Session 5(a) Assessing model accuracy

```
> library(ISLR)
```

Slide 5(a)-page 13: **How to calculate MSE in R?**

MSE in Regression:

```
> fix(Auto)
> lm.fit.Auto=lm(mpg~horsepower,data=Auto)
> mean((Auto$mpg-predict(lm.fit.Auto,Auto))^2)
> [1] 23.94366
```

Slide 5(a)-p16: **How to Calculate Error Rate in R**

Error rate in Classification:

```
> fix(Default)
#multiple logistic regression
>
glm.fit=glm(default~income+balance+student,data=Default,family=binomial)
> summary(glm.fit)

Call:
glm(formula = default ~ income + balance + student, family = binomial,
    data = Default)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4691  -0.1418  -0.0557  -0.0203   3.7383

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
income       3.033e-06  8.203e-06   0.370  0.71152
balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2920.6  on 9999  degrees of freedom
Residual deviance: 1571.5  on 9996  degrees of freedom
AIC: 1579.5

Number of Fisher Scoring iterations: 8

#making predictions as probabilities
> glm.probs=predict(glm.fit,type="response")
```

```
> glm.probs[1:10]
           1            2            3            4            5
1.428724e-03 1.122204e-03 9.812272e-03 4.415893e-04 1.935506e-03
           6            7            8            9           10
1.989518e-03 2.333767e-03 1.086718e-03 1.638333e-02 2.080617e-05
```

#The contrasts function indicates that R has created a dummy variable with a 1 for Yes.

```
> contrasts(Default$default)
    Yes
No    0
Yes   1
> length(Default$default)
[1] 10000
```

#create a vector for predicting yes or no. It has the same length as Default$default, and has “No” as the initial values.

```
> glm.pred=rep("No",10000)
```

#Set those whose prob > 0.5 to be “Yes”

```
> glm.pred[glm.probs>.5]="Yes"
```

#The table function shows the confusion matrix

```
> table(glm.pred,Default$default)

glm.pred   No  Yes
     No  9627  228
     Yes   40  105
```

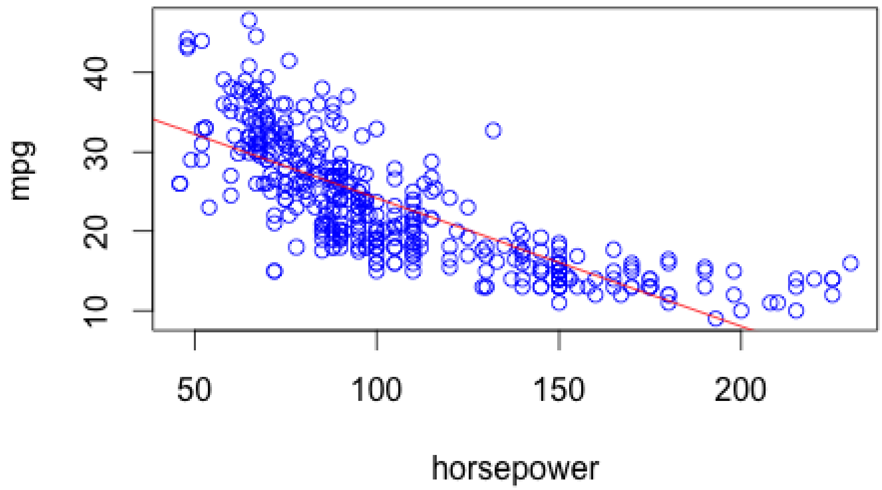
#using mean to show the accuracy and error rate

```
> mean(glm.pred==Default$default)
[1] 0.9732
> mean(glm.pred!=Default$default)
[1] 0.0268
```

Session 5(b) Cross validation

Slide 5(b) p.6 **Example: Auto Data**

```
>
plot(Auto$horsepower,Auto$mpg,xlab="horsepower",ylab="mpg",col="blue")
> abline(lm.fit.train,col="red")
```

