JW Ch 5 Lab

Ruchik Jani

12th April, 2024

Instructions and time-saving hints. To begin this lab, download the archive and unpack it. Inside the folder, you will find a Rmarkdown file.

The purpose of this lab is to implement linear regression in R. You will reproduce the commands blocks in JW Section 5.3 in this Rmarkdown file. **Please do not retype all the R commands as shown in Section 5.3.** Instead, go to link to text for Lab JW Ch 5, cut and paste the text into your Rmarkdown file. Then break up the R commands into the R chunks shown in the text. (R chunks are braced by triple backticks and a leading {r}, as before.) Do not put all commands into the same block! The idea is to imitate the code chunks in Section 5.3 with the additional plots and output omitted from the text. Below, I have given you the first two blocks for reference.

After creating a new chunk, you may knit the document. Read Section 5.3 as you go. You may comment out the `fix()` commands if you do not wish to close editor windows during knitting. You may also wish to comment out the commands which deliberately produce an error during the exercise. Since R chunks are independent pieces of code, you will produce some errors when imitating the format in the book. For example, the `abline()` commands must be grouped in the same chunk as the initial `plot()` command.

Congratulations! When the document is successfully knitted, you should reread Section 5.3 along with the Rmarkdown output. You should see the commands along with the output plots. Submit your Rmd file along with an unzipped PDF of the result before the deadline.

Chaper 5 Lab: Cross-Validation and the Bootstrap

The Validation Set Approach

The below code snippet loads the *ISLR* package, providing access to datasets and functions for statistical learning. It then sets the random number generator seed to 1, ensuring reproducibility of random outcomes. After that it samples 196 numbers from 1 to 392 without replacement, storing them in the *train* variable.

```
#install.packages("ISLR") #To install the ISLR package on your system, remove the comment
# symbol "#" from the beginning of the line.
library(ISLR) #Load ISLR package
set.seed(1) #Set random number generator seed to 1
train=sample(392,196) #Sample 196 numbers from 1 to 392 without replacement
```

The below code snippet is used to fit a linear regression model (*lm*) where *mpg* is predicted by *horsepower* using data from the *Auto* dataset, considering only the subset specified by the *train* variable.

```
# Linear regression: mpg ~ horsepower using 'train' subset.
lm.fit=lm(mpg~horsepower,data=Auto,subset=train)
```

The below code snippet first attaches the *Auto* dataset, allowing direct access to its variables. Then, it calculates the mean squared error of a linear regression model (*lm.fit*) predicting *mpg* values based on the *Auto* dataset, excluding the subset specified by *train*. The estimated test MSE for the linear regression fit is 23.26601.

```
attach(Auto) # Attach 'Auto' dataset
mean((mpg-predict(lm.fit,Auto))[-train]^2) # Mean squared error excluding 'train'
```

```
## [1] 23.26601
```

The below code snippet fits two polynomial regression models of degrees 2 and 3, respectively, to predict *mpg* based on the *horsepower* variable from the *Auto* dataset, using only the subset specified by *train*. It then calculates the mean squared error (MSE) for each model, evaluating their predictive performance on the entire dataset excluding the *train* subset. These error rates are 18.71646 and 18.79401, respectively.

```
lm.fit2=lm(mpg~poly(horsepower,2),data=Auto,subset=train) # Fit quadratic regression model
mean((mpg-predict(lm.fit2,Auto))[-train]^2) # Calculate MSE for quadratic model
```

```
## [1] 18.71646
```

```
lm.fit3=lm(mpg~poly(horsepower,3),data=Auto,subset=train) # Fit cubic regression model
mean((mpg-predict(lm.fit3,Auto))[-train]^2) # Calculate MSE for cubic model
```

```
## [1] 18.79401
```

The below code snippet uses a different training dataset and then calculates the validation set MSE for all the cases we have defined previously. The validation set MSE for the models with linear, quadratic and cubic terms are 25.72651, 20.43036 and 20.38533 respectively.

```
set.seed(2)  # Set seed for reproducibility
train = sample(392, 196)  # Randomly sample 196 indices for training data
lm.fit = lm(mpg ~ horsepower, subset = train)  # Fit linear regression model
mean((mpg - predict(lm.fit, Auto))[-train]^2)  # Calculate MSE for linear model
```

```
## [1] 25.72651
```

```
# Fit quadratic regression model
lm.fit2 = lm(mpg ~ poly(horsepower, 2), data = Auto, subset = train)
mean((mpg - predict(lm.fit2, Auto))[-train]^2)  # Calculate MSE for quadratic model
```

```
## [1] 20.43036
```

```
# Fit cubic regression model
lm.fit3 = lm(mpg ~ poly(horsepower, 3), data = Auto, subset = train)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)  # Calculate MSE for cubic model
```

```
## [1] 20.38533
```

The outcomes are consistent with previous test error findings: a quadratic model outperforms linear, and minimal support exists for a cubic model.

