Chapter 5: Resampling Methods Applied Exercise

Phuong Dong Le

```r
library(class)
library(MASS)
library(ISLR)
library(RColorBrewer)
library(corrplot)
library(boot)
### GGplot:
library(ggplot2)
library(ggthemes)
library(tidyverse)
### Styling for tables and figures:
library(kableExtra)
library(gridExtra)
```

Exercise 5: This question involves the data set `Default`

```r
attach(Default)
Default = Default[complete.cases(Default),]
dim(Default)
```

```
## [1] 10000     4
```

```r
str(Default)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

```r
summary(Default)
```

```
##  default    student       balance           income     
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772  
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340  
##                        Median : 823.6   Median :34553  
##                        Mean   : 835.4   Mean   :33517  
##                        3rd Qu.:1166.3   3rd Qu.:43808  
##                        Max.   :2654.3   Max.   :73554  
```

Part (a)

- We fit a logistic regression model that uses `Income` and `balance` to predict `default`:

```r
glm.fit = glm(default ~ balance + income, data = Default, family = "binomial")
summary(glm.fit)
```

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

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## ---
## 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: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

Part (b)

- We use the validation set approach, and estimate the test error of this model.

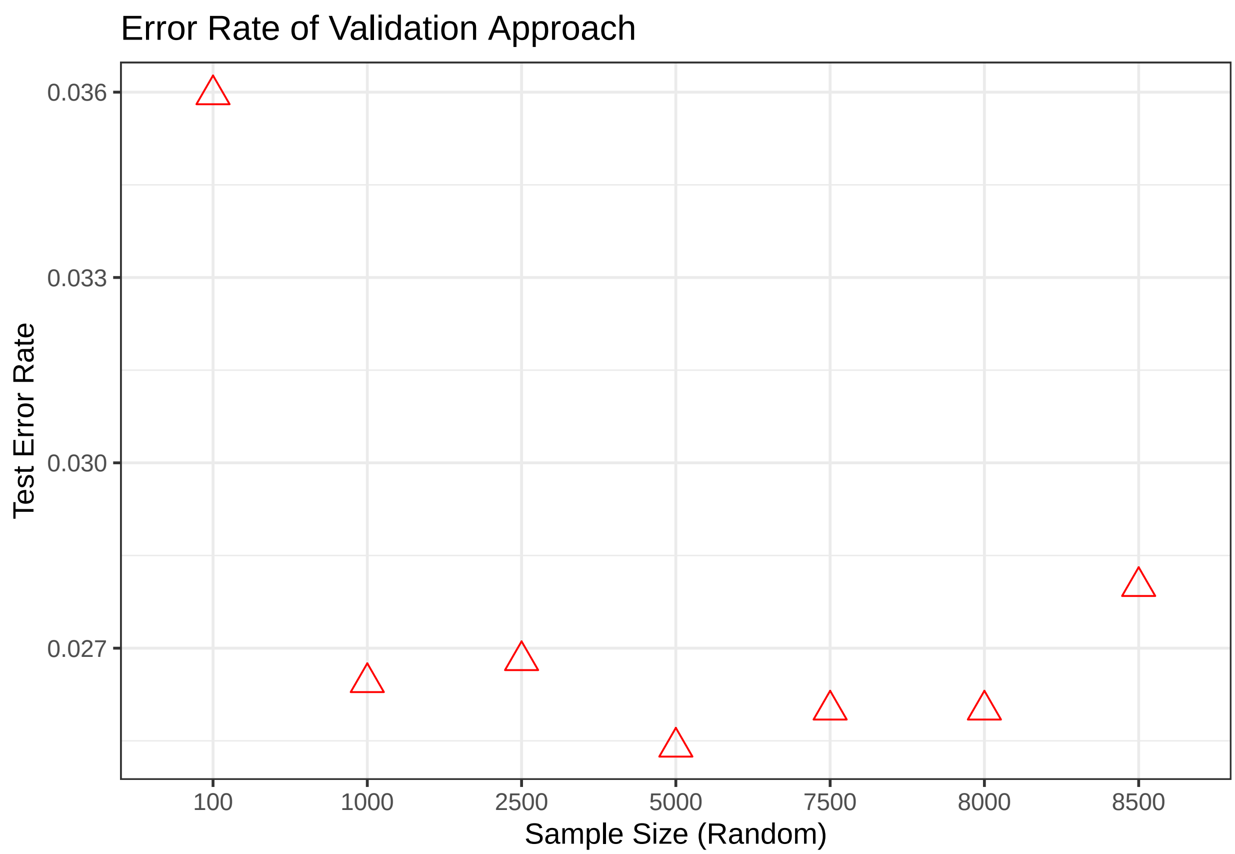
```r
n = dim(Default)[1]
```

- We fit multiple logistic regression models:

```r
Size = c(0.01,0.1,0.25,0.5,0.75, 0.8, 0.85)
SampleSize = Size*n
glmpred = rep(NA)
ErrorRate = rep(NA)
for(i in 1:length(SampleSize)){
### Split into training and validation set:
  set.seed(1)
train = sample(x = n, size = SampleSize[i])
### Training Set:
Default.train = Default[train,]
Default.test = Default[-train,]
### Testing Set:
default.train = Default[train,]$default
default.test  = Default[-train,]$default

glm.fit = glm(default ~ income + balance, data = Default,
               subset = train,
               family = "binomial")
glm.probs = predict(glm.fit, Default.test, type = "response")
glm.pred = rep("No", length(glm.probs))
glm.pred[glm.probs > 0.5] = "Yes"
ErrorRate[i] = mean(glm.pred !=default.test)


}
```

```r
DataErrorRate  = data.frame(SampleSize, ErrorRate)
```

```r
ErrorPlot = ggplot(data = DataErrorRate,
                   aes(x = factor(SampleSize),
                       y = ErrorRate)) +
  geom_point(pch = 2, size = 4, color = "red") +
  ggtitle(label = "Error Rate of Validation Approach") +
  xlab(label = "Sample Size (Random)") +
  ylab(label = "Test Error Rate") + theme_bw()



print(ErrorPlot)
```



Part (d)

- We consider a logistic regression model to predicct the probability of default using income, balance and a dummy variable student:

```r
glm.fit = glm(default ~ income + balance + factor(student),
              data = Default, family = "binomial")
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance + factor(student), family = "binomial",
##     data = Default)
##
## Deviance Residuals:
```

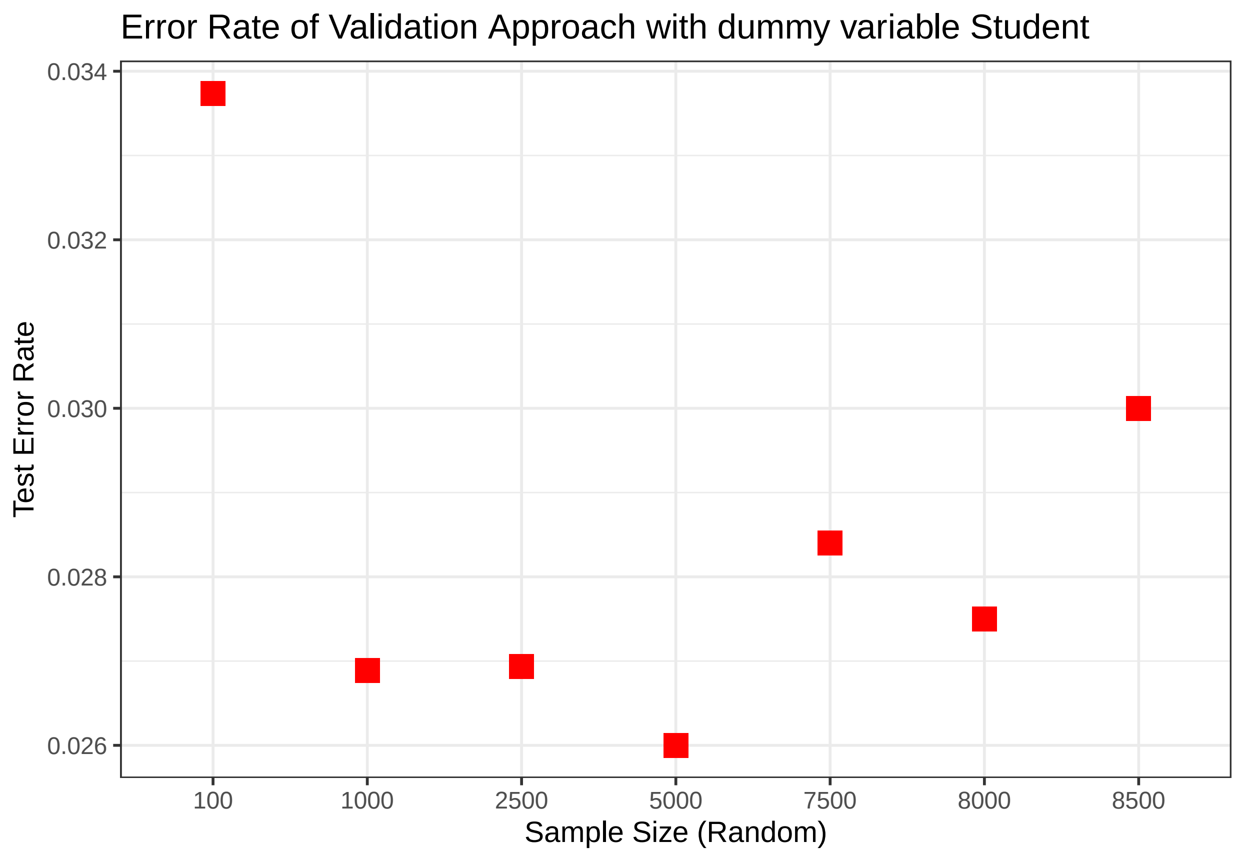
```
##     Min       1Q   Median       3Q      Max
## -2.4691  -0.1418  -0.0557  -0.0203   3.7383
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
## income               3.033e-06  8.203e-06   0.370  0.71152
## balance              5.737e-03  2.319e-04  24.738  < 2e-16 ***
## factor(student)Yes -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
```

- Repeat Part B for this particular model:

```r
Size = c(0.01,0.1,0.25,0.5,0.75, 0.8, 0.85)
SampleSize = Size*n
glmpred = rep(NA)
ErrorRate = rep(NA)
for(i in 1:length(SampleSize)){
### Split into training and validation set:
  set.seed(1)
train = sample(x = n, size = SampleSize[i])
### Training Set:
Default.train = Default[train,]
Default.test = Default[-train,]
### Testing Set:
default.train = Default[train,]$default
default.test  = Default[-train,]$default

glm.fit = glm(default ~ income + balance + factor(student), data = Default,
               subset = train,
               family = "binomial")
glm.probs = predict(glm.fit, Default.test, type = "response")
glm.pred = rep("No", length(glm.probs))
glm.pred[glm.probs > 0.5] = "Yes"
ErrorRate[i] = mean(glm.pred !=default.test)
}
```

```r
DataErrorRate  = data.frame(SampleSize, ErrorRate)

ErrorPlot = ggplot(data = DataErrorRate,
                   aes(x = factor(SampleSize),
                       y = ErrorRate)) +
  geom_point(pch = 15, size = 4, color = "red") +
  ggtitle(label = "Error Rate of Validation Approach with dummy variable Student") +
  xlab(label = "Sample Size (Random)") +
  ylab(label = "Test Error Rate") + theme_bw()
```

```r
print(ErrorPlot)
```



Exercise 6: Computing the logistic regression coefficients in 2 different ways: bootstrap and using the stadard formula in the `glm()`

Part (a)

```r
glm.fit = glm(default ~ income + balance, data = Default,
              family = "binomial")

summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
```

```
## ---
## 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: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

- The standard error estimated are outlined in the output above.

Part (b)

- We write boot.fn function takes input data Default and index output the coefficients estimates for income and balance:

```r
n.len = dim(Default)[1]
boot.fn = function(data, index)
{
  form.model = default ~ income + balance
  glm.fit = glm(formula = form.model, data = data,
                family = "binomial",
                subset = index)
  coef.est = coefficients(glm.fit)
  return(coef.est)
}

### Return the est. coefficient as using full data set:
boot.fn(data = Default, index = 1:n.len)
```

```
##   (Intercept)        income       balance
## -1.154047e+01  2.080898e-05  5.647103e-03
```

- We perfom Bootstrap on $R = 10,000$

```r
N = 1000
boot.coef = boot(data = Default, boot.fn, R = N )
print(boot.coef)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = N)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -3.874290e-02 4.347696e-01
## t2*  2.080898e-05  1.572321e-07 4.864492e-06
## t3*  5.647103e-03  1.834251e-05 2.300607e-04
```

- The standard errors are respectively $4.351099e-01, 4.673198e-06, 2.336489e-04$ for $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$.

Part (d)

The standard errors are estimated to be very close when using bootstrap and method `glm()` function.

Exercise 7: This question involves method Leave-One-Out-Cross-Validation (LOOCV) method.

Part (a)

```r
attach(Weekly)
glm.fit = glm(Direction ~ Lag1 + Lag2,
              data = Weekly,
              family = "binomial")
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.623  -1.261   1.001   1.083   1.506
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22122    0.06147   3.599 0.000319 ***
## Lag1        -0.03872    0.02622  -1.477 0.139672
## Lag2         0.06025    0.02655   2.270 0.023232 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1488.2  on 1086  degrees of freedom
## AIC: 1494.2
##
## Number of Fisher Scoring iterations: 4
```

Part (b)

- We fit a logistic regression model that predicts Predict using Lag1 and Lag2 except for the first observation:

```r
glm.fit = glm(
  Direction ~ Lag1 + Lag2,
              data = Weekly[-1,],
              family = "binomial"
)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly[-1,
##     ])
##
## Deviance Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -1.6258  -1.2617   0.9999   1.0819   1.5071
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22324    0.06150   3.630 0.000283 ***
## Lag1        -0.03843    0.02622  -1.466 0.142683
## Lag2         0.06085    0.02656   2.291 0.021971 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1494.6  on 1087  degrees of freedom
## Residual deviance: 1486.5  on 1085  degrees of freedom
## AIC: 1492.5
##
## Number of Fisher Scoring iterations: 4
```

Part (c)

- We predict the direction of the first observation: $P(Direction = "Up"|Lag1, Lag2) > 0.5$ and check this if the observation correctly classified:

```
glm.pred.First = predict.glm(glm.fit, Weekly[1,], type = "response")

Class.First.Observation = (glm.pred.First > 0.5)

print(list(
  Pred.First.Observation =  glm.pred.First,
  Class.First.Observation = Class.First.Observation)
)
```

```
## $Pred.First.Observation
##         1
## 0.5713923
##
## $Class.First.Observation
##    1
## TRUE
```

- The observation is correctly classified.

Part (d)

- We write a loop from i = 1 to i = n where n is the number of observations in the data set that:
- (i). Fit the logistic regression model using all but except ith observation to predict Direction using Lag1 and Lag2.
- (ii). Compute the posterior probability of the market moving up for the ith observation.
- (iii). Use the posterior probability for the ith observation in order to predict whether or not the market moves up.
- (iv). Determine an error was made in predicting the direction for ith observation. If an error was made then indicate this as a 1, and otherwise indicate it as a 0.

```
### Create the vector to store values of error:
ErrorMade = rep(NA)
```

```r
### length of data set:
n = dim(Weekly)[1]


for(i in 1:n){
  glm.fit = glm(
    Direction ~ Lag1 + Lag2,
    data = Weekly[-i,],
    family = "binomial"
  )
  Predict.Up = predict.glm(glm.fit,
                          newdata = Weekly[i,],
                          type = "response") > 0.5

  True.Data = Weekly[i,]$Direction == "Up"

  if (Predict.Up != True.Data){
    ErrorMade[i] = 1
  }
  else{
    ErrorMade[i] = 0
  }

}
```

Part (e)

- We compute the Test Error Rate:

```r
Test.Error.Rate = mean(ErrorMade)

print(list(
  Test.Error.Rate = Test.Error.Rate
))
```

```
## $Test.Error.Rate
## [1] 0.4499541
```

- The LOOCV test error rate is about 44.9% which seems large, this indicates that logistic regression model predicting Direction using Lag1 and Lag2 is not a good model.

Exercise 8: We perform cross-validation from a simulated data set:

Part (a)

- We generate the simulated data set:

```r
set.seed(1)
x = rnorm(100)
eps = rnorm(100)
y = x - 2*x^2 + eps

DataSet  = data.frame(x,y, eps)
```

- The n value is 100, and p the number of predictors is 2. The model of this is:

