Lab-05

Resampling Methods

Validation Set Approach

Part 1: Split the data

First we will need to load the Auto dataset.

```
library(ISLR)
```

Now we can set a seed to make our results repeatable, and define the size of the train and test sets. In the validation set approach these will each be 50% of the original data. We then define a function to apply to the dataset index.

```
set.seed(1)

sample_size <- floor(0.5 * nrow(Auto))

train_index <- sample(seq_len(nrow(Auto)), size = sample_size)
```

Now we can define each set by applying the function to the original data.

```
train <- Auto[train_index, ]
test <- Auto[-train_index, ]

nrow(train)
```

```
## [1] 196
```

Part 2: Build the linear model

We now build a linear model using the training data.

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

summary(lm.fit.train)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.698  -3.085  -0.216   2.680  16.770
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.340377   1.002269   40.25   <2e-16 ***
## horsepower  -0.161701   0.008809  -18.36   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 4.692 on 194 degrees of freedom
## Multiple R-squared:  0.6346, Adjusted R-squared:  0.6327
## F-statistic: 336.9 on 1 and 194 DF,  p-value: < 2.2e-16
```

Part 3: Estimate MSE in test set

To get the mean square error (mse) of the model we can use the following command.

```
mean((test$mpg-predict(lm.fit.train, test))^2)
```

```
## [1] 26.14142
```

Part 4: Polynomial modes

We can also calculate the quadratic and cubic model error and compare them to the linear one.

```
# Quadratic Model
lm.fit2.train = lm(mpg~poly(horsepower, 2), data = train)

mean((test$mpg-predict(lm.fit2.train, test))^2)# has a smaller test error
```

```
## [1] 19.82259
```

```
# Cubic Model
lm.fit3.train = lm(mpg~poly(horsepower, 3), data = train)

mean((test$mpg-predict(lm.fit3.train, test))^2)# has a smaller test error
```

```
## [1] 19.78252
```

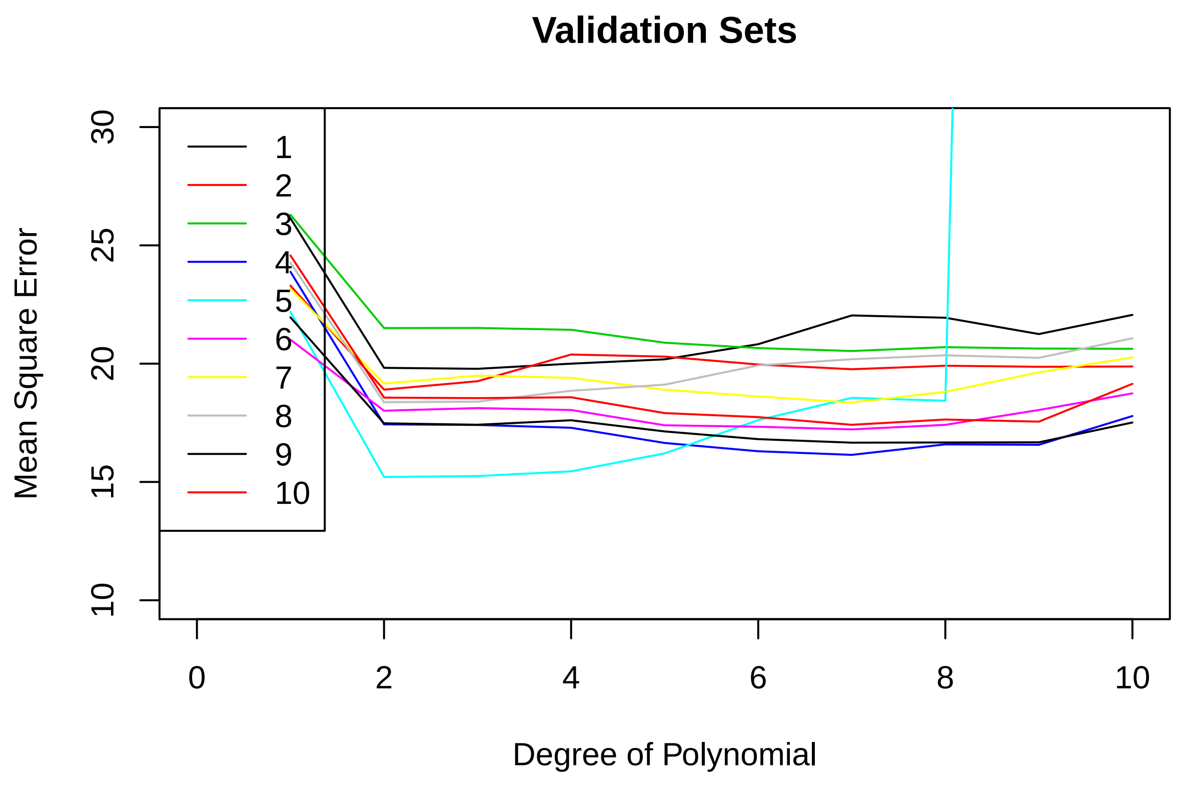
Part 5: Analysis

By comparing the mse for each model, we can see that the improvement from the linear model (1st degree) to the quadratic model (2nd) is dramatic >20%, whereas the improvement from the quadratic to cubic (3r degree) is far less pronounced <1%.

Part 6: Plot polynomial models

In order to plot the polynomial mse of models using data with 10 different seed values, we can create a matrix to store the degree values for each seed. We can then assigned the values using a nested loop, where the other loop itterating through seeds and the inner loop itterating through degree values. Once all the degree values have been captured for each seed, we can use them to plot each mse line in the outer loop. The code below illustrates this:

```
plot(0,ylab="Mean Square Error", ylim = c(10,30), xlim= c(0,10),
     xlab="Degree of Polynomial", main="Validation Sets")

# Matrix to store mse for each itteration
errors.ploy <- matrix(nrow=10, ncol=10 )

for (i in 1:10){

  set.seed(i)
  train_index <- sample(seq_len(nrow(Auto)), size = sample_size)

  # Apply split to datset
  train <- Auto[train_index, ]
  test <- Auto[-train_index, ]

  for (j in 1:10) {

    # fit model
    lm.fit.train <- lm(mpg~poly(horsepower, j), data = train)

    # assign to matrix
    errors.ploy[i, j] <-mean((test$mpg-predict(lm.fit.train, test))^2)

  }

  # plot each seed line
  lines(errors.ploy[i,],col=i)
}

legend("topleft",c("1","2","3","4","5","6","7","8","9","10"),lty=rep(1,10),col=1:10)
```



Leave-One-Out Cross Validation

Part 7: Polynomial models LOOCV

We can use a similar iterative approach for LOOCV, to compare errors for polynomial models, As we do require seed values, we can store the results in a 1 dimensional vector. Here will will fit a generalised linear model (GLM), which store the error data itself, meaning this can be extracted by name.

```
# Fit a generalised linear model
glm.fit.train <- glm(mpg~horsepower, data = train)

# Get the intercep
coef(glm.fit.train)
```

```
## (Intercept)  horsepower
##   38.473784   -0.147263
```

```
# load the boot library
library(boot)

cv.error <- cv.glm(train, glm.fit.train)

cv.error$delta
```

```
## [1] 24.43402 24.43264
```

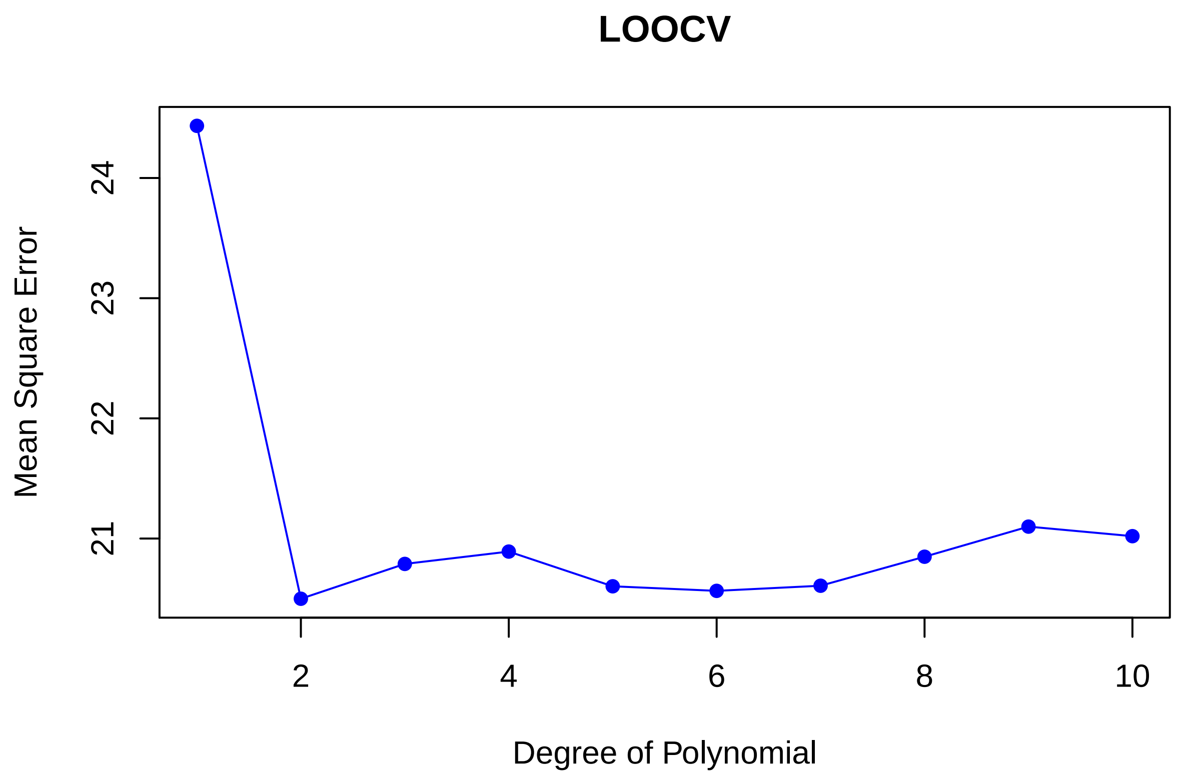
```
# Put it in a loop

