both methods are important tools in the practical application of many statistical learning procedures. For example, cross-validation can be used to estimate the test error associated with a given statistical learning method in order to evaluate its performance, or to select the appropriate level of flexibility. The process of evaluating a model's performance is known as *model assessment*, whereas the process of selecting the proper level of flexibility for a model is known as *model selection*. The bootstrap is used in several contexts, most commonly to provide a measure of accuracy of a parameter estimate or of a given statistical learning method.

model
assessment

model
selection

5.1 Cross-Validation

In Chapter 2 we discuss the distinction between the *test error rate* and the *training error rate*. The test error is the average error that results from using a statistical learning method to predict the response on a new observation—that is, a measurement that was not used in training the method. Given a data set, the use of a particular statistical learning method is warranted if it results in a low test error. The test error can be easily calculated if a designated test set is available. Unfortunately, this is usually not the case. In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training. But as we saw in Chapter 2, the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

In the absence of a very large designated test set that can be used to directly estimate the test error rate, a number of techniques can be used to estimate this quantity using the available training data. Some methods make a mathematical adjustment to the training error rate in order to estimate the test error rate. Such approaches are discussed in Chapter 6. In this section, we instead consider a class of methods that estimate the test error rate by *holding out* a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.

In Sections 5.1.1–5.1.4, for simplicity we assume that we are interested in performing regression with a quantitative response. In Section 5.1.5 we consider the case of classification with a qualitative response. As we will see, the key concepts remain the same regardless of whether the response is quantitative or qualitative.

5.1.1 The Validation Set Approach

Suppose that we would like to estimate the test error associated with fitting a particular statistical learning method on a set of observations. The *validation set approach*, displayed in Figure 5.1, is a very simple strategy for this task. It involves randomly dividing the available set of observations into two parts, a *training set* and a *validation set* or *hold-out set*. The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set. The resulting validation set error rate—typically assessed using MSE in the case of a quantitative response—provides an estimate of the test error rate.

validation
set approach

validation
set
hold-out set

We illustrate the validation set approach on the **Auto** data set. Recall from Chapter 3 that there appears to be a non-linear relationship between **mpg** and **horsepower**, and that a model that predicts **mpg** using **horsepower** and **horsepower**² gives better results than a model that uses only a linear term. It is natural to wonder whether a cubic or higher-order fit might provide

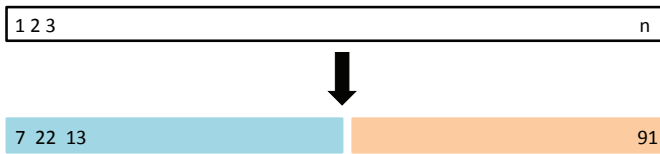


FIGURE 5.1. A schematic display of the validation set approach. A set of n observations are randomly split into a training set (shown in blue, containing observations 7, 22, and 13, among others) and a validation set (shown in beige, and containing observation 91, among others). The statistical learning method is fit on the training set, and its performance is evaluated on the validation set.

even better results. We answer this question in Chapter 3 by looking at the p-values associated with a cubic term and higher-order polynomial terms in a linear regression. But we could also answer this question using the validation method. We randomly split the 392 observations into two sets, a training set containing 196 of the data points, and a validation set containing the remaining 196 observations. The validation set error rates that result from fitting various regression models on the training sample and evaluating their performance on the validation sample, using MSE as a measure of validation set error, are shown in the left-hand panel of Figure 5.2. The validation set MSE for the quadratic fit is considerably smaller than for the linear fit. However, the validation set MSE for the cubic fit is actually slightly larger than for the quadratic fit. This implies that including a cubic term in the regression does not lead to better prediction than simply using a quadratic term.

Recall that in order to create the left-hand panel of Figure 5.2, we randomly divided the data set into two parts, a training set and a validation set. If we repeat the process of randomly splitting the sample set into two parts, we will get a somewhat different estimate for the test MSE. As an illustration, the right-hand panel of Figure 5.2 displays ten different validation set MSE curves from the **Auto** data set, produced using ten different random splits of the observations into training and validation sets. All ten curves indicate that the model with a quadratic term has a dramatically smaller validation set MSE than the model with only a linear term. Furthermore, all ten curves indicate that there is not much benefit in including cubic or higher-order polynomial terms in the model. But it is worth noting that each of the ten curves results in a different test MSE estimate for each of the ten regression models considered. And there is no consensus among the curves as to which model results in the smallest validation set MSE. Based on the variability among these curves, all that we can conclude with any confidence is that the linear fit is not adequate for this data.

The validation set approach is conceptually simple and is easy to implement. But it has two potential drawbacks:

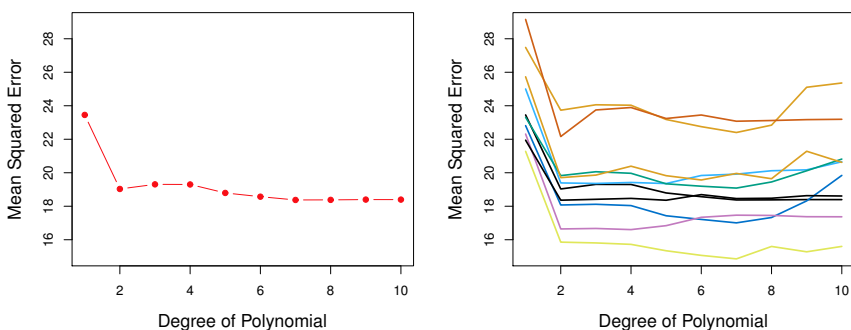


FIGURE 5.2. The validation set approach was used on the `Auto` data set in order to estimate the test error that results from predicting `mpg` using polynomial functions of `horsepower`. Left: Validation error estimates for a single split into training and validation data sets. Right: The validation method was repeated ten times, each time using a different random split of the observations into a training set and a validation set. This illustrates the variability in the estimated test MSE that results from this approach.

1. As is shown in the right-hand panel of Figure 5.2, the validation estimate of the test error rate can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
2. In the validation approach, only a subset of the observations—those that are included in the training set rather than in the validation set—are used to fit the model. Since statistical methods tend to perform worse when trained on fewer observations, this suggests that the validation set error rate may tend to *overestimate* the test error rate for the model fit on the entire data set.

In the coming subsections, we will present *cross-validation*, a refinement of the validation set approach that addresses these two issues.

