REGRESSION ANALYSIS
&
PREDICTIVE MODELING - III

Resampling Methods

DSM 1003

MOHAMMAD WASIQ
Data Science

Resampling Methods

Mohammad Wasiq

25/03/2022

Table of Contents

Book :-: **An Introduction to Statistical Learning with Applications in R** by **Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani**
Teacher :-: **Prof. Ahmedur Rehman Sir**
Composer :-: **Mohammad Wasiq** *(Data Science)*

1 Resampling Methods

Resampling Methods are an *indispensable tool* in modern statistics . They involve repeatedly drawing samples from a *training set* and refitting a model on interest on each sample in order to obtain additional information about the *fitted model* . Resampling approaches can be computationally expensive, because they involve fitting the same statistical method multiple times using different subsets of the training data.

Here we discuss two of the most commonly used resampling methods :
Cross-Validation and **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 model .

1.1 Cross - Validation (CV)

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 .
In contrast, the **training error** can be easily calculated by applying the statistical learning method to the observations used in its training .

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 .

(From GeeksforGeeks)

In machine learning, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the cross-validation technique.

Cross-Validation : Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

The three steps involved in cross-validation are as follows : -
(i) Reserve some portion of sample data-set.
(ii) Using the rest data-set train the model.
(iii) Test the model using the reserve portion of the data-set.

Methods of Cross Validation
1. Validation
2. LOOCV (Leave One Out Cross Validation)
3. K-Fold Cross Validation

1.1.1 The Validation Set Approach

It involves randomly dividing the available set of observations into two parts, a *training set* and a *validation set*. The resulting validation set error rate- *Typically assessing using MSE* in the case of a *quantitative response* - provides an estimate of the test error rate.

The statistical learning method is fit on the training set, and its performance is evaluated on the validation set. The validation set approach is conceptually simple and is easy to implement. But it has two potential drawbacks :

i. 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 .

ii. 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.

Steps Involved in the Validation Set Approach :
1. A random splitting of the dataset into a certain ratio(generally 70-30 or 80-20 ratio is preferred)
2. Training of the model on the training data set
3. The resultant model is applied to the validation set
4. Model's accuracy is calculated through prediction error by using model performance metrics

1.1.2 LOOCV (Leave-One-Out Cross-Valiadation)

Leave-one-out cross-validation (LOOCV) is closely related to the *validation set* , but it attempts to address that method's 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 $(x_2, y_2), \ldots, (x_n, y_n)$ make up the *training set* . The statistical learning method is fit on the $n-1$ training observations and a prediction $\widehat{y_1}$ is made for the excluded observation using its value x_1 . Since (x_1, y_1) was not used in the fitting process , $MSE = (y_1 - \widehat{y_1})$ provides an approx. 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) . Repeating this approach n times *produces* n squared errors, $MSE_1, \ldots, MSE_n$. The *LOOCV* estimate for the test MSE is the average of these n test error estimates :

$$\boldsymbol{CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i}$$

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

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 :

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

where, $\hat{y}_i$ is the i^{th} fitted value from the original least squares fit and h_i is the leverage (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. $\boldsymbol{h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^{n}(x_{i'} - \bar{x})^2}}$)

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* or linear discriminant analysis *LDA*

GeeksforGeeks In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages also. An advantage of using this method is that we make use of all data points and hence it is low bias. The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

1.1.3 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. This process results in k estimates of the test error, MSE_1, MSE_2 ,…, MSE_k. The *k-fold CV* estimate is computed by averaging these values

$$\boldsymbol{CV_k = \frac{1}{k}\sum_{i=1}^{k} MSE_i}$$

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. But *cross-validation* is a very general approach that can be applied to almost any statistical learning method. There also can be other non-computational advantages to performing *5-fold* or *10-fold* CV, which involve the bias-variance trade-off .

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.

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*.

Geeks for Geeks In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Advantages of train/test split : This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times. Simpler to examine the detailed results of the testing process.

Advantages of cross-validation : More accurate estimate of out-of-sample accuracy. More "efficient" use of data as every observation is used for both training and testing.

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

As we know 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 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.

1.1.5 Cross-Validation on Classification Problems

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

$$\boldsymbol{CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i}$$

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

1.2 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. he 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.

