Lab 5 Solutions

Validation set approach

1) Randomly pick half of the data as the training data. Remember to set a seed to make your result repeatable.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.4.2
```

```
set.seed(100)
train <- sample(392,196)
print(train)
```

```
##   [1] 121 101 216  22 182 188 314 143 210  66 239 337 107 152 289 253  77
##  [18] 135 375 258 200 264 372 277 155  63 282 322 370 391 177 336 126 343
##  [35] 249 318  65 224 351  46 117 304 273 378 384 171 271 306  72 106 114
##  [52]  68  81  94 364  86  42  78 338  71 154 215 317 223 147 352 149 145
##  [69]  80 225 133 359 184 309 211 199 346 245 262  29 144 187 286 354  12
##  [86] 178 323  76  92 329 274  64 108 374 316 116 332  37   9 227  96 342
## [103] 308 105 165 197 278 201   4 379 236 293  23  67 269  11 377 334  56
## [120] 230 300 380 128 158  95   8 265 254 146  27 367 345 192 129 150   5
## [137] 392 369 118 160 170 389  36 281  31 181 234 255 257  49 123 355  34
## [154]  41 325 194 275 186 244 163 190 132 111 295  21 229   7 276 347 260
## [171]  98  93 259 247 173 333 207 141  99 130  62 283 237 142  39  73  26
## [188] 280  61 241 285  88 382 303 131  59
```

2) Build a linear regression model based the training data.

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

```
##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto, subset = train)
##
## Coefficients:
## (Intercept)   horsepower
##     39.0570      -0.1501
```

3) Estimate the test MSE based on the other half (as test data)

```
error <- rep(0,3)
error[1] <- mean((Auto$mpg-predict(lm.fit.train, Auto))[-train]^2)
print(error[1])
```

```
## [1] 24.9355
```

4) Now try to build polynomial regression of degree 2 and 3 using `lm(y~poly(x,i))`, where `y` is the response variable, `x` is the predictor variable and `i` is the highest degree of `x`. Compute the test MSE for the two models.

```r
lm.fit.train2 <- lm(mpg~poly(horsepower,2),data=Auto,subset=train)
error[2] <- mean((Auto$mpg-predict(lm.fit.train2, Auto))[-train]^2)

lm.fit.train3 <- lm(mpg~poly(horsepower,3),data=Auto,subset=train)
error[3] <- mean((Auto$mpg-predict(lm.fit.train3, Auto))[-train]^2)


sapply(error,print)
```

```
## [1] 24.9355
## [1] 21.61717
## [1] 21.70125
```

```
## [1] 24.93550 21.61717 21.70125
```

or

```r
for(i in 1:3){
  print(paste("CV error for degree", i, "is", error[i]))
}
```

