Lab 8: Cross-Validation and Bootstrap

Note: Some of the commands in this lab may take a while to run on your com-puter.

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.

seed

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.

sample()

```
> library(ISLR)
> 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] 26.14
```

Therefore, the estimated test MSE for the linear regression fit is 26.14. 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] 19.82
> lm.fit3=lm(mpg~poly(horsepower,3),data=Auto,subset=train)
> mean((mpg-predict(lm.fit3,Auto))[-train]^2)
[1] 19.78
```

These error rates are 19.82 and 19.78, 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] 23.30
> lm.fit2=lm(mpg~poly(horsepower,2),data=Auto,subset=train)
> mean((mpg-predict(lm.fit2,Auto))[-train]^2)
[1] 18.90
> lm.fit3=lm(mpg~poly(horsepower,3),data=Auto,subset=train)
> mean((mpg-predict(lm.fit3,Auto))[-train]^2)
[1] 19.26
```

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 23.30, 18.90, and 19.26, 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`.

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 previous labs, 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
     39.936      -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

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

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 $i = 5$, computes the associated cross-validation error, and stores it in the ith element of the vector `cv.error`. We begin by initializing the vector. This command will likely take a couple of minutes to run.

for loop

```
> 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] 24.23 19.25 19.33 19.42 19.03
```

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.

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] 24.21 19.19 19.31 19.34 18.88 19.02 18.90 19.71 18.95 19.50
```

Notice that the computation time is much 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 following formula for LOOCV;

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2,$$

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

4 The 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. 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()`

Now, we use `Portfolio` data set in the `ISLR` package. 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]
+ return((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] 0.576
```

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(1)
> alpha.fn(Portfolio,sample(100,100,replace=T))
[1] 0.596
```

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      -7.315e-05   0.0886
```

The final output shows that using the original data, $\hat{\alpha} = 0.5758$, and that the bootstrap estimate for $\text{SE}(\hat{\alpha})$ is 0.0886.

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 $\text{SE}(\hat{\beta}_0)$ and $\text{SE}(\hat{\beta}_1)$.

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)
+ return(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
  38.739     -0.148
> boot.fn(Auto,sample(392,392,replace=T))
(Intercept) horsepower
  40.038     -0.160
```

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       0.0297   0.8600
t2* -0.158      -0.0003   0.0074
```

This indicates that the bootstrap estimate for $\mathrm{SE}(\hat{\beta}_0)$ is 0.86, and that the bootstrap estimate for $\mathrm{SE}(\hat{\beta}_1)$ is 0.0074. As discussed before, 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 $\hat{\beta}_1$ 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

$$\mathrm{SE}(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right], \quad \mathrm{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

depend on the unknown parameter σ^2, the noise variance. We then estimate σ^2 using the RSS. Now although the formula for the standard errors do not rely on the linear model being correct, the estimate for σ^2 does. 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, there is now a better correspondence between the bootstrap estimates and the standard estimates of $\mathrm{SE}(\hat{\beta}_0)$, $\mathrm{SE}(\hat{\beta}_1)$ and $\mathrm{SE}(\hat{\beta}_2)$.

```
> boot.fn=function(data,index)
+ coefficients(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*  56.900    6.098e-03 2.0945
t2* -0.466    -1.777e-04 0.0334
t3*  0.001     1.324e-06 0.0001

> summary(lm(mpg~horsepower+I(horsepower^2),data=Auto))$coef
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      56.9001    1.80043      32 1.7e-109
horsepower       -0.4662    0.03112     -15  2.3e-40
I(horsepower^2)   0.0012    0.00012      10  2.2e-21
```