5.1.2 Leave-One-Out Cross-Validation

Leave-one-out cross-validation (LOOCV) is closely related to the validation set approach of Section 5.1.1, but it attempts to address that method's

leave-one-out
cross-validation

drawbacks. Like the validation set approach, LOOCV involves splitting the set of observations into two parts. However, instead of creating two subsets of comparable size, a single observation (x_1, y_1) is used for the validation set, and the remaining observations $\{(x_2, y_2), \dots, (x_n, y_n)\}$ make up the training set. The statistical learning method is fit on the $n - 1$ training observations, and a prediction \hat{y}_1 is made for the excluded observation,

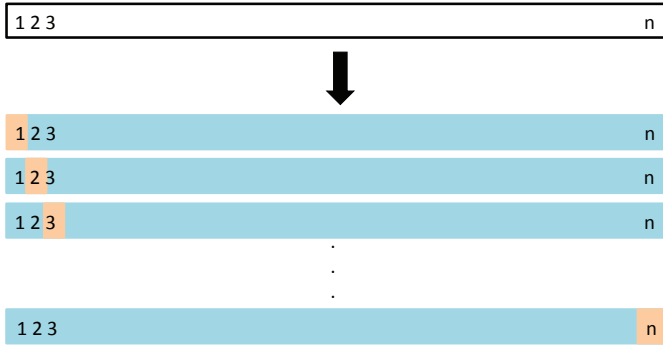


FIGURE 5.3. A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSEs. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.

using its value x_1 . Since (x_1, y_1) was not used in the fitting process, $\text{MSE}_1 = (y_1 - \hat{y}_1)^2$ provides an approximately unbiased estimate for the test error. But even though MSE_1 is unbiased for the test error, it is a poor estimate because it is highly variable, since it is based upon a single observation (x_1, y_1) .

We can repeat the procedure by selecting (x_2, y_2) for the validation data, training the statistical learning procedure on the $n - 1$ observations $\{(x_1, y_1), (x_3, y_3), \dots, (x_n, y_n)\}$, and computing $\text{MSE}_2 = (y_2 - \hat{y}_2)^2$. Repeating this approach n times produces n squared errors, $\text{MSE}_1, \dots, \text{MSE}_n$. The LOOCV estimate for the test MSE is the average of these n test error estimates:

$$\text{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^n \text{MSE}_i. \quad (5.1)$$

A schematic of the LOOCV approach is illustrated in Figure 5.3.

LOOCV has a couple of major advantages over the validation set approach. First, it has far less bias. In LOOCV, we repeatedly fit the statistical learning method using training sets that contain $n - 1$ observations, almost as many as are in the entire data set. This is in contrast to the validation set approach, in which the training set is typically around half the size of the original data set. Consequently, the LOOCV approach tends not to overestimate the test error rate as much as the validation set approach does. Second, in contrast to the validation approach which will yield different results when applied repeatedly due to randomness in the training/validation set splits, performing LOOCV multiple times will

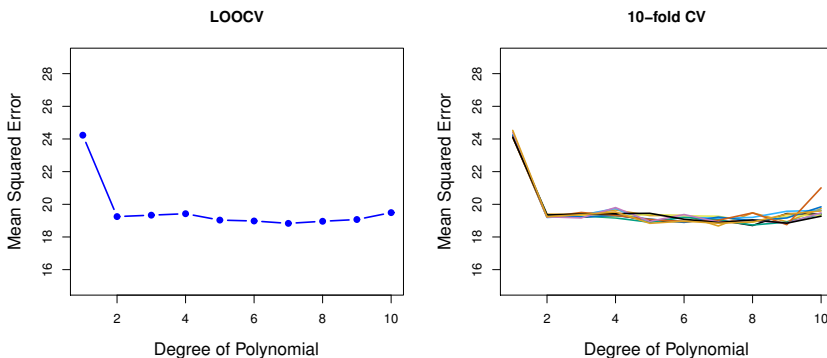


FIGURE 5.4. Cross-validation was used on the **Auto** data set in order to estimate the test error that results from predicting **mpg** using polynomial functions of **horsepower**. Left: The LOOCV error curve. Right: 10-fold CV was run nine separate times, each with a different random split of the data into ten parts. The figure shows the nine slightly different CV error curves.

always yield the same results: there is no randomness in the training/validation set splits.

We used LOOCV on the **Auto** data set in order to obtain an estimate of the test set MSE that results from fitting a linear regression model to predict **mpg** using polynomial functions of **horsepower**. The results are shown in the left-hand panel of Figure 5.4.

LOOCV has the potential to be expensive to implement, since the model has to be fit n times. This can be very time consuming if n is large, and if each individual model is slow to fit. With least squares linear or polynomial regression, an amazing shortcut makes the cost of LOOCV the same as that of a single model fit! The following formula holds:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2, \quad (5.2)$$

where \hat{y}_i is the i th fitted value from the original least squares fit, and h_i is the leverage defined in (3.37) on page 99.¹ This is like the ordinary MSE, except the i th residual is divided by $1 - h_i$. The leverage lies between $1/n$ and 1, and reflects the amount that an observation influences its own fit. Hence the residuals for high-leverage points are inflated in this formula by exactly the right amount for this equality to hold.

LOOCV is a very general method, and can be used with any kind of predictive modeling. For example we could use it with logistic regression

¹In the case of multiple linear regression, the leverage takes a slightly more complicated form than (3.37), but (5.2) still holds.

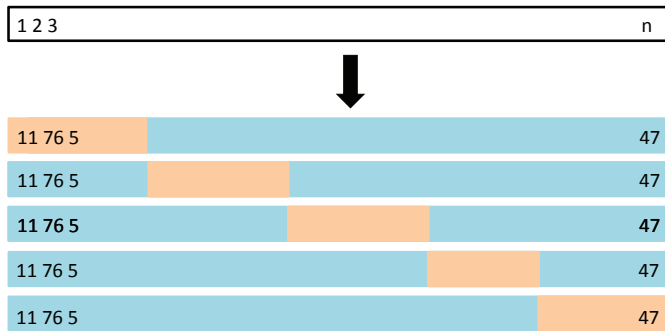


FIGURE 5.5. A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.

or linear discriminant analysis, or any of the methods discussed in later chapters. The magic formula (5.2) does not hold in general, in which case the model has to be refit n times.

5.1.3 k -Fold Cross-Validation

An alternative to LOOCV is k -fold CV. This approach involves randomly dividing the set of observations into k groups, or *folds*, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k - 1$ folds. The mean squared error, MSE_1 , is then computed on the observations in the held-out fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error, $\text{MSE}_1, \text{MSE}_2, \dots, \text{MSE}_k$. The k -fold CV estimate is computed by averaging these values,

k -fold CV

$$\text{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^k \text{MSE}_i. \quad (5.3)$$

Figure 5.5 illustrates the k -fold CV approach.

It is not hard to see that LOOCV is a special case of k -fold CV in which k is set to equal n . In practice, one typically performs k -fold CV using $k = 5$ or $k = 10$. What is the advantage of using $k = 5$ or $k = 10$ rather than $k = n$? The most obvious advantage is computational. LOOCV requires fitting the statistical learning method n times. This has the potential to be computationally expensive (except for linear models fit by least squares, in which case formula (5.2) can be used). But cross-validation is a very general approach that can be applied to almost any statistical learning method. Some statistical learning methods have computationally intensive

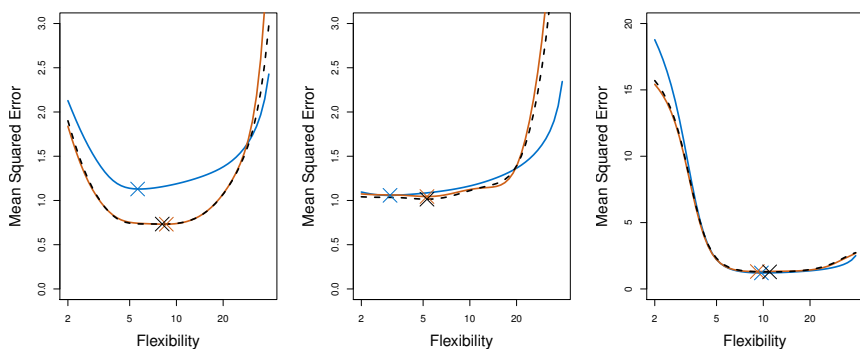


FIGURE 5.6. True and estimated test MSE for the simulated data sets in Figures 2.9 (left), 2.10 (center), and 2.11 (right). The true test MSE is shown in blue, the LOOCV estimate is shown as a black dashed line, and the 10-fold CV estimate is shown in orange. The crosses indicate the minimum of each of the MSE curves.

fitting procedures, and so performing LOOCV may pose computational problems, especially if n is extremely large. In contrast, performing 10-fold CV requires fitting the learning procedure only ten times, which may be much more feasible. As we see in Section 5.1.4, there also can be other non-computational advantages to performing 5-fold or 10-fold CV, which involve the bias-variance trade-off.