```
## [1] "CV error for degree 1 is 24.9355023300347"
## [1] "CV error for degree 2 is 21.6171692515163"
## [1] "CV error for degree 3 is 21.7012502912075"
```

5) What conclusion could we draw from the above comparison of degree 1 (linear) and degree 2 (quadratic) and degree 3 (cubic) regression models?

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`.

6) Choose 10 different seeds. For each seed, calculate the test MSE for models of degree from 1 to 10. You may use a nested for-loop to do that. Plot the variability on the results. Can you obtain a similar plot as below.

The first snippet of code records all the result in a matrix. Each row of the matrix represents a seed. Each column of the matrix represents a degree of the polynomial.

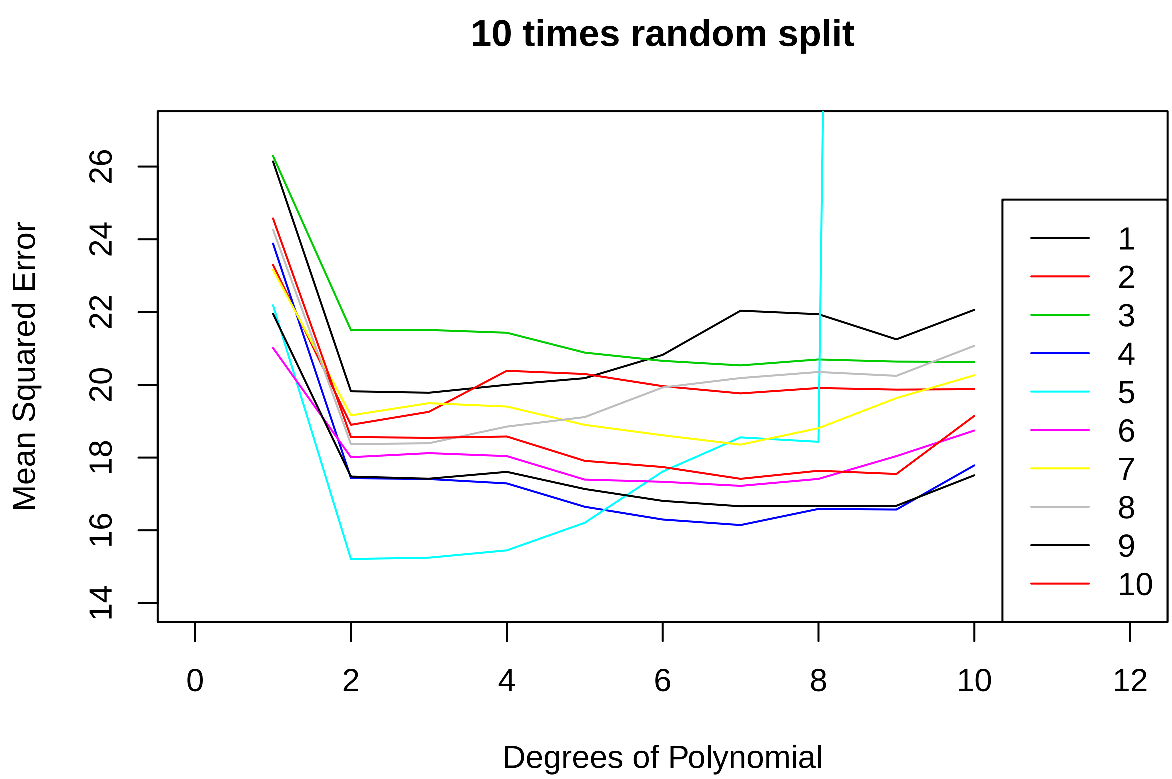
```r
set.seed(1)
train <- sample(392,196)
errors <- rep(0,10)
for(i in 1:10){
     lm.fit.train <- lm(mpg~poly(horsepower,i),data=Auto,subset=train)
      errors[i] <- mean((Auto$mpg-predict(lm.fit.train, Auto))[-train]^2)
}


plot(errors,
     col=1, pch=".", type="l",
     xlab="Degrees of Polynomial",ylab="Mean Squared Error",
     main="10 times random split",
     ylim = c(14,27), xlim= c(0,12))

errorMatrix <- matrix(nrow=10,ncol=10)
errorMatrix[1,] <- errors
```

```
for(i in 2 : 10){
  set.seed(i)
  train <- sample(392,196)
  for(j in 1:10){
    lm.fit.train <- lm(mpg~poly(horsepower,j),data=Auto,subset=train)
    errorMatrix[i,j] <- mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
  }
  lines(errorMatrix[i,],col=i)
}