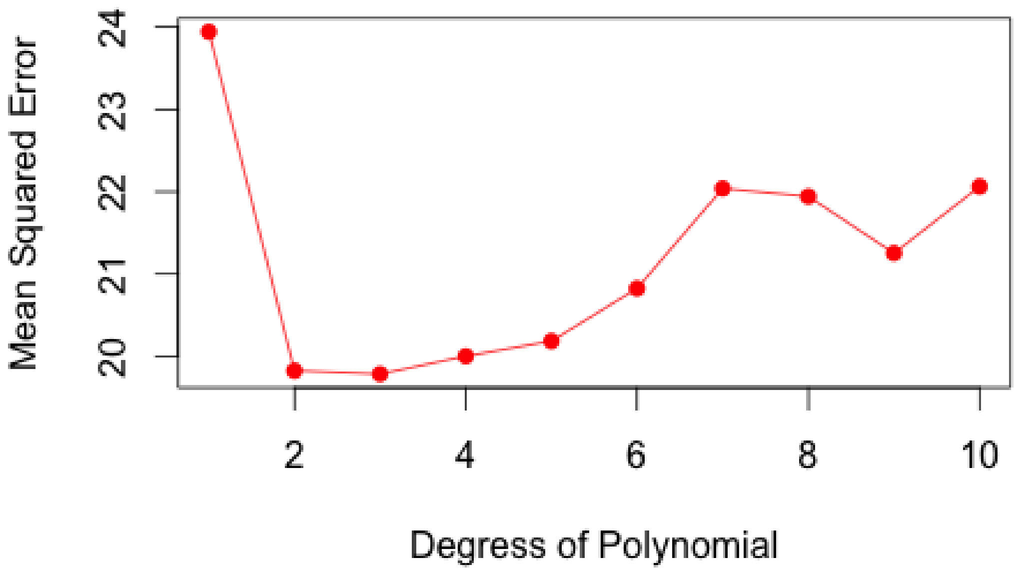


Slide 5(b)-p9: **Results: Auto Data**

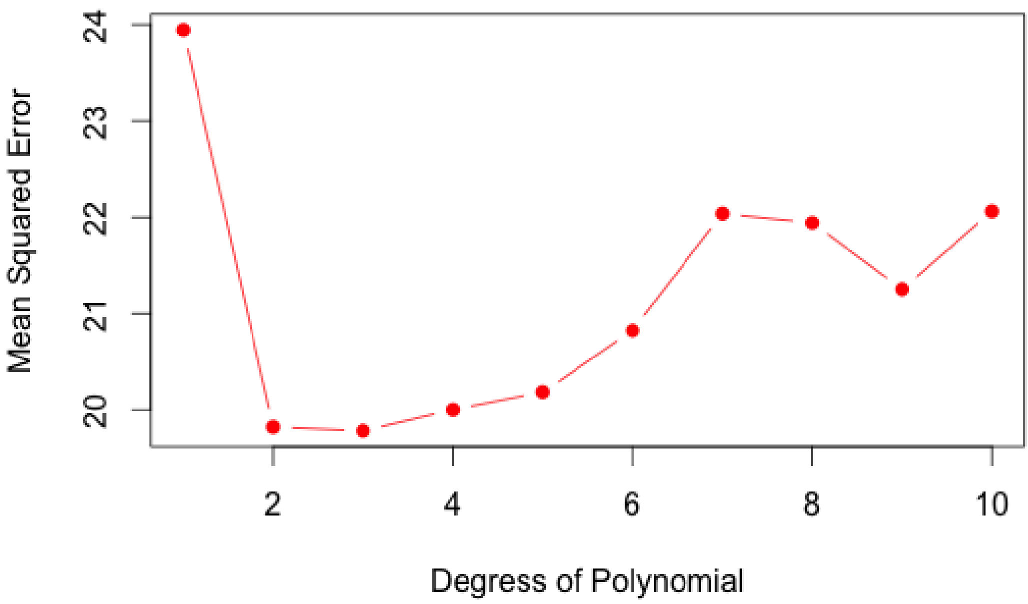
```
> set.seed(1)
> train=sample(392,196)
> errors<-rep(0,10)
> errors[1]<-23.94366
> for(i in 2:10){
+     lm.fit.train<-lm(mpg~poly(horsepower,i),data=Auto,subset=train)
+      errors[i]<-mean((Auto$mpg-predict(lm.fit.train, Auto))[-train]^2)
+     }
```

Plot left:

```
> plot(errors,col="red",pch=16,xlab="Degress of Polynomial",ylab="Mean
Squared Error")
> lines(errors,col="red")
```



#this is to plot in broken lines:
>plot(errors,col="red",pch=16,xlab="Degress of Polynomial",ylab="Mean Squared Error",type='b')
#to add a line (e.g. errors_low: 0.5 lower than errors) is simply:
>errors_low=errors-0.5
>lines(errors_low)