The right-hand panel of Figure 5.4 displays nine different 10-fold CV estimates for the `Auto` data set, each resulting from a different random split of the observations into ten folds. As we can see from the figure, there is some variability in the CV estimates as a result of the variability in how the observations are divided into ten folds. But this variability is typically much lower than the variability in the test error estimates that results from the validation set approach (right-hand panel of Figure 5.2).

When we examine real data, we do not know the *true* test MSE, and so it is difficult to determine the accuracy of the cross-validation estimate. However, if we examine simulated data, then we can compute the true test MSE, and can thereby evaluate the accuracy of our cross-validation results. In Figure 5.6, we plot the cross-validation estimates and true test error rates that result from applying smoothing splines to the simulated data sets illustrated in Figures 2.9–2.11 of Chapter 2. The true test MSE is displayed in blue. The black dashed and orange solid lines respectively show the estimated LOOCV and 10-fold CV estimates. In all three plots, the two cross-validation estimates are very similar. In the right-hand panel of Figure 5.6, the true test MSE and the cross-validation curves are almost identical. In the center panel of Figure 5.6, the two sets of curves are similar at the lower degrees of flexibility, while the CV curves overestimate the test set MSE for higher degrees of flexibility. In the left-hand panel of Figure 5.6,

the CV curves have the correct general shape, but they underestimate the true test MSE.

When we perform cross-validation, our goal might be to determine how well a given statistical learning procedure can be expected to perform on independent data; in this case, the actual estimate of the test MSE is of interest. But at other times we are interested only in the location of the *minimum point in the estimated test MSE curve*. This is because we might be performing cross-validation on a number of statistical learning methods, or on a single method using different levels of flexibility, in order to identify the method that results in the lowest test error. For this purpose, the location of the minimum point in the estimated test MSE curve is important, but the actual value of the estimated test MSE is not. We find in Figure 5.6 that despite the fact that they sometimes underestimate the true test MSE, all of the CV curves come close to identifying the correct level of flexibility—that is, the flexibility level corresponding to the smallest test MSE.

5.1.4 Bias-Variance Trade-Off for k -Fold Cross-Validation

We mentioned in Section 5.1.3 that k -fold CV with $k < n$ has a computational advantage to LOOCV. But putting computational issues aside, a less obvious but potentially more important advantage of k -fold CV is that it often gives more accurate estimates of the test error rate than does LOOCV. This has to do with a bias-variance trade-off.

It was mentioned in Section 5.1.1 that the validation set approach can lead to overestimates of the test error rate, since in this approach the training set used to fit the statistical learning method contains only half the observations of the entire data set. Using this logic, it is not hard to see that LOOCV will give approximately unbiased estimates of the test error, since each training set contains $n - 1$ observations, which is almost as many as the number of observations in the full data set. And performing k -fold CV for, say, $k = 5$ or $k = 10$ will lead to an intermediate level of bias, since each training set contains approximately $(k - 1)n/k$ observations—fewer than in the LOOCV approach, but substantially more than in the validation set approach. Therefore, from the perspective of bias reduction, it is clear that LOOCV is to be preferred to k -fold CV.

However, we know that bias is not the only source for concern in an estimating procedure; we must also consider the procedure's variance. It turns out that LOOCV has higher variance than does k -fold CV with $k < n$. Why is this the case? When we perform LOOCV, we are in effect averaging the outputs of n fitted models, each of which is trained on an almost identical set of observations; therefore, these outputs are highly (positively) correlated with each other. In contrast, when we perform k -fold CV with $k < n$, we are averaging the outputs of k fitted models that are somewhat less correlated with each other, since the overlap between the training sets in

each model is smaller. Since the mean of many highly correlated quantities has higher variance than does the mean of many quantities that are not as highly correlated, the test error estimate resulting from LOOCV tends to have higher variance than does the test error estimate resulting from k -fold CV.

To summarize, there is a bias-variance trade-off associated with the choice of k in k -fold cross-validation. Typically, given these considerations, one performs k -fold cross-validation using $k = 5$ or $k = 10$, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

5.1.5 Cross-Validation on Classification Problems

In this chapter so far, we have illustrated the use of cross-validation in the regression setting where the outcome Y is quantitative, and so have used MSE to quantify test error. But cross-validation can also be a very useful approach in the classification setting when Y is qualitative. In this setting, cross-validation works just as described earlier in this chapter, except that rather than using MSE to quantify test error, we instead use the number of misclassified observations. For instance, in the classification setting, the LOOCV error rate takes the form

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n \text{Err}_i, \quad (5.4)$$

where $\text{Err}_i = I(y_i \neq \hat{y}_i)$. The k -fold CV error rate and validation set error rates are defined analogously.

As an example, we fit various logistic regression models on the two-dimensional classification data displayed in Figure 2.13. In the top-left panel of Figure 5.7, the black solid line shows the estimated decision boundary resulting from fitting a standard logistic regression model to this data set. Since this is simulated data, we can compute the *true* test error rate, which takes a value of 0.201 and so is substantially larger than the Bayes error rate of 0.133. Clearly logistic regression does not have enough flexibility to model the Bayes decision boundary in this setting. We can easily extend logistic regression to obtain a non-linear decision boundary by using polynomial functions of the predictors, as we did in the regression setting in Section 3.3.2. For example, we can fit a *quadratic* logistic regression model, given by

$$\log \left(\frac{p}{1-p} \right) = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_2^2. \quad (5.5)$$

The top-right panel of Figure 5.7 displays the resulting decision boundary, which is now curved. However, the test error rate has improved only slightly,