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



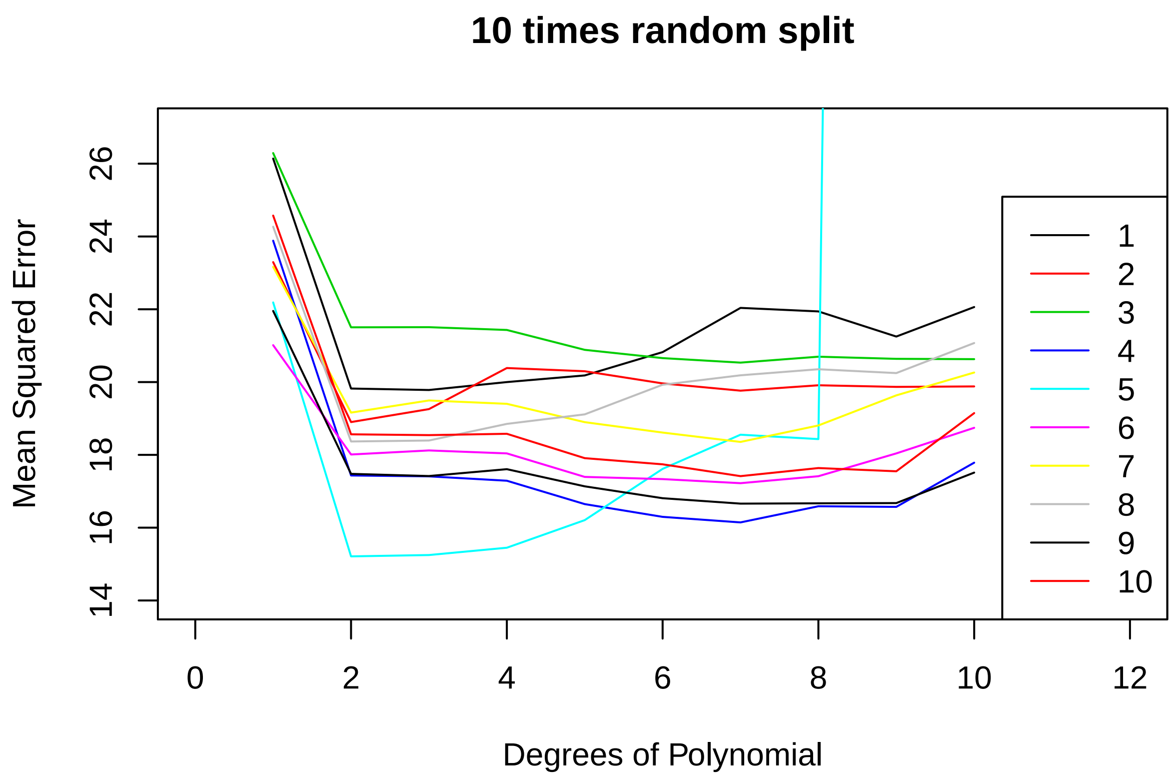
We can simplify the code above as follows:

```
plot(1,
     col=1,pch=".",type="l",
     xlab="Degrees of Polynomial",ylab="Mean Squared Error",
     main="10 times random split",
     ylim = c(14,27), xlim= c(0,12))
#plot(1) creates a plot with one dot at (1,1).
#Normally we use this to plot an "empty" plot.
#We later use lines() to add lines to this plot.

errorMatrix <- matrix(nrow=10,ncol=10)
```

```
for(i in 1 : 10){ #i is the seed.
  #You may change this to for example
  #for(i in seq(100,by = 20, length.out=10))
  set.seed(i)
  train <- sample(392,196)
  for(j in 1:10){#j is the degree of the polynomial
    lm.fit.train <- lm(mpg~poly(horsepower,j),data=Auto,subset=train)
    errorMatrix[i,j] <- mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
  }
  lines(errorMatrix[i,],col=i)
}

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



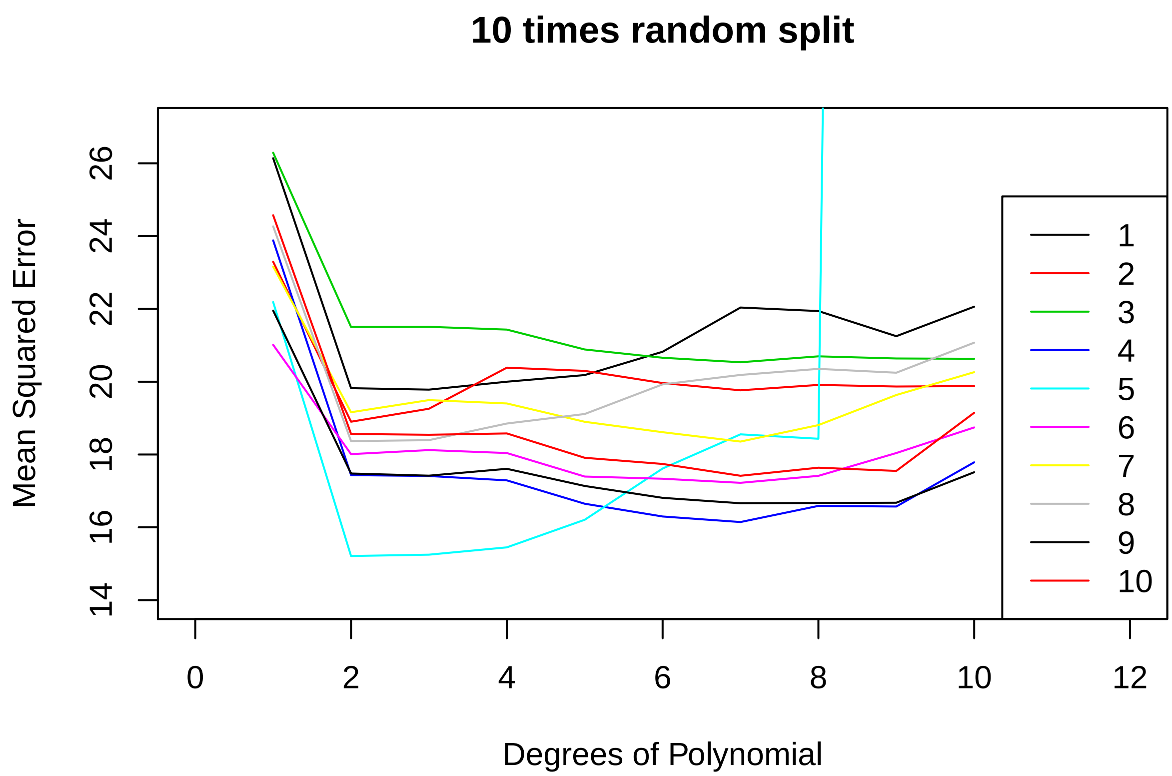
We can also use a single vector instead of a matrix to record and plot the test MSE. But the drawback is that the previous data will be overwritten.