deltas = vector("integer", 10)

for (i in 1:10){

  glm.fit.train <- glm(mpg~poly(horsepower, i), data = train)

  cv.error <- cv.glm(train, glm.fit.train)

  deltas[i] <- cv.error$delta[1]

}

deltas
```

```
##  [1] 24.43402 20.49895 20.78869 20.89138 20.60261 20.56460 20.60720
##  [8] 20.84852 21.09914 21.01966
```

Part 8: Plot LOOCV

To plot the result, we can plot each element of the vector against a simple sequence of 1-10.

```
plot(seq(1:10), deltas, pch = 16, col="blue",
     ylab="Mean Square Error", xlab="Degree of Polynomial", main="LOOCV")
lines(deltas, col="blue")
```



K-Folds Cross Validation

Part 9: Polynomial Models K-Folds

We can also compare error using the K-Folds validation approach. This data is also capture in the GLM.

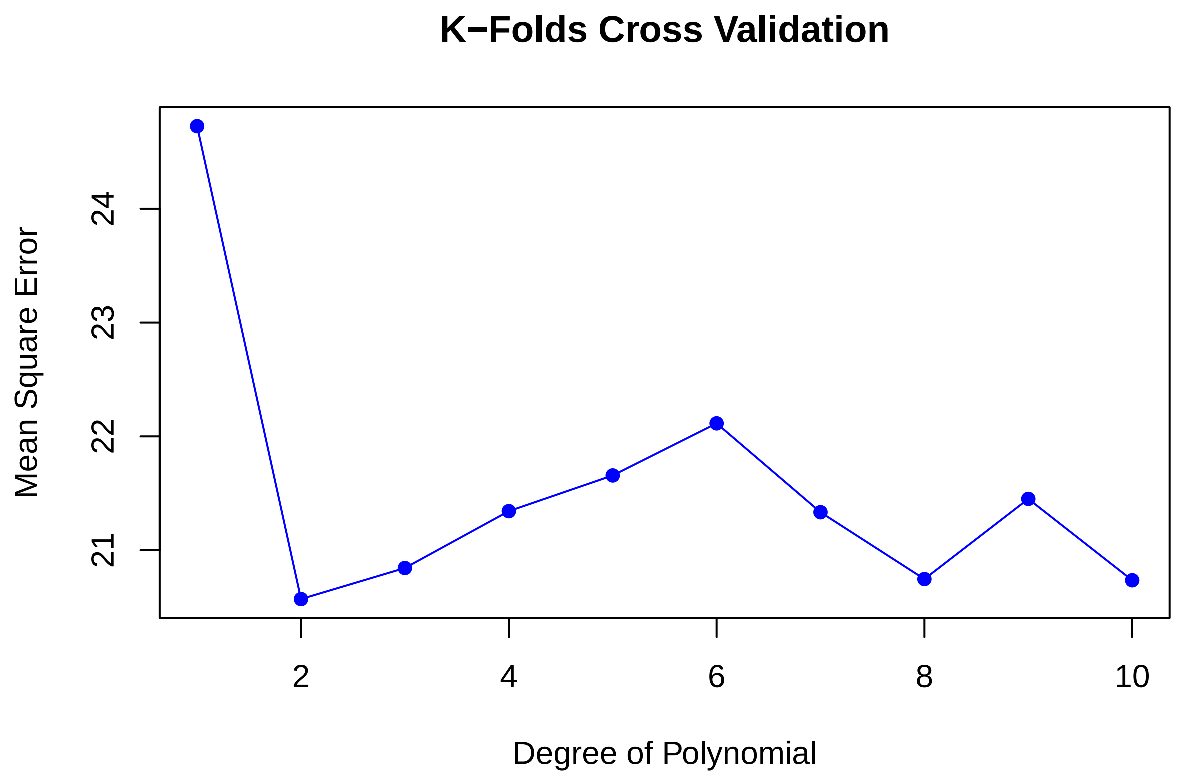
```r
cv.errors <- vector("integer", 10)

for (i in 1:10){

  glm.fit.train <- glm(mpg~poly(horsepower, i), data = train)