Leave-One-Out Cross-Validation

The below code snippet fits a generalized linear model (GLM) to predict *mpg* based on *horsepower* from the *Auto* dataset. Then, it retrieves and displays the coefficients of the fitted GLM, which represent the relationship between *mpg* and *horsepower*.

```
glm.fit = glm(mpg ~ horsepower, data = Auto)  # Fit a generalized linear model
coef(glm.fit)  # Display coefficients of the model
```

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

The below code snippet does the same thing by using a different function: *lm*, instead of *glm* which was used before. Both the functions have the same linear regression model as seen from the output coefficient values which are the same.

```
lm.fit = lm(mpg ~ horsepower, data = Auto)  # Fit linear regression model
coef(lm.fit)  # Extract coefficients from the fitted model
```

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

The below code snippet utilizes the boot package for bootstrapping methods. It fits a linear regression model to predict *mpg* based on *horsepower* using the *glm()* function. Then, it performs cross-validation on the model using *cv.glm()*, estimating prediction error. Finally, it displays the cross-validation error estimates stored in the *cv.err* object by accessing the *delta* vector. Our cross-validation estimate for the test error is approximately 24.23.

```
library(boot)  # Load the boot package
glm.fit = glm(mpg ~ horsepower, data = Auto)  # Fit a linear regression model
cv.err = cv.glm(Auto, glm.fit)  # Perform cross-validation
cv.err$delta  # Display the cross-validation error estimates
```

```
## [1] 24.23151 24.23114
```

The below code snippet iterates through polynomial degrees from 1 to 5, fitting polynomial regression models of increasing complexity to predict *mpg* based on *horsepower*. Within each iteration, it computes the cross-validation error using *cv.glm()* and stores the result in the *cv.error* vector. Finally, it displays the cross-validation errors for each polynomial degree.

```
cv.error = rep(0, 5)  # Initialize vector to store cross-validation errors
for (i in 1:5) {  # Loop over polynomial degrees from 1 to 5
  glm.fit = glm(mpg ~ poly(horsepower, i), data = Auto)  # Fit polynomial regression model
  cv.error[i] = cv.glm(Auto, glm.fit)$delta[1]  # Store cross-validation error for
  #current degree of polynomial
}
cv.error  # Display cross-validation errors
```

```
## [1] 24.23151 19.24821 19.33498 19.42443 19.03321
```

We observe a significant decrease in the estimated test MSE from the linear to the quadratic fits, but subsequent polynomial degrees show no visible improvement.

k-Fold Cross-Validation

The below code snippet sets a seed for reproducibility, then iterates through polynomial degrees from 1 to 10, fitting polynomial regression models of increasing complexity to predict *mpg* based on *horsepower*. Within each iteration, it computes the 10-fold cross-validation error using *cv.glm()* and stores the result in the *cv.error.10* vector. Finally, it displays the 10-fold cross-validation errors for each polynomial degree.

```
set.seed(17)  # Set seed for reproducibility
cv.error.10 = rep(0, 10)  # Initialize vector for 10-fold cross-validation errors
for (i in 1:10) {  # Loop over polynomial degrees from 1 to 10
  glm.fit = glm(mpg ~ poly(horsepower, i), data = Auto)  # Fit polynomial regression model
  cv.error.10[i] = cv.glm(Auto, glm.fit, K = 10)$delta[1]  # Store 10-fold
  #cross-validation error for current model degree
}
cv.error.10  # Display 10-fold cross-validation errors
```

```
##  [1] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
##  [9] 18.87013 20.95520
```

We continue to find little support for the idea that using cubic or higher-order polynomial terms reduces test error compared to a quadratic fit. In the previous section, we observed that the two delta values are nearly identical in LOOCV, but differ slightly in k-fold CV, with one being the standard estimate and the other a bias-corrected version.