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 $Var(\alpha X + (1 - \alpha)Y)$. One can show that the value that minimizes the risk is given by

$$\boldsymbol{\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}}$$

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

In reality, the quantities σ_X^2, σ_Y^2, σ_{XY} are unknown. We can compute estimates for these quantities $\hat{\sigma}_X^2$, $\hat{\sigma}_Y^2, \hat{\sigma}_{XY}$ 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

$$\boldsymbol{\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}}$$

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 on a simple data set, which we call Z, that contains only $n =$ 3observations. 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 observations 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 X and Y values are included. We can use Z^{*1} to produce a new bootstrap estimate for α , which can

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 $Z^{*1}, Z^{*2}, \ldots, Z^{*B}$, and B corresponding α estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$. we can compute the standard error of these bootstrap estimates using the formula

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

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

1.3 Labs : Cross-Validation and the Bootstrap

1.3.1 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.

We begin by using the `sample()` function to split the set of observations `sample()` 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)
```

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)

## The following objects are masked from Auto (pos = 3):
##
##     acceleration, cylinders, displacement, horsepower, mpg, name, origin,
weight, year

## The following objects are masked from Auto (pos = 6):
##
##     acceleration, cylinders, displacement, horsepower, mpg, name, origin,
weight, year
```

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

## [1] 23.26601
```

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] 18.71646

lm.fit3 <- lm(mpg ~ poly(horsepower , 3), data = Auto ,
              subset = train)

mean((mpg - predict(lm.fit3 , Auto))[-train]^2)

## [1] 18.79401
```

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] 25.72651

lm.fit2 <- lm(mpg ~ poly (horsepower , 2), data = Auto ,
              subset = train)

mean((mpg - predict(lm.fit2 , Auto))[-train]^2)

## [1] 20.43036

lm.fit3 <- lm(mpg ~ poly(horsepower , 3), data = Auto ,
              subset = train)

mean ((mpg - predict (lm.fit3 , Auto))[-train]^2)

## [1] 20.38533
```

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

1.3.2 LOOCV (Leave-One-Out Cross-Validation)

The *LOOCV* estimate can be automatically computed for any generalized *linear model* using the $glm(\quad)$ and $cv.glm(\quad)$ 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,

```
glm.fit <- glm (mpg ~ horsepower , data = Auto)

coef (glm.fit)

## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

OR

```
lm.fit <- lm(mpg ~ horsepower , data = Auto)
coef (lm.fit)
```

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

```
library(boot)

glm.fit <- glm(mpg ~ horsepower, data = Auto)

cv.err <- cv.glm(data = Auto , glmfit = glm.fit)

cv.err$delta

## [1] 24.23151 24.23114
```

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 to the *LOOCV statistic* given in $CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$.
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.

```
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] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305
18.96115 19.06863 19.49093
```

The estimated test *MSE* between the *linear* and *quadratic* fits, but then no clear improvement from using *higher-order polynomials*

1.3.3 k-Fold Cross-Validation

The $cv.glm(\quad)$ 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] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061
19.14666 18.87013 20.95520
```

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

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 $CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$. The second

is a bias-corrected version. On this data set, the two estimates are very similar to each other.

1.3.4 Bootstrap

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 :
(i) We must create a function that computes the statistic of interest.
(ii) We use the **boot()** function, which is part of the *boot* library, to `boot()` perform the bootstrap by repeatedly sampling observations from the data set with replacement.

The **Portfolio** data set in the *ISLR2* package is simulated data of 100 pairs of returns. 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 $\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$ 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] 0.5758321
```

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] 0.5385326
```

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

```
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* 0.5758321 0.0007959475  0.08969074
```

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 $SE(\beta_0)$ and $SE(\beta_1)$

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. 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
##  39.9358610  -0.1578447
```

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
##  40.3404517  -0.1634868
```

```
boot.fn(Auto , sample(392, 392, replace = T))

## (Intercept)  horsepower
##  40.1186906  -0.1577063
```

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* 39.9358610  0.0544513229 0.841289790
## t2* -0.1578447 -0.0006170901 0.007343073
```

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. These can be obtained using the **summary()** function

```
summary(lm(mpg ~ horsepower , data = Auto))$coef

##               Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
## horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

The standard error estimates for $\hat{\beta}_0$ and $\hat{\beta}_1$ obtained using the formulas are 0.717 for the intercept and 0.0064 for the slope. Interestingly, these are somewhat different from the estimates obtained using the bootstrap. 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 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* 56.900099702  3.511640e-02 2.0300222526
## t2* -0.466189630 -7.080834e-04 0.0324241984
## t3*  0.001230536  2.840324e-06 0.0001172164

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

##                     Estimate   Std. Error   t value      Pr(>|t|)
## (Intercept)     56.900099702 1.8004268063  31.60367 1.740911e-109
## horsepower      -0.466189630 0.0311246171 -14.97816  2.289429e-40
## I(horsepower^2)  0.001230536 0.0001220759  10.08009  2.196340e-21
```