Lab5_Answer

Anyi Guo

06/11/2018

(I) 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")
set.seed(100)
train<-sample(nrow(Auto),nrow(Auto)/2)
```

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

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

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

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

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

test MSE for linear model is `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.

```
lm.fit2.train<-lm(mpg~poly(horsepower,2),data=Auto,subset=train)
lm.fit3.train<-lm(mpg~poly(horsepower,3),data=Auto,subset=train)
mean((Auto$mpg-predict(lm.fit2.train,Auto))[-train]^2)
```

```
## [1] 21.61717
```

```
mean((Auto$mpg-predict(lm.fit3.train,Auto))[-train]^2)
```

```
## [1] 21.70125
```

MSE for degree 2(quadratic) is `21.61717`. (best out of 3) MSE for degree 3(cubic) is `21.70125`.

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

degree 2 has the smallest MSE, and is thus the best out of three.

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 in Figure 1.

- Hint: In order to do that you need to plot one curve first, and repeat the same procedure for another 9 times (using a for-loop) where each time a different seed is chosen.

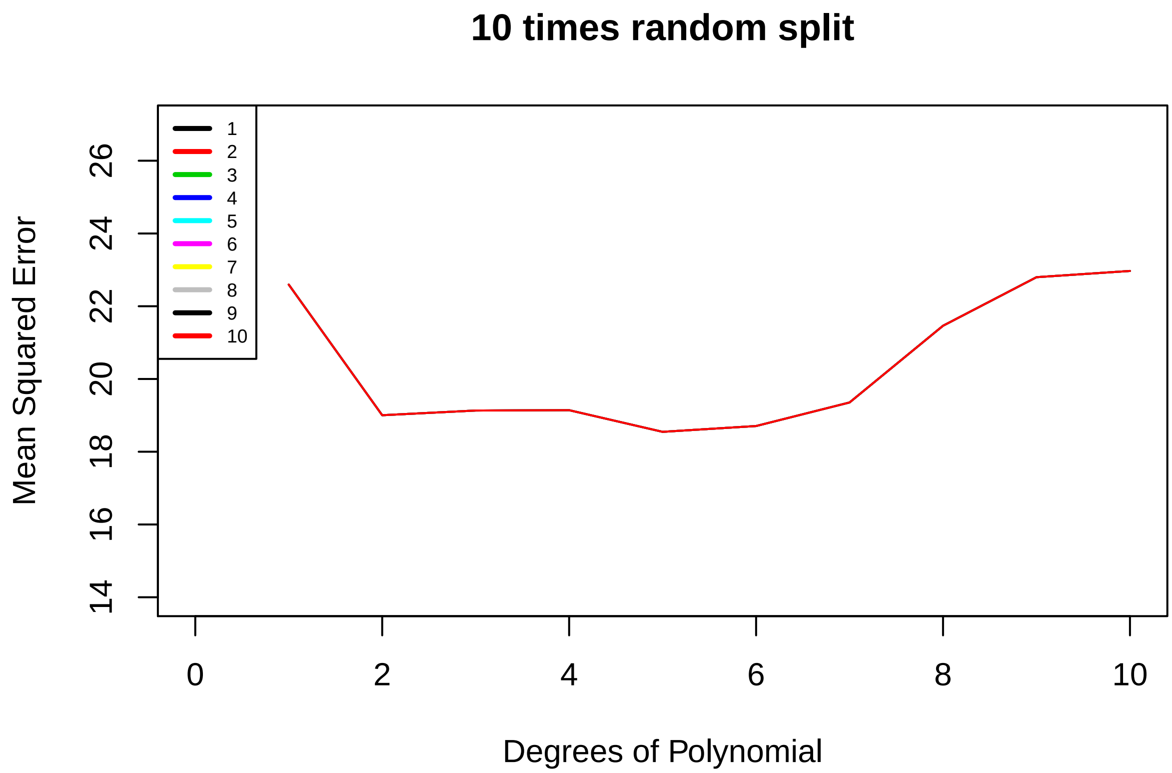
In order to plot one curve, you need to obtain a vector of size 10, where each element of the vector records the test MSE of the model with degree i (i = 1, 2, .10). This can be implemented by a for-loop to go through degree from 1 to 10.

First method - keeping only one vector for errors. It is easier to understand, but all previous data will be lost.