```
plot(1,
     col=1,pch=".",type="l",
     xlab="Degrees of Polynomial",ylab="Mean Squared Error",
     main="10 times random split",
     ylim = c(14,27), xlim= c(0,12))
#plot(1) creates a plot with one dot at (1,1).
#Normally we use this to plot an "empty" plot.
```

```
#We later use lines() to add lines to this plot.

errors <- NULL #create an empty vector called errors
for(i in 1 : 10){ #i is the seed.
  #You may change this to for example
  #for(i in seq(100,by = 20, length.out=10))
  set.seed(i)
  train=sample(392,196)
  for(j in 1:10){#j is the degree of the polynomial
    lm.fit.train <- lm(mpg~poly(horsepower,j),data=Auto,subset=train)
    errors[j] <- mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
  }
  lines(errors,col=i)
}

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



(II) LOOCV

7) Experiment on the LOOCV for increasingly complex polynomial fits. More specifically, write a for-loop to increase the degree i, as in `lm(y~poly(x,i))`, from 1 to 10 and record the LOOCV estimate for the test error for each degree.

```r
library(boot)
```

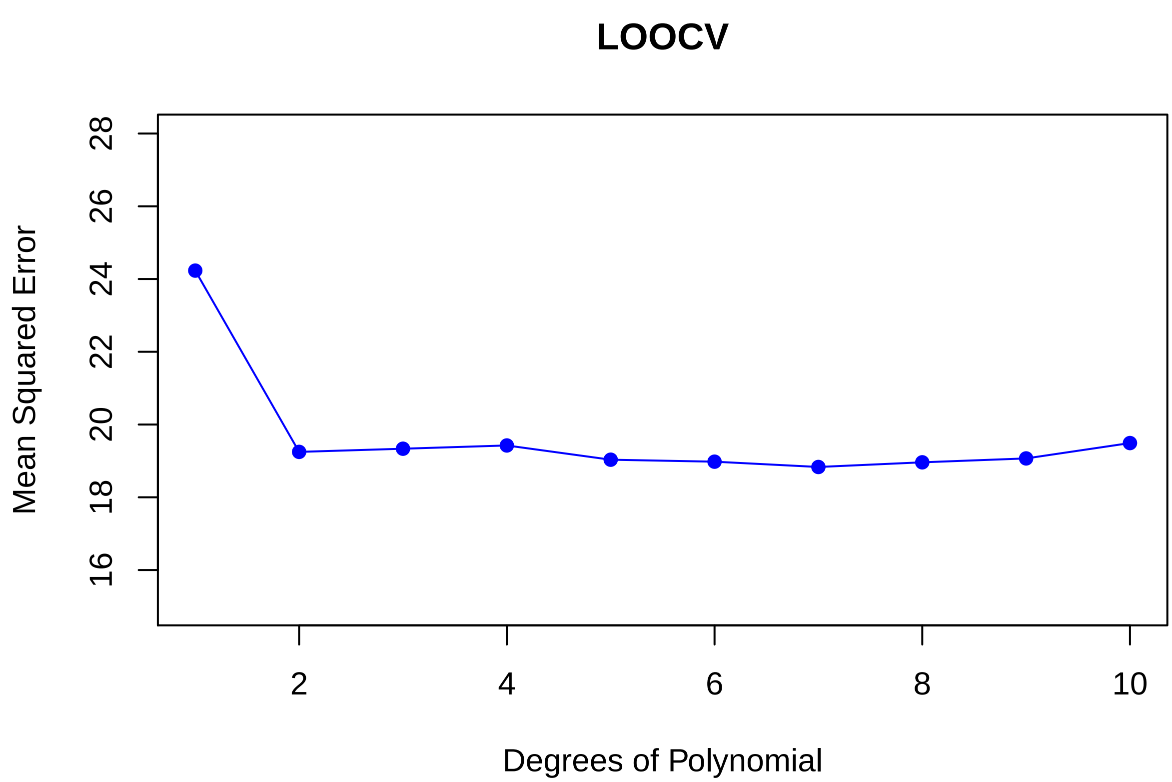
```
## Warning: package 'boot' was built under R version 3.4.2
```

```r
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]
}
#the above for loop takes a while
print(cv.error)
```

```
##  [1] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305
##  [8] 18.96115 19.06863 19.49093
```