  # get the first value from delta 1
  cv.errors[i] = cv.glm(train, glm.fit.train ,K=10 )$delta[1]
}


cv.errors
```

```
##  [1] 24.72535 20.57065 20.84350 21.34309 21.65677 22.11433 21.33356
##  [8] 20.74634 21.45090 20.73569
```

Part 10: K-Folds single Plot

We can plot the data using the same method as shown in part 8.

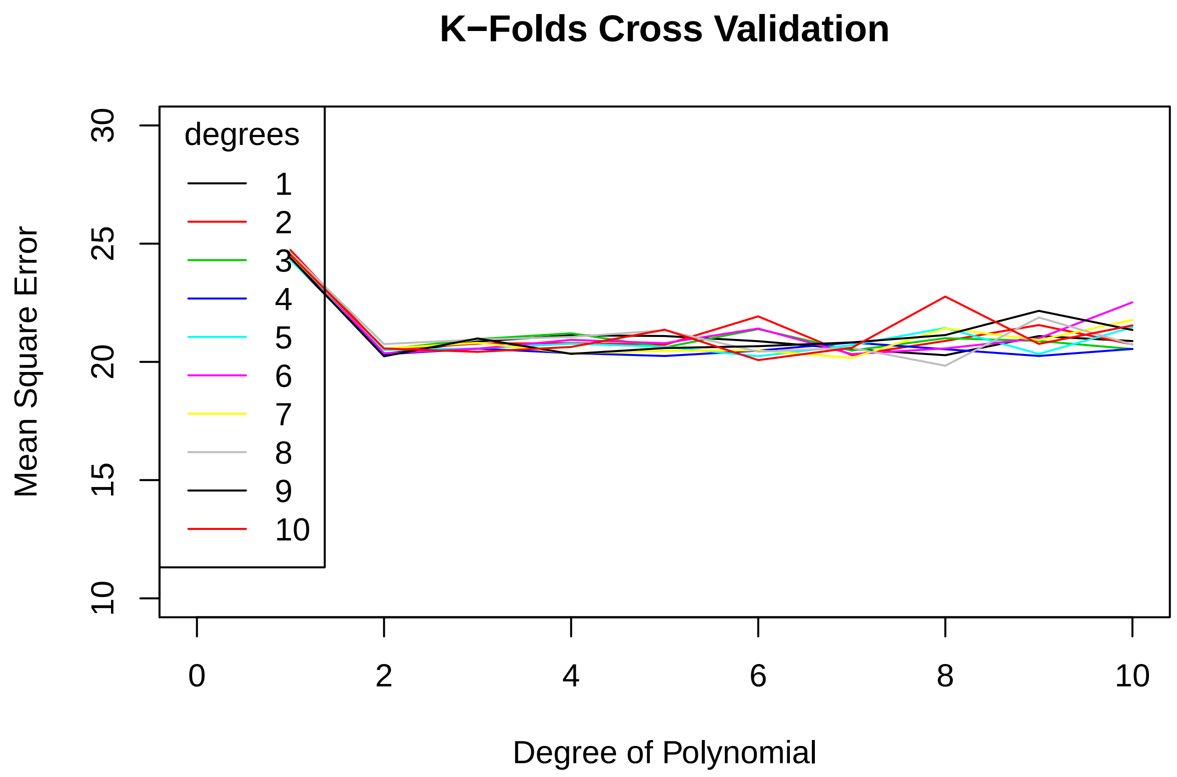
```r
plot(seq(1:10), cv.errors, pch = 16 , col="blue",
     ylab="Mean Square Error", xlab="Degree of Polynomial", main="K-Folds Cross Validation")
lines(cv.errors, col="blue")
```



Part 11: Plot K-Folds

Finally, we can also compare error data for differing seed values, using the same nested loop approach seen in part 6.

```r
plot(0,ylab="Mean Square Error", ylim = c(10,30), xlim= c(0,10),
     xlab="Degree of Polynomial", main="K-Folds Cross Validation")

kfold.errors <- matrix(nrow=10,ncol=10)

for(i in 1:10){

  set.seed(i)

  for(j in 1:10){

    glm.fit.train <- glm(mpg~poly(horsepower, j), data = train)

    kfold.errors[i, j] <- cv.glm(train, glm.fit.train ,K=10)$delta[1]

    }
  lines(kfold.errors[i, ],col=i)
}

legend(title = "degrees", "topleft",c("1","2","3","4","5","6","7","8","9","10"),lty=rep(1,10),col=1:10)
```


K−Folds Cross Validation
degrees
Mean Square Error
Degree of Polynomial