```
plot(0, xlab="Degrees of Polynomial",ylab="Mean Squared Error", main="10 times random split",ylim = c(1


train<-sample(392,196)
errors<-rep(0,10)

for(i in 2:10){
        set.seed(i)
        for(j in 1:10){
                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("topleft",c("1","2","3","4","5","6","7","8","9","10"),lty=rep(1,10),col=1:10,lwd=rep(2.5,10),cex
```



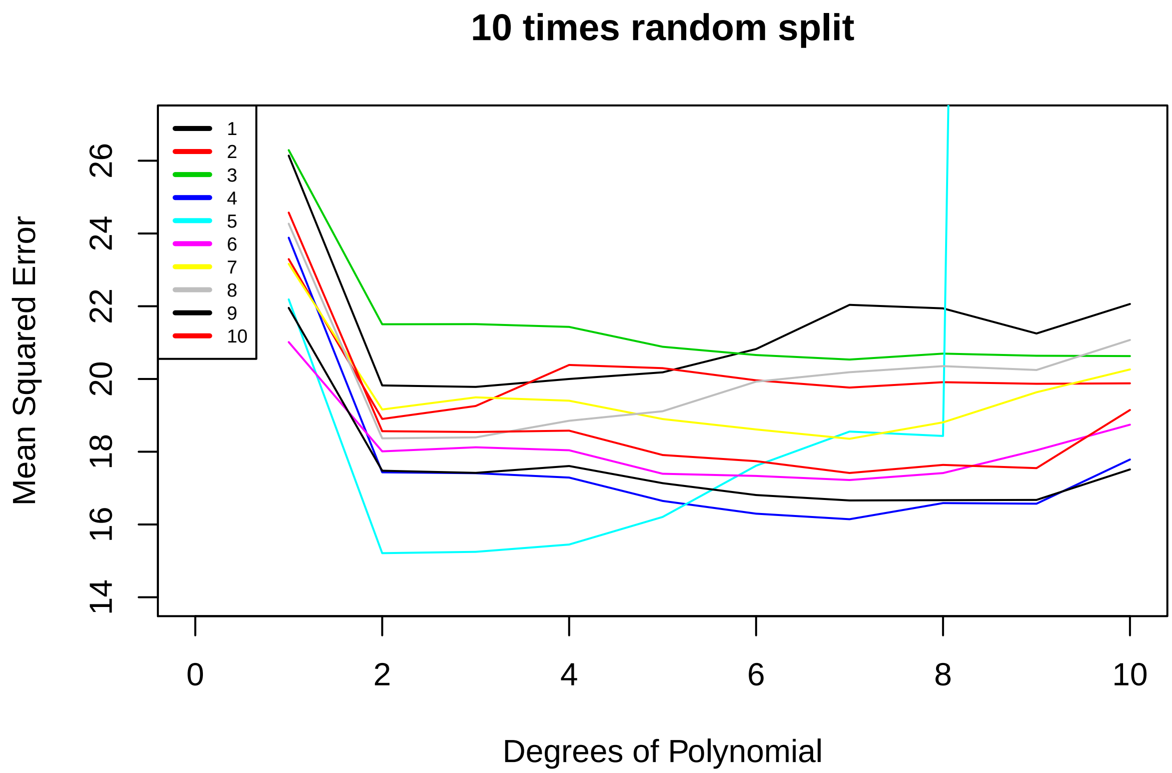
Second method - keeping a two-dimensional matrix. It will store all the errors calculated so far. :

```
plot(0, xlab="Degrees of Polynomial",ylab="Mean Squared Error", main="10 times random split",ylim = c(14