The Bootstrap

Estimating the Accuracy of a Statistic of Interest

The below code snippet defines a function named *alpha.fn* that takes two arguments: *data*, a dataset with variables X and Y, and index, a vector of indices indicating which observations to use. Within the function, it subsets the X and Y variables based on the provided indices and calculates *alpha* using the given formula, which represents a measure of association between two variables adjusted for their individual variances.

```
alpha.fn = function(data, index) {  # Define a function to calculate alpha
  X = data$X[index]  # Subset X values using index
  Y = data$Y[index]  # Subset Y values using index
  return((var(Y) - cov(X, Y)) / (var(X) + var(Y) - 2 * cov(X, Y)))  # Calculate alpha using formula
}
```

The below code snippet calls the *alpha.fn* function with the *Portfolio* dataset and the indices 1 through 100 as arguments to estimate *alpha*.

```
alpha.fn(Portfolio, 1:100)  # Calculate alpha for the first 100 observations in the Portfolio dataset
```

```
## [1] 0.5758321
```

The below code snippet sets a seed for reproducibility and then calls the *alpha.fn* function with the *Portfolio* dataset and a random sample of 100 observations drawn with replacement. The function calculates *alpha* using the specified sample of data. This means that *alpha* is calculated on a new dataset.

```r
set.seed(1)  # Set seed for reproducibility
alpha.fn(Portfolio, sample(100, 100, replace = T))  # Calculate for random sample of 100 observations
```

```
## [1] 0.7368375
```

This below code snippet performs bootstrapping on the *Portfolio* dataset with 1000 replications. During each replication, it calculates *alpha* using the *alpha.fn* function. The output is a collection of 1000 bootstrap estimates of *alpha*.

```r
# Perform bootstrapping with 1000 replications using the alpha.fn function on the Portfolio dataset
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 -0.001695873  0.09366347
```

The final result indicates $alpha = 0.5758321$ for the original data, with a bootstrap standard deviation estimate of SE($alpha$) as 0.09050198. The values for bias and std. error will change every time the code chunk is executed, since bootstrapping involves resampling from the original dataset with replacement. As a result, the resampled datasets used for calculating the bootstrap estimates of bias and standard error vary from one execution to another, leading to different values for these statistics each time the code is run.

Estimating the Accuracy of a Linear Regression Model

The below code snippet defines a function named *boot.fn* that fits a linear regression model to predict *mpg* based on *horsepower* using a subset of data specified by the indices. The function returns the coefficients of the fitted model. Then, it calls the *boot.fn* function with the *Auto* dataset and all indices (1 through 392), which fits a linear regression model and returns the coefficients.

```r
boot.fn = function(data, index)  # Define a function to fit linear regression and return coefficients
  return(coef(lm(mpg ~ horsepower, data = data, subset = index)))  # return regression coefficients
boot.fn(Auto, 1:392)  # Call the boot.fn function with the Auto dataset and all indices
```

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

The below code snippet sets a seed, then fits linear regression models using two different random samples of 392 observations with replacement from the *Auto* dataset.

```r
set.seed(1)  # Set seed
boot.fn(Auto, sample(392, 392, replace = T))  # Fit linear regression with random sample
```

```
## (Intercept)  horsepower
##  40.3404517  -0.1634868
```

```
boot.fn(Auto, sample(392, 392, replace = T))  # Fit linear regression with another random sample
```

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

The below code performs bootstrapping on the *Auto* dataset with 1000 replications. During each replication, it fits a linear regression model using the *boot.fn* function. The output is a collection of bootstrap estimates of the coefficients. The bootstrap estimates of SE(B0) which is intercept and SE(B1) which is slope are 0.86846537 and 0.00745485 respectively. Tbese values change with different program runs as explained before.

```
boot(Auto, boot.fn, 1000)  # Bootstrap with 1000 replications using boot.fn on Auto
```

```
##
## 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
```

The below code snippet computes a summary of the linear regression model. It then extracts the coefficients from the summary using *$coef*. The standard error estimates for intercept and slope are 0.717498656 and 0.006445501 respectively. These estimates are different from the bootstrap estimates. The bootstrap estimates are more accurate since they are independent of any assumptions. The summary method is based on standard formulas to calculate linear regression coefficients which rely on certain assumptions on variance and error.

```
summary(lm(mpg ~ horsepower, data = Auto))$coef  # Extract coefficients from linear regression summary
```

```
##               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 below code snippet defines a function named *boot.fn* that fits a quadratic regression model using a subset of data specified by the indices. Then, it sets a seed for reproducibility, performs bootstrapping with 1000 replications on the *Auto* dataset using the *boot.fn* function, and computes a summary of the quadratic regression model, extracting the coefficients from the summary.

```
boot.fn = function(data, index)  # Define function to fit quadratic regression and return coefficients
 coefficients(lm(mpg ~ horsepower + I(horsepower^2), data = data, subset = index))  # Fit quadratic
  #regression and return coefficients
set.seed(1)  # Set seed for reproducibility
boot(Auto, boot.fn, 1000)  # Perform bootstrapping with 1000 replications using boot.fn on Auto
```

```
##
## 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  # Extract coefficients from summary
```

```
##                     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
```

From the above results, we can conclude that the quadratic model exhibits a strong fit to the data. This can be inferred by the close agreement between the bootstrap estimates and the standard estimates of SE(B0), SE(B1), and SE(B2).