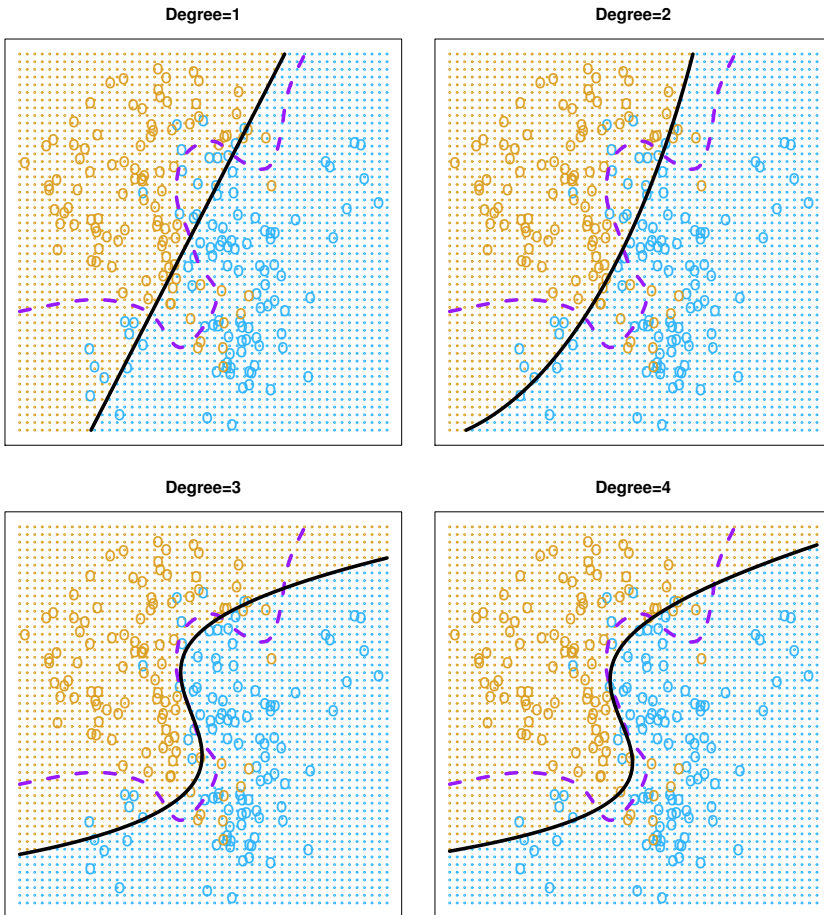


FIGURE 5.7. Logistic regression fits on the two-dimensional classification data displayed in Figure 2.13. The Bayes decision boundary is represented using a purple dashed line. Estimated decision boundaries from linear, quadratic, cubic and quartic (degrees 1–4) logistic regressions are displayed in black. The test error rates for the four logistic regression fits are respectively 0.201, 0.197, 0.160, and 0.162, while the Bayes error rate is 0.133.

to 0.197. A much larger improvement is apparent in the bottom-left panel of Figure 5.7, in which we have fit a logistic regression model involving cubic polynomials of the predictors. Now the test error rate has decreased to 0.160. Going to a quartic polynomial (bottom-right) slightly increases the test error.

In practice, for real data, the Bayes decision boundary and the test error rates are unknown. So how might we decide between the four logistic regression models displayed in Figure 5.7? We can use cross-validation in

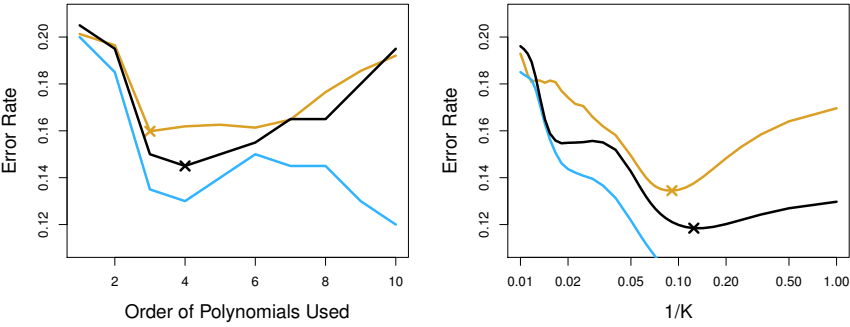


FIGURE 5.8. Test error (brown), training error (blue), and 10-fold CV error (black) on the two-dimensional classification data displayed in Figure 5.7. Left: Logistic regression using polynomial functions of the predictors. The order of the polynomials used is displayed on the x-axis. Right: The KNN classifier with different values of K , the number of neighbors used in the KNN classifier.

order to make this decision. The left-hand panel of Figure 5.8 displays in black the 10-fold CV error rates that result from fitting ten logistic regression models to the data, using polynomial functions of the predictors up to tenth order. The true test errors are shown in brown, and the training errors are shown in blue. As we have seen previously, the training error tends to decrease as the flexibility of the fit increases. (The figure indicates that though the training error rate doesn't quite decrease monotonically, it tends to decrease on the whole as the model complexity increases.) In contrast, the test error displays a characteristic U-shape. The 10-fold CV error rate provides a pretty good approximation to the test error rate. While it somewhat underestimates the error rate, it reaches a minimum when fourth-order polynomials are used, which is very close to the minimum of the test curve, which occurs when third-order polynomials are used. In fact, using fourth-order polynomials would likely lead to good test set performance, as the true test error rate is approximately the same for third, fourth, fifth, and sixth-order polynomials.

The right-hand panel of Figure 5.8 displays the same three curves using the KNN approach for classification, as a function of the value of K (which in this context indicates the number of neighbors used in the KNN classifier, rather than the number of CV folds used). Again the training error rate declines as the method becomes more flexible, and so we see that the training error rate cannot be used to select the optimal value for K . Though the cross-validation error curve slightly underestimates the test error rate, it takes on a minimum very close to the best value for K .

5.2 The Bootstrap

The *bootstrap* is a widely applicable and extremely powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method. As a simple example, the bootstrap can be used to estimate the standard errors of the coefficients from a linear regression fit. In the specific case of linear regression, this is not particularly useful, since we saw in Chapter 3 that standard statistical software such as **R** outputs such standard errors automatically. However, the power of the bootstrap lies in the fact that it can be easily applied to a wide range of statistical learning methods, including some for which a measure of variability is otherwise difficult to obtain and is not automatically output by statistical software.

bootstrap

In this section we illustrate the bootstrap on a toy example in which we wish to determine the best investment allocation under a simple model. In Section 5.3 we explore the use of the bootstrap to assess the variability associated with the regression coefficients in a linear model fit.

Suppose that we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y , respectively, where X and Y are random quantities. We will invest a fraction α of our money in X , and will invest the remaining $1 - \alpha$ in Y . Since there is variability associated with the returns on these two assets, we wish to choose α to minimize the total risk, or variance, of our investment. In other words, we want to minimize $\text{Var}(\alpha X + (1 - \alpha)Y)$. One can show that the value that minimizes the risk is given by

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}, \quad (5.6)$$

where $\sigma_X^2 = \text{Var}(X)$, $\sigma_Y^2 = \text{Var}(Y)$, and $\sigma_{XY} = \text{Cov}(X, Y)$.

In reality, the quantities σ_X^2 , σ_Y^2 , and σ_{XY} are unknown. We can compute estimates for these quantities, $\hat{\sigma}_X^2$, $\hat{\sigma}_Y^2$, and $\hat{\sigma}_{XY}$, using a data set that contains past measurements for X and Y . We can then estimate the value of α that minimizes the variance of our investment using

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}. \quad (5.7)$$

Figure 5.9 illustrates this approach for estimating α on a simulated data set. In each panel, we simulated 100 pairs of returns for the investments X and Y . We used these returns to estimate σ_X^2 , σ_Y^2 , and σ_{XY} , which we then substituted into (5.7) in order to obtain estimates for α . The value of $\hat{\alpha}$ resulting from each simulated data set ranges from 0.532 to 0.657.