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

for(i in 1:10){
        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("topleft",c("1","2","3","4","5","6","7","8","9","10"),lty=rep(1,10),col=1:10,lwd=rep(2.5,10),cex
```



A snippet that returns the square of each number in foo.squared

```
foo=seq(1,100,by=2)
foo.squared=NULL
for(i in 1:50){
        foo.squared[i]=foo[i]^2
}
print(foo.squared)
```

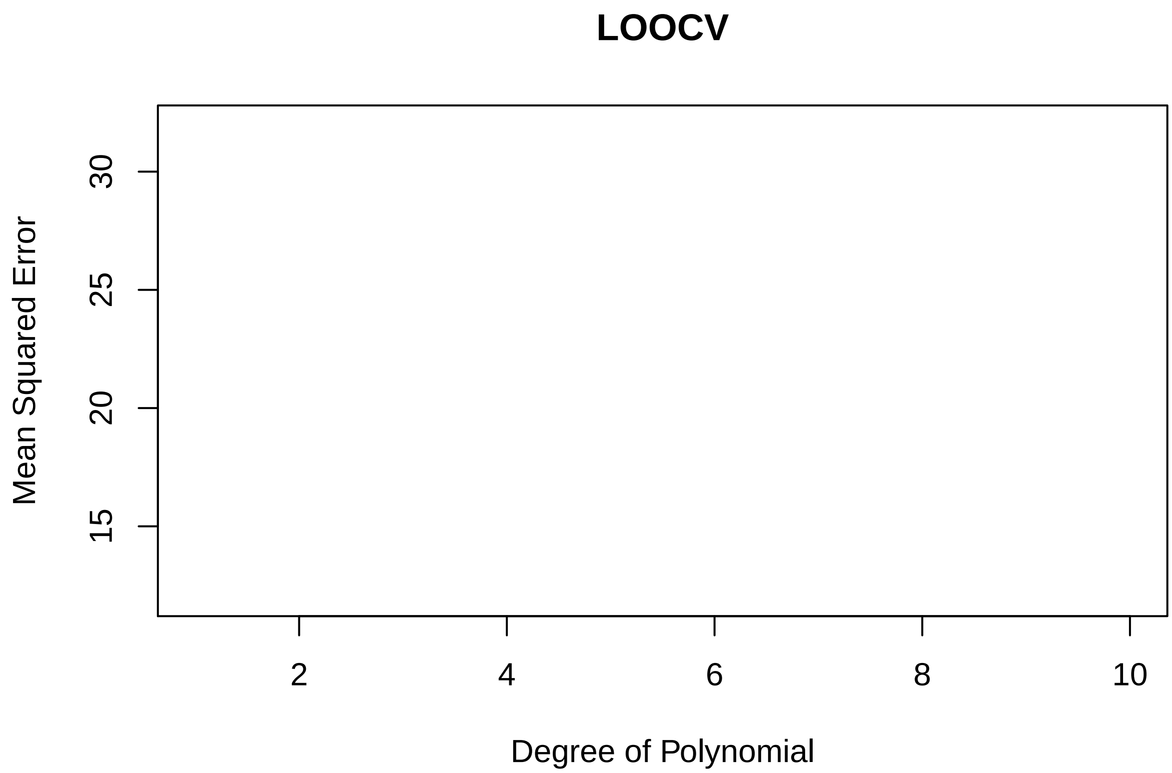
```
##  [1]    1    9   25   49   81  121  169  225  289  361  441  529  625  729
## [15]  841  961 1089 1225 1369 1521 1681 1849 2025 2209 2401 2601 2809 3025
## [29] 3249 3481 3721 3969 4225 4489 4761 5041 5329 5625 5929 6241 6561 6889
## [43] 7225 7569 7921 8281 8649 9025 9409 9801
```

(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.

8) Plot the result from 7) where x-axis is the degree i and y-axis is the LOOCV estimate for the test error. Can you plot a similar one as in Figure 2?

