5.3 Lab: Cross-Validation and the Bootstrap

We want to explore resampling techniques.

5.3.1 The Validation Set Approach

We want to use the validation set approach to estimate the test error rates that result from fitting different linear models on the **Auto** data.

For setup we need to use the **set.seed()** func to make things reproducible. This is usually a good idea working with RNG's. We are using the **sample()** function for to split up our data into halves. Wich meens a subset size of 196 out of the 392 observations. We will refere to these as the trainingset.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.6.3
```

```
set.seed(1)
train = sample(392, 196)
```

Then we use the **subset** option in **lm()** to fit a linear regression only using the training set.

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

We then use the **predict()** function to estimate the response for all 392 observatons, and use the **mean()** function to calculate the MSE of the 196 observations in the validation set. Note: **-train** index selects only the observations that are not in the trainingset.

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

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

As we can see the estimated MSE for the linear regression fit is 23.27 (26.14 from the book). To estimate the test error for the quadratic and cubic regressions we can use **poly()**.

```
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
```

The error rates are 18.72 (19.78) and 18.79 (19.78). If we choose a different training set we will get slighty different errrors 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 training and validation set we get the resoult's 25.72, 20.43, 20.39 (23.30, 18.90, and 19.2). Wich is different but similar. This resoult is consisten with our previus findings: a model that predicts **mpg** using a quadratic function of **horsepower**, and there is little evidenc in favor of a model that uses cubic function of **horsepower**.

5.3.2 Leave-one-out Cross-Validation (LOOCV)

The LOOCV estimate can be automatically computed for any generalized linear model using the **glm()** and **cv.glm()** functions. If we use **glm()** without a specified family it will do the same as the **lm()** function. But we need the **glm()**, eaven tho we are going to do a normal linear model, since **glm()** works with **cv.glm()**, wich is a part of the boot libarary.

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

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

The **cv.glm()** function makes a list with several components. The two numbers in the ___delta_ vector contain the cross-validation results. In this case the numbers are equvivalent. Our Cross-validation estimate for the test erro is aprroximatley 24.23.

We can do 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=5, computed associated cross-validation error, and stores it in the *ith* element of the vecotr **cv.error**. We start by initializing the vector. This commad will likely take a couple of minutes to run.

```
cv.error = rep(0,5)
for (i in 1:5){
  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
```

As mentioned earlier (in the book fig 5.4) we observe a sharp drop in MSE from linear to quadratic, but no improvment going further.

5.3.3 k-fold Cross-Validation

The **cv.glm()** function can also be used to implement k-fold CV. We will use an example with k=10, a common choice for k, on the Auto data set. We still use the seed method and initialize a vector 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
##  [9] 18.87013 20.95520
```

We still see that there is little evidence that using cubic or higher-order polynomial terms leads to lower test error than simply using a quadratic fit.

5.3.4 The Bootstrap

We illustrate the use of the bootstrap in the simple example of Section 5.2 (from the book)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 boostrap approach is that it can be applied in almost all situations. No complicated mathematical calulations are required. Performing bootstrap analysis in **R** only need two steps. **First**, we must create a function, that computes the statistic of interest. **Second**, we use the **boot()** function, wich is part of the boot libarary, to perform the bootstrap by repeatedly sampling observations from the data set with replacement.

The **Portfolio** data set in the **ISLR** package is described in Section 5.2. To ilustrate the use of bootstrap on this data, we must first create a function, **alpha.fn()**, wich takes (X, Y) as input data, as well as a vector indicating which observations should be used to estimate α. The function then returns the estimate for α based on the selected observations.

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

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

```
set.seed(2)
alpha.fn(Portfolio, sample(100, 100, replace = T))
```

```
## [1] 0.5539577
```

We can implement a boostrap analysis by performing this command many times, recording all of the corresponding estimates for α, and computing the standard deviation. But the **boot()** function automates 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 0.0009480333  0.09303824
```

The final output shows that using the original data, $\hat{\alpha} = 0.5758$, and that the bootstrap estimate for SE(α) is 0.0930 (0.866 book).

Estimating the Accuracy of a Linear Regression Model