It is natural to wish to quantify the accuracy of our estimate of α . To estimate the standard deviation of $\hat{\alpha}$, we repeated the process of simulating 100 paired observations of X and Y , and estimating α using (5.7), 1,000 times. We thereby obtained 1,000 estimates for α , which we can call

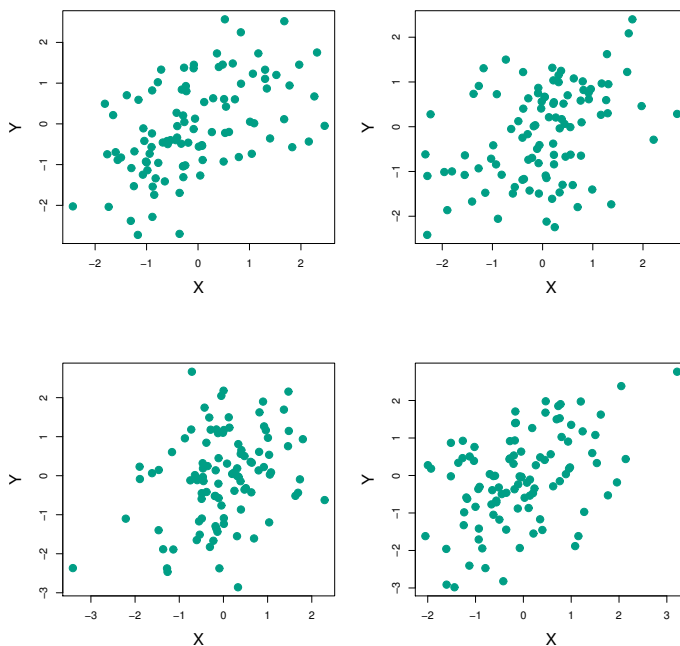


FIGURE 5.9. Each panel displays 100 simulated returns for investments X and Y . From left to right and top to bottom, the resulting estimates for α are 0.576, 0.532, 0.657, and 0.651.

$\hat{\alpha}_1, \hat{\alpha}_2, \dots, \hat{\alpha}_{1,000}$. The left-hand panel of Figure 5.10 displays a histogram of the resulting estimates. For these simulations the parameters were set to $\sigma_X^2 = 1, \sigma_Y^2 = 1.25$, and $\sigma_{XY} = 0.5$, and so we know that the true value of α is 0.6. We indicated this value using a solid vertical line on the histogram. The mean over all 1,000 estimates for α is

$$\bar{\alpha} = \frac{1}{1000} \sum_{r=1}^{1000} \hat{\alpha}_r = 0.5996,$$

very close to $\alpha = 0.6$, and the standard deviation of the estimates is

$$\sqrt{\frac{1}{1000-1} \sum_{r=1}^{1000} (\hat{\alpha}_r - \bar{\alpha})^2} = 0.083.$$

This gives us a very good idea of the accuracy of $\hat{\alpha}$: $\text{SE}(\hat{\alpha}) \approx 0.083$. So roughly speaking, for a random sample from the population, we would expect $\hat{\alpha}$ to differ from α by approximately 0.08, on average.

In practice, however, the procedure for estimating $\text{SE}(\hat{\alpha})$ outlined above cannot be applied, because for real data we cannot generate new samples

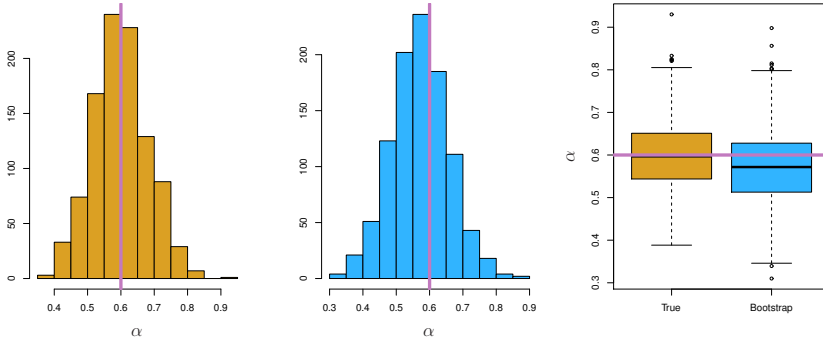


FIGURE 5.10. Left: A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population. Center: A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set. Right: The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α .

from the original population. However, the bootstrap approach allows us to use a computer to emulate the process of obtaining new sample sets, so that we can estimate the variability of $\hat{\alpha}$ without generating additional samples. Rather than repeatedly obtaining independent data sets from the population, we instead obtain distinct data sets by repeatedly sampling observations *from the original data set*.

This approach is illustrated in Figure 5.11 on a simple data set, which we call Z , that contains only $n = 3$ observations. We randomly select n observations from the data set in order to produce a bootstrap data set, Z^{*1} . The sampling is performed *with replacement*, which means that the same observation can occur more than once in the bootstrap data set. In this example, Z^{*1} contains the third observation twice, the first observation once, and no instances of the second observation. Note that if an observation is contained in Z^{*1} , then both its X and Y values are included. We can use Z^{*1} to produce a new bootstrap estimate for α , which we call $\hat{\alpha}^{*1}$. This procedure is repeated B times for some large value of B , in order to produce B different bootstrap data sets, $Z^{*1}, Z^{*2}, \dots, Z^{*B}$, and B corresponding α estimates, $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \dots, \hat{\alpha}^{*B}$. We can compute the standard error of these bootstrap estimates using the formula

with
replacement

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1} \sum_{r=1}^B \left(\hat{\alpha}^{*r} - \frac{1}{B} \sum_{r'=1}^B \hat{\alpha}^{*r'} \right)^2}. \quad (5.8)$$

This serves as an estimate of the standard error of $\hat{\alpha}$ estimated from the original data set.

The bootstrap approach is illustrated in the center panel of Figure 5.10, which displays a histogram of 1,000 bootstrap estimates of α , each com-

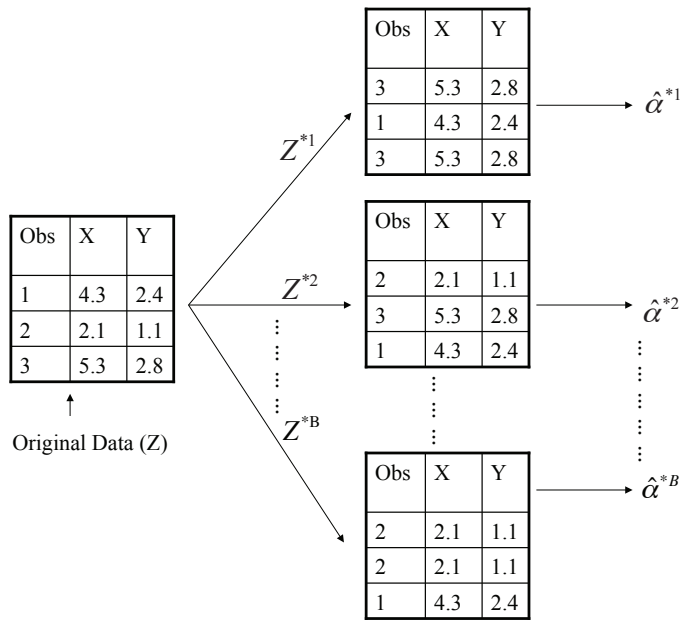


FIGURE 5.11. A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α .

puted using a distinct bootstrap data set. This panel was constructed on the basis of a single data set, and hence could be created using real data. Note that the histogram looks very similar to the left-hand panel, which displays the idealized histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population. In particular the bootstrap estimate $SE(\hat{\alpha})$ from (5.8) is 0.087, very close to the estimate of 0.083 obtained using 1,000 simulated data sets. The right-hand panel displays the information in the center and left panels in a different way, via boxplots of the estimates for α obtained by generating 1,000 simulated data sets from the true population and using the bootstrap approach. Again, the boxplots have similar spreads, indicating that the bootstrap approach can be used to effectively estimate the variability associated with $\hat{\alpha}$.