```
library(boot)
plot(0,xlab="Degree of Polynomial",ylab="Mean Squared Error", main = "LOOCV",xlim=c(1,10),ylim=c(12,32)
```




```r
errorMatrix<-matrix(nrow = 10,ncol = 2)
errors <-rep(0,10)
for(i in 1:10){
        set.seed(i)
        glm.fit<-glm(mpg~poly(horsepower,i),data=Auto)
        cv.err<-cv.glm(Auto,glm.fit)
        errors[i]<-cv.err$delta
}
```

```
## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement
```

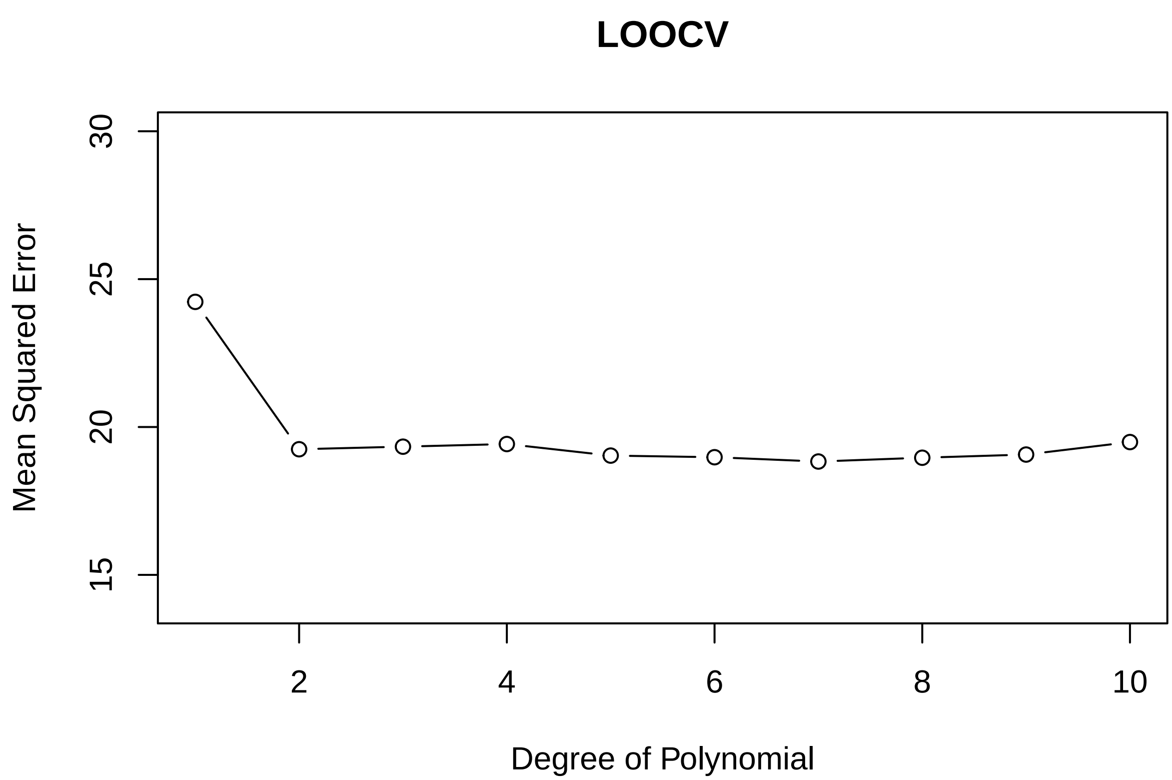
```
## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement

## Warning in errors[i] <- cv.err$delta: le nombre d'objets à remplacer n'est
## pas multiple de la taille du remplacement
```