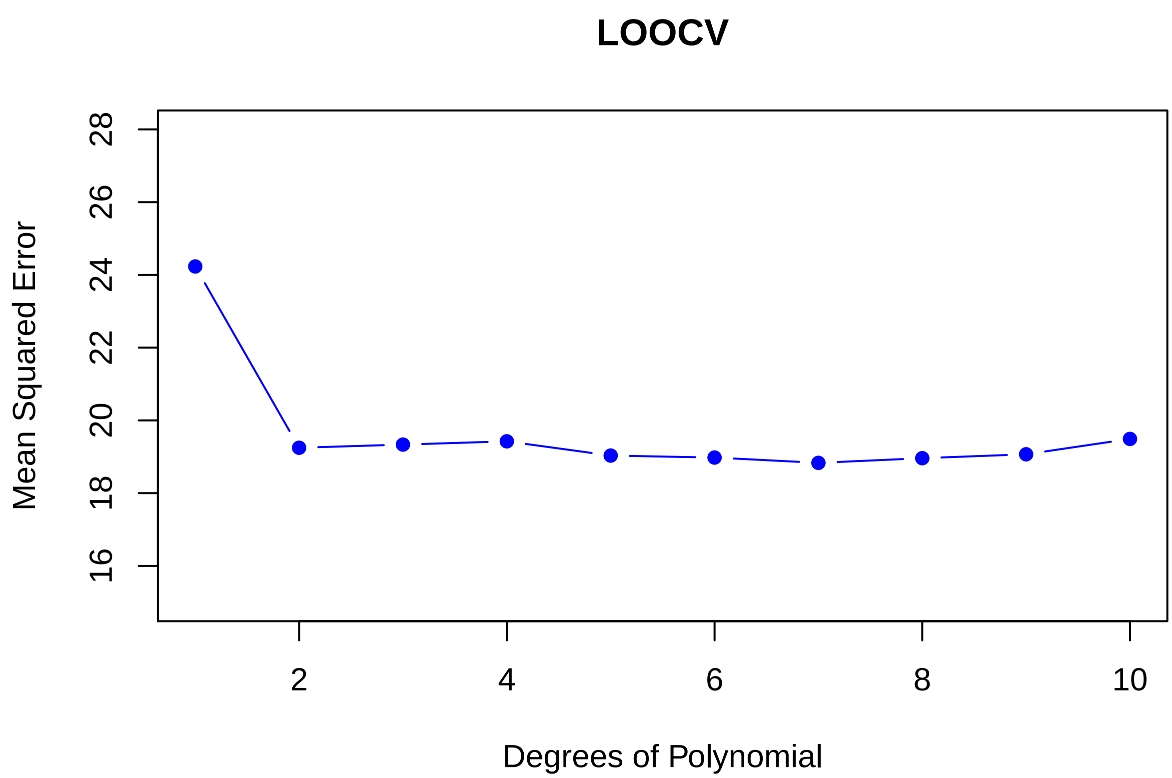
8) Plot the result from 7) where x-axis is the degree i and y-axis is the LOOCV estimate for the test error.

```r
plot(cv.error,
     col="blue",pch=16,
     xlab="Degrees of Polynomial",ylab="Mean Squared Error",
     main="LOOCV",ylim=c(15,28))
lines(cv.error,col="blue")
```



or use the following to make the line broken:

```r
plot(cv.error,
     col="blue",pch=16,
     xlab="Degrees of Polynomial",ylab="Mean Squared Error",
     main="LOOCV",
     ylim=c(15,28),type="b") #type="b" means broken lines
```



(III) K-fold CV

9) Set a seed. Write a for-loop to increase the degree `i`, as in `lm(y~poly(x,i))`, from 1 to 10 and record the 10-fold CV estimate for the test error for each degree.

10) Plot the result from 9) where x-axis is the degree `i` and y-axis is the 10-fold CV estimate for the test error.

11) Set 9 different seeds and repeat 9) and 10). Plot all the results into one plot.

```r
#plot(1) creates an empty plot as a base.
plot(1,
     type="l",
     xlab="Degrees of Polynomial",ylab="Mean Squared Error",
     main="K-fold CV",
     ylim = c(14,27), xlim = c(0,12))

cv.errors <- rep(0,10)
```

```r
for(i in 1:10){ #10 lines
        set.seed(i)
        for(j in 1:10){#10 degrees
          glm.fit <- glm(mpg~poly(horsepower,j),data=Auto)
          cv.errors[j] <- cv.glm(Auto,glm.fit,K=10)$delta[1]
    }
    lines(cv.errors,col=i)
}

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