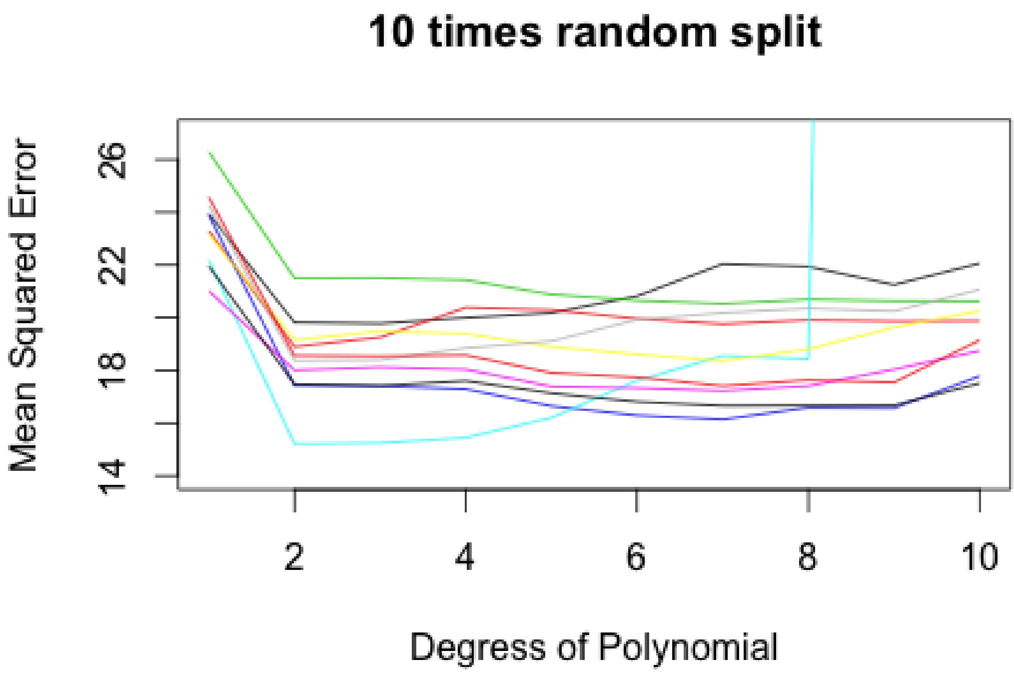
Slide 5(b)-9 **Results: Auto Data**

#Plot the figure on the right

the continuous lines without dots on the right can be achieved by:

```
> plot(errors,col="red", xlab="Degress of Polynomial",ylab="Mean Squared Error",type="l",main="10 times random split")
```

```
> errorMatrix<-matrix(nrow=10,ncol=10)
> errorMatrix[1,]=errors
> plot(errors, col=1, xlab="Degress of Polynomial", ylab="Mean Squared Error", type="l", main="10 times random split", ylim = c(14,27))
> 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)
+      }
```



Slide 5(b)-p13: **Perform LOOCV in R**

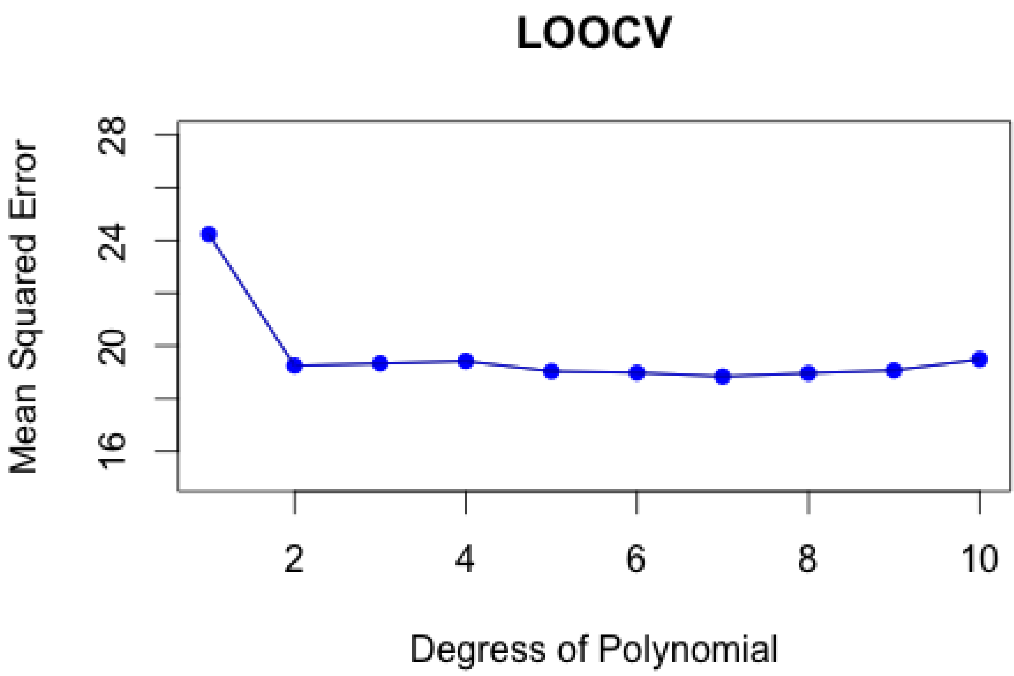
```
> glm.fit=glm(mpg~horsepower,data=Auto)
># This is the same as lm.fit(mpg~horsepower,data=Auto)
> library(boot) #cv.glm() is in the boot library
> cv.err=cv.glm(Auto,glm.fit)
#cv.glm() does the LOOCV
>
> cv.err$delta
[1] 24.23151 24.23114
```