```
errorMatrix[,2]<-errors
errorMatrix[,1]<-seq(1,10)
plot(errorMatrix,ylim=c(14,30),type="b",xlab="Degree of Polynomial",ylab="Mean Squared Error",main="LOO
```



(III) K-fold CV

Implement k-fold CV by passing the argument K in cv.glm(data, glmfit, cost, K). The errors are recorded in delta. There are two numbers associated with delta: • The first number is the raw/standard CV estimate of prediction error. • The second number is the adjusted CV estimate. The adjustment is designed to compensate for the bias introduced by not using leave-one-out cross-validation.

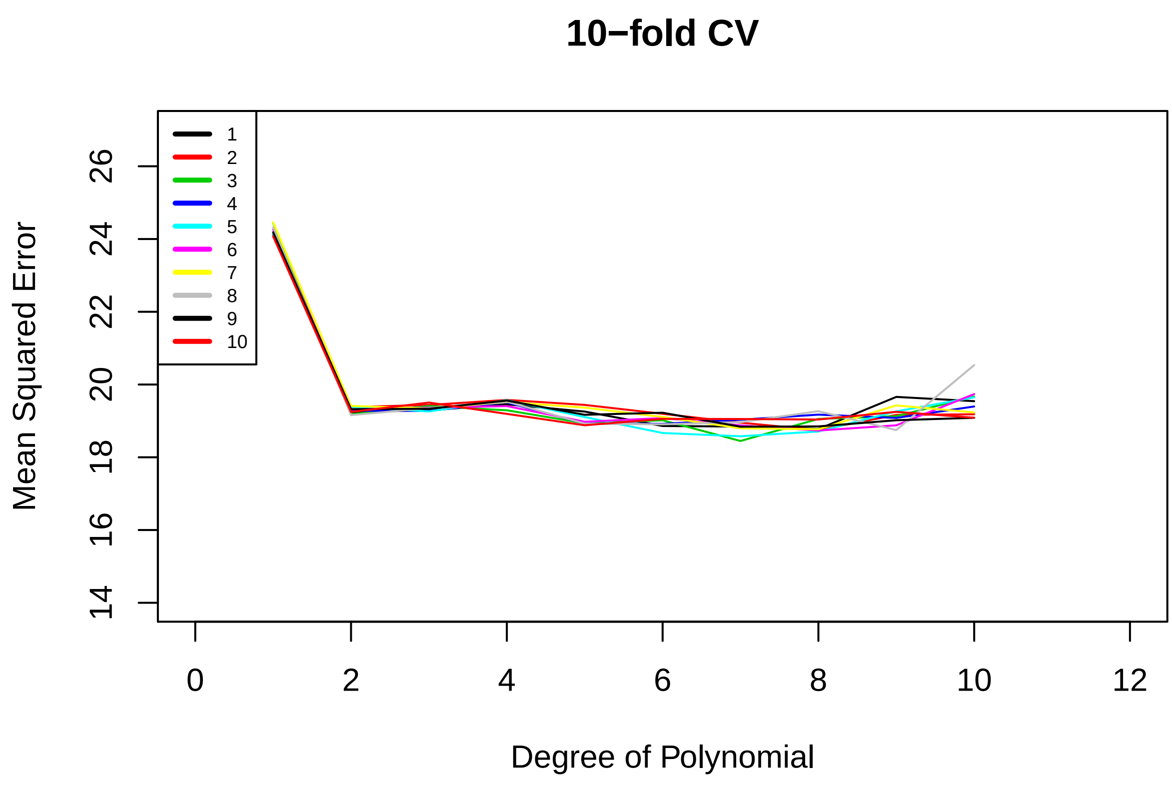
It is sufficient to report the raw CV error to estimate the test errors. The following three questions can be answered by one chunk of code.

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.

```
plot(1,type="l",xlab="Degree of Polynomial",ylab="Mean Squared Error",main="10-fold CV", xlim=c(0,12),yl

cv.errors<-rep(0,10)
for(i in 1:10){
        set.seed(i)
       for(j in 1:10){
               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("topleft",c("1","2","3","4","5","6","7","8","9","10"),lty=rep(1,10),col=1:10,lwd=rep(2.5,10),cex
```



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 like the one in Figure 3.