5.3 Lab: Cross-Validation and the Bootstrap

In this lab, we explore the resampling techniques covered in this chapter. Some of the commands in this lab may take a while to run on your computer.

5.3.1 The Validation Set Approach

We explore the use of the validation set approach in order to estimate the test error rates that result from fitting various linear models on the `Auto` data set.

Before we begin, we use the `set.seed()` function in order to set a *seed* for R's random number generator, so that the reader of this book will obtain precisely the same results as those shown below. It is generally a good idea to set a random seed when performing an analysis such as cross-validation that contains an element of randomness, so that the results obtained can be reproduced precisely at a later time.

We begin by using the `sample()` function to split the set of observations into two halves, by selecting a random subset of 196 observations out of the original 392 observations. We refer to these observations as the training set.

```
> library(ISLR2)
> set.seed(1)
> train <- sample(392, 196)
```

(Here we use a shortcut in the `sample` command; see `?sample` for details.) We then use the `subset` option in `lm()` to fit a linear regression using only the observations corresponding to the training set.

```
> lm.fit <- lm(mpg ~ horsepower, data = Auto, subset = train)
```

We now use the `predict()` function to estimate the response for all 392 observations, and we use the `mean()` function to calculate the MSE of the 196 observations in the validation set. Note that the `-train` index below selects only the observations that are not in the training set.

```
> attach(Auto)
> mean((mpg - predict(lm.fit, Auto))[-train]^2)
[1] 23.27
```

Therefore, the estimated test MSE for the linear regression fit is 23.27. We can use the `poly()` function to estimate the test error for the quadratic and cubic regressions.

```
> lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto,
  subset = train)
> mean((mpg - predict(lm.fit2, Auto))[-train]^2)
[1] 18.72
> lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,
  subset = train)
> mean((mpg - predict(lm.fit3, Auto))[-train]^2)
[1] 18.79
```

These error rates are 18.72 and 18.79, respectively. If we choose a different training set instead, then we will obtain somewhat different errors on the validation set.

```

> set.seed(2)
> train <- sample(392, 196)
> lm.fit <- lm(mpg ~ horsepower, subset = train)
> mean((mpg - predict(lm.fit, Auto))[-train]^2)
[1] 25.73
> lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto,
  subset = train)
> mean((mpg - predict(lm.fit2, Auto))[-train]^2)
[1] 20.43
> lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,
  subset = train)
> mean((mpg - predict(lm.fit3, Auto))[-train]^2)
[1] 20.39

```

Using this split of the observations into a training set and a validation set, we find that the validation set error rates for the models with linear, quadratic, and cubic terms are 25.73, 20.43, and 20.39, respectively.

These results are consistent with our previous findings: a model that predicts `mpg` using a quadratic function of `horsepower` performs better than a model that involves only a linear function of `horsepower`, and there is little evidence in favor of a model that uses a cubic function of `horsepower`.

5.3.2 Leave-One-Out Cross-Validation

The LOOCV estimate can be automatically computed for any generalized linear model using the `glm()` and `cv.glm()` functions. In the lab for Chapter 4, we used the `glm()` function to perform logistic regression by passing in the `family = "binomial"` argument. But if we use `glm()` to fit a model without passing in the `family` argument, then it performs linear regression, just like the `lm()` function. So for instance,

`cv.glm()`

```

> glm.fit <- glm(mpg ~ horsepower, data = Auto)
> coef(glm.fit)
(Intercept)  horsepower
    39.936      -0.158

```

and

```

> lm.fit <- lm(mpg ~ horsepower, data = Auto)
> coef(lm.fit)
(Intercept)  horsepower
    39.936      -0.158

```

yield identical linear regression models. In this lab, we will perform linear regression using the `glm()` function rather than the `lm()` function because the former can be used together with `cv.glm()`. The `cv.glm()` function is part of the `boot` library.

```

> library(boot)
> glm.fit <- glm(mpg ~ horsepower, data = Auto)
> cv.err <- cv.glm(Auto, glm.fit)
> cv.err$delta

```

```

      1      1
24.23 24.23

```

The `cv.glm()` function produces a list with several components. The two numbers in the `delta` vector contain the cross-validation results. In this case the numbers are identical (up to two decimal places) and correspond to the LOOCV statistic given in (5.1). Below, we discuss a situation in which the two numbers differ. Our cross-validation estimate for the test error is approximately 24.23.

We can repeat this procedure for increasingly complex polynomial fits. To automate the process, we use the `for()` function to initiate a *for loop* which iteratively fits polynomial regressions for polynomials of order $i = 1$ to $i = 10$, computes the associated cross-validation error, and stores it in the i th element of the vector `cv.error`. We begin by initializing the vector.

`for()`
for loop

```

> cv.error <- rep(0, 10)
> for (i in 1:10) {
+   glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
+   cv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]
+ }
> cv.error
[1] 24.23 19.25 19.33 19.42 19.03 18.98 18.83 18.96 19.07 19.49

```

As in Figure 5.4, we see a sharp drop in the estimated test MSE between the linear and quadratic fits, but then no clear improvement from using higher-order polynomials.

5.3.3 *k*-Fold Cross-Validation

The `cv.glm()` function can also be used to implement k -fold CV. Below we use $k = 10$, a common choice for k , on the `Auto` data set. We once again set a random seed and initialize a vector in which we will store the CV errors corresponding to the polynomial fits of orders one to ten.

```

> set.seed(17)
> cv.error.10 <- rep(0, 10)
> for (i in 1:10) {
+   glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
+   cv.error.10[i] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]
+ }
> cv.error.10
[1] 24.27 19.27 19.35 19.29 19.03 18.90 19.12 19.15 18.87 20.96

```

Notice that the computation time is shorter than that of LOOCV. (In principle, the computation time for LOOCV for a least squares linear model should be faster than for k -fold CV, due to the availability of the formula (5.2) for LOOCV; however, unfortunately the `cv.glm()` function does not make use of this formula.) We still see little evidence that using cubic or higher-order polynomial terms leads to lower test error than simply using a quadratic fit.

We saw in Section 5.3.2 that the two numbers associated with `delta` are essentially the same when LOOCV is performed. When we instead perform k -fold CV, then the two numbers associated with `delta` differ slightly. The first is the standard k -fold CV estimate, as in (5.3). The second is a bias-corrected version. On this data set, the two estimates are very similar to each other.

5.3.4 The Bootstrap

We illustrate the use of the bootstrap in the simple example of Section 5.2, as well as on an example involving estimating the accuracy of the linear regression model on the `Auto` data set.

Estimating the Accuracy of a Statistic of Interest

One of the great advantages of the bootstrap approach is that it can be applied in almost all situations. No complicated mathematical calculations are required. Performing a bootstrap analysis in `R` entails only two steps. First, we must create a function that computes the statistic of interest. Second, we use the `boot()` function, which is part of the `boot` library, to perform the bootstrap by repeatedly sampling observations from the data set with replacement. `boot()`

The `Portfolio` data set in the `ISLR2` package is simulated data of 100 pairs of returns, generated in the fashion described in Section 5.2. To illustrate the use of the bootstrap on this data, we must first create a function, `alpha.fn()`, which takes as input the (X, Y) data as well as a vector indicating which observations should be used to estimate α . The function then outputs the estimate for α based on the selected observations.

```
> alpha.fn <- function(data, index) {
+   X <- data$X[index]
+   Y <- data$Y[index]
+   (var(Y) - cov(X, Y)) / (var(X) + var(Y) - 2 * cov(X, Y))
+ }
```

This function *returns*, or outputs, an estimate for α based on applying (5.7) to the observations indexed by the argument `index`. For instance, the following command tells `R` to estimate α using all 100 observations.

```
> alpha.fn(Portfolio, 1:100)
[1] 0.576
```

The next command uses the `sample()` function to randomly select 100 observations from the range 1 to 100, with replacement. This is equivalent to constructing a new bootstrap data set and recomputing $\hat{\alpha}$ based on the new data set.

```
> set.seed(7)
> alpha.fn(Portfolio, sample(100, 100, replace = T))
[1] 0.539
```

We can implement a bootstrap analysis by performing this command many times, recording all of the corresponding estimates for α , and computing the resulting standard deviation. However, the `boot()` function automates this approach. Below we produce $R = 1,000$ bootstrap estimates for α .

`boot()`

```
> boot(Portfolio, alpha.fn, R = 1000)

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = Portfolio, statistic = alpha.fn, R = 1000)

Bootstrap Statistics :
      original      bias      std. error
t1*   0.5758      0.001      0.0897
```

The final output shows that using the original data, $\hat{\alpha} = 0.5758$, and that the bootstrap estimate for $SE(\hat{\alpha})$ is 0.0897.

Estimating the Accuracy of a Linear Regression Model

The bootstrap approach can be used to assess the variability of the coefficient estimates and predictions from a statistical learning method. Here we use the bootstrap approach in order to assess the variability of the estimates for β_0 and β_1 , the intercept and slope terms for the linear regression model that uses `horsepower` to predict `mpg` in the `Auto` data set. We will compare the estimates obtained using the bootstrap to those obtained using the formulas for $SE(\hat{\beta}_0)$ and $SE(\hat{\beta}_1)$ described in Section 3.1.2.

We first create a simple function, `boot.fn()`, which takes in the `Auto` data set as well as a set of indices for the observations, and returns the intercept and slope estimates for the linear regression model. We then apply this function to the full set of 392 observations in order to compute the estimates of β_0 and β_1 on the entire data set using the usual linear regression coefficient estimate formulas from Chapter 3. Note that we do not need the `{` and `}` at the beginning and end of the function because it is only one line long.

```
> boot.fn <- function(data, index)
+   coef(lm(mpg ~ horsepower, data = data, subset = index))
> boot.fn(Auto, 1:392)
(Intercept) horsepower
    39.936    -0.158
```

The `boot.fn()` function can also be used in order to create bootstrap estimates for the intercept and slope terms by randomly sampling from among the observations with replacement. Here we give two examples.

```
> set.seed(1)
> boot.fn(Auto, sample(392, 392, replace = T))
(Intercept) horsepower
    40.341    -0.164
```

```
> boot.fn(Auto, sample(392, 392, replace = T))
(Intercept) horsepower
  40.119      -0.158
```

Next, we use the `boot()` function to compute the standard errors of 1,000 bootstrap estimates for the intercept and slope terms.

```
> boot(Auto, boot.fn, 1000)

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = Auto, statistic = boot.fn, R = 1000)

Bootstrap Statistics :
      original      bias      std. error
t1*   39.936    0.0545    0.8413
t2*   -0.158   -0.0006    0.0073
```

This indicates that the bootstrap estimate for $SE(\hat{\beta}_0)$ is 0.84, and that the bootstrap estimate for $SE(\hat{\beta}_1)$ is 0.0073. As discussed in Section 3.1.2, standard formulas can be used to compute the standard errors for the regression coefficients in a linear model. These can be obtained using the `summary()` function.

```
> summary(lm(mpg ~ horsepower, data = Auto))$coef
      Estimate Std. Error t value Pr(>|t|)
(Intercept)   39.936     0.71750    55.7 1.22e-187
horsepower    -0.158     0.00645   -24.5 7.03e-81
```

The standard error estimates for $\hat{\beta}_0$ and $\hat{\beta}_1$ obtained using the formulas from Section 3.1.2 are 0.717 for the intercept and 0.0064 for the slope. Interestingly, these are somewhat different from the estimates obtained using the bootstrap. Does this indicate a problem with the bootstrap? In fact, it suggests the opposite. Recall that the standard formulas given in Equation 3.8 on page 66 rely on certain assumptions. For example, they depend on the unknown parameter σ^2 , the noise variance. We then estimate σ^2 using the RSS. Now although the formulas for the standard errors do not rely on the linear model being correct, the estimate for σ^2 does. We see in Figure 3.8 on page 92 that there is a non-linear relationship in the data, and so the residuals from a linear fit will be inflated, and so will $\hat{\sigma}^2$. Secondly, the standard formulas assume (somewhat unrealistically) that the x_i are fixed, and all the variability comes from the variation in the errors ϵ_i . The bootstrap approach does not rely on any of these assumptions, and so it is likely giving a more accurate estimate of the standard errors of $\hat{\beta}_0$ and $\hat{\beta}_1$ than is the `summary()` function.

Below we compute the bootstrap standard error estimates and the standard linear regression estimates that result from fitting the quadratic model to the data. Since this model provides a good fit to the data (Figure 3.8),

there is now a better correspondence between the bootstrap estimates and the standard estimates of $SE(\hat{\beta}_0)$, $SE(\hat{\beta}_1)$ and $SE(\hat{\beta}_2)$.

```
> boot.fn <- function(data, index)
+   coef(
+     lm(mpg ~ horsepower + I(horsepower^2),
+       data = data, subset = index)
+   )
> set.seed(1)
> boot(Auto, boot.fn, 1000)
```

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:

```
boot(data = Auto, statistic = boot.fn, R = 1000)
```

Bootstrap Statistics :

	original	bias	std. error
t1*	56.9001	3.51e-02	2.0300
t2*	-0.4661	-7.08e-04	0.0324
t3*	0.0012	2.84e-06	0.0001

```
> summary(
+   lm(mpg ~ horsepower + I(horsepower^2), data = Auto)
+ )$coef
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	56.9001	1.8004	32	1.7e-109
horsepower	-0.4662	0.0311	-15	2.3e-40
I(horsepower^2)	0.0012	0.0001	10	2.2e-21

5.4 Exercises

Conceptual

- Using basic statistical properties of the variance, as well as single-variable calculus, derive (5.6). In other words, prove that α given by (5.6) does indeed minimize $\text{Var}(\alpha X + (1 - \alpha)Y)$.
- We will now derive the probability that a given observation is part of a bootstrap sample. Suppose that we obtain a bootstrap sample from a set of n observations.
 - What is the probability that the first bootstrap observation is *not* the j th observation from the original sample? Justify your answer.
 - What is the probability that the second bootstrap observation is *not* the j th observation from the original sample?
 - Argue that the probability that the j th observation is *not* in the bootstrap sample is $(1 - 1/n)^n$.

- (d) When $n = 5$, what is the probability that the j th observation is in the bootstrap sample?
- (e) When $n = 100$, what is the probability that the j th observation is in the bootstrap sample?
- (f) When $n = 10,000$, what is the probability that the j th observation is in the bootstrap sample?
- (g) Create a plot that displays, for each integer value of n from 1 to 100,000, the probability that the j th observation is in the bootstrap sample. Comment on what you observe.
- (h) We will now investigate numerically the probability that a bootstrap sample of size $n = 100$ contains the j th observation. Here $j = 4$. We repeatedly create bootstrap samples, and each time we record whether or not the fourth observation is contained in the bootstrap sample.

```
> store <- rep(NA, 10000)
> for(i in 1:10000){
  store[i] <- sum(sample(1:100, rep=TRUE) == 4) > 0
}
> mean(store)
```

Comment on the results obtained.

3. We now review k -fold cross-validation.

- (a) Explain how k -fold cross-validation is implemented.
 - (b) What are the advantages and disadvantages of k -fold cross-validation relative to:
 - i. The validation set approach?
 - ii. LOOCV?
4. Suppose that we use some statistical learning method to make a prediction for the response Y for a particular value of the predictor X . Carefully describe how we might estimate the standard deviation of our prediction.

Applied

5. In Chapter 4, we used logistic regression to predict the probability of **default** using **income** and **balance** on the **Default** data set. We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.
- (a) Fit a logistic regression model that uses **income** and **balance** to predict **default**.

- (b) Using the validation set approach, estimate the test error of this model. In order to do this, you must perform the following steps:
 - i. Split the sample set into a training set and a validation set.
 - ii. Fit a multiple logistic regression model using only the training observations.
 - iii. Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the `default` category if the posterior probability is greater than 0.5.
 - iv. Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.
 - (c) Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.
 - (d) Now consider a logistic regression model that predicts the probability of `default` using `income`, `balance`, and a dummy variable for `student`. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for `student` leads to a reduction in the test error rate.
6. We continue to consider the use of a logistic regression model to predict the probability of `default` using `income` and `balance` on the `Default` data set. In particular, we will now compute estimates for the standard errors of the `income` and `balance` logistic regression coefficients in two different ways: (1) using the bootstrap, and (2) using the standard formula for computing the standard errors in the `glm()` function. Do not forget to set a random seed before beginning your analysis.
- (a) Using the `summary()` and `glm()` functions, determine the estimated standard errors for the coefficients associated with `income` and `balance` in a multiple logistic regression model that uses both predictors.
 - (b) Write a function, `boot.fn()`, that takes as input the `Default` data set as well as an index of the observations, and that outputs the coefficient estimates for `income` and `balance` in the multiple logistic regression model.
 - (c) Use the `boot()` function together with your `boot.fn()` function to estimate the standard errors of the logistic regression coefficients for `income` and `balance`.
 - (d) Comment on the estimated standard errors obtained using the `glm()` function and using your bootstrap function.

7. In Sections 5.3.2 and 5.3.3, we saw that the `cv.glm()` function can be used in order to compute the LOOCV test error estimate. Alternatively, one could compute those quantities using just the `glm()` and `predict.glm()` functions, and a for loop. You will now take this approach in order to compute the LOOCV error for a simple logistic regression model on the `Weekly` data set. Recall that in the context of classification problems, the LOOCV error is given in (5.4).

- (a) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2`.
- (b) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2` *using all but the first observation*.
- (c) Use the model from (b) to predict the direction of the first observation. You can do this by predicting that the first observation will go up if $P(\text{Direction} = \text{"Up"} | \text{Lag1}, \text{Lag2}) > 0.5$. Was this observation correctly classified?
- (d) Write a for loop from $i = 1$ to $i = n$, where n is the number of observations in the data set, that performs each of the following steps:
 - i. Fit a logistic regression model using all but the i th observation to predict `Direction` using `Lag1` and `Lag2`.
 - ii. Compute the posterior probability of the market moving up for the i th observation.
 - iii. Use the posterior probability for the i th observation in order to predict whether or not the market moves up.
 - iv. Determine whether or not an error was made in predicting the direction for the i th observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0.
- (e) Take the average of the n numbers obtained in (d)iv in order to obtain the LOOCV estimate for the test error. Comment on the results.

8. We will now perform cross-validation on a simulated data set.

- (a) Generate a simulated data set as follows:

```
> set.seed(1)
> x <- rnorm(100)
> y <- x - 2 * x^2 + rnorm(100)
```

In this data set, what is n and what is p ? Write out the model used to generate the data in equation form.

- (b) Create a scatterplot of X against Y . Comment on what you find.
- (c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

- i. $Y = \beta_0 + \beta_1 X + \epsilon$
- ii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$
- iii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$
- iv. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$.

Note you may find it helpful to use the `data.frame()` function to create a single data set containing both X and Y .

- (d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?
 - (e) Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.
 - (f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?
9. We will now consider the **Boston** housing data set, from the **ISLR2** library.
- (a) Based on this data set, provide an estimate for the population mean of `medv`. Call this estimate $\hat{\mu}$.
 - (b) Provide an estimate of the standard error of $\hat{\mu}$. Interpret this result.
Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the number of observations.
 - (c) Now estimate the standard error of $\hat{\mu}$ using the bootstrap. How does this compare to your answer from (b)?
 - (d) Based on your bootstrap estimate from (c), provide a 95% confidence interval for the mean of `medv`. Compare it to the results obtained using `t.test(Boston$medv)`.
Hint: You can approximate a 95% confidence interval using the formula $[\hat{\mu} - 2SE(\hat{\mu}), \hat{\mu} + 2SE(\hat{\mu})]$.
 - (e) Based on this data set, provide an estimate, $\hat{\mu}_{med}$, for the median value of `medv` in the population.
 - (f) We now would like to estimate the standard error of $\hat{\mu}_{med}$. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.
 - (g) Based on this data set, provide an estimate for the tenth percentile of `medv` in Boston census tracts. Call this quantity $\hat{\mu}_{0.1}$. (You can use the `quantile()` function.)
 - (h) Use the bootstrap to estimate the standard error of $\hat{\mu}_{0.1}$. Comment on your findings.