The MSE is 24.23151.

Slide 5(b)-p17 left: **Auto Data: LOOCV vs. k-fold CV**
Next, we plot the LOOCV

```
> 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
> cv.error
 [1] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305
18.96115
 [9] 19.06863 19.49093

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

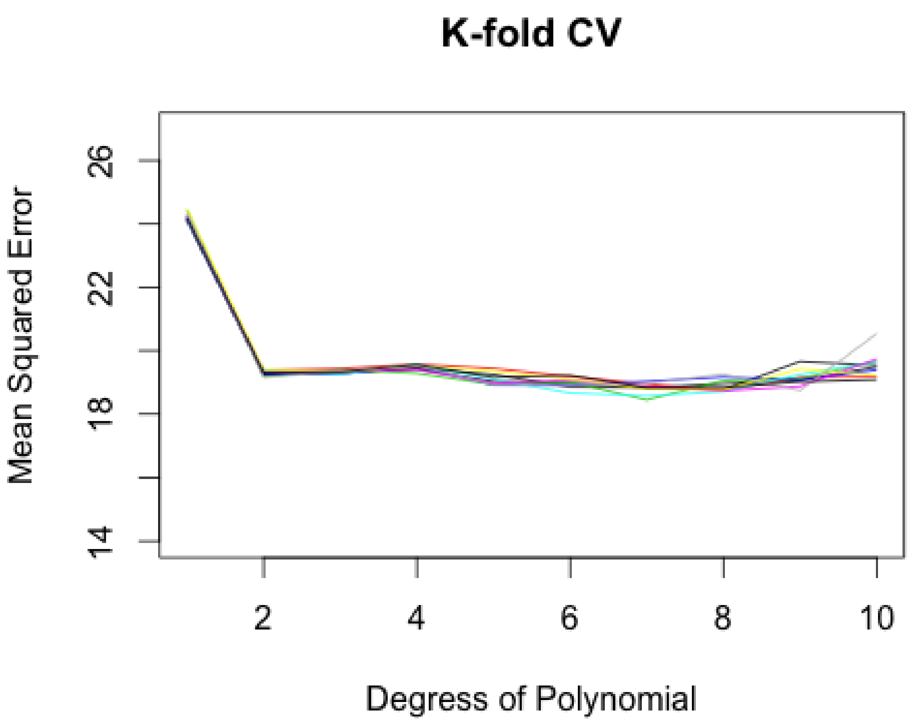



```
#or use the following to make the line broken:
>  plot(cv.error,col="blue",pch=16,xlab="Degress of Polynomial",ylab="Mean
Squared Error",main="LOOCV",ylim=c(15,28),type=”b”)
```

Slide 5(b)-p17 right: **Auto Data: LOOCV vs. k-fold CV**

```
> plot(cv.error,col=1,pch=".",xlab="Degress of Polynomial",ylab="Mean Squared Error",type="l",main="K-fold CV",ylim = c(14,27))
> cv.error.matrix=matrix(nrow=9,ncol=10)

>for(i in 1:9){
        set.seed(i)
        for(j in 1:10){
        glm.fit=glm(mpg~poly(horsepower,j),data=Auto)
        cv.error.matrix[i,j]<-cv.glm(Auto,glm.fit,K=10)$delta[1]
    }
    lines(cv.error.matrix[i,],col=i)
}
```




```
#another way of doing it:
> plot(1,xlab="Degress of Polynomial",ylab="Mean Squared Error",type="l",main="K-fold CV",ylim = c(14,27))

>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.error.matrix[j]<-cv.glm(Auto,glm.fit,K=10)$delta[1]
    }
    lines(cv.error.matrix,col=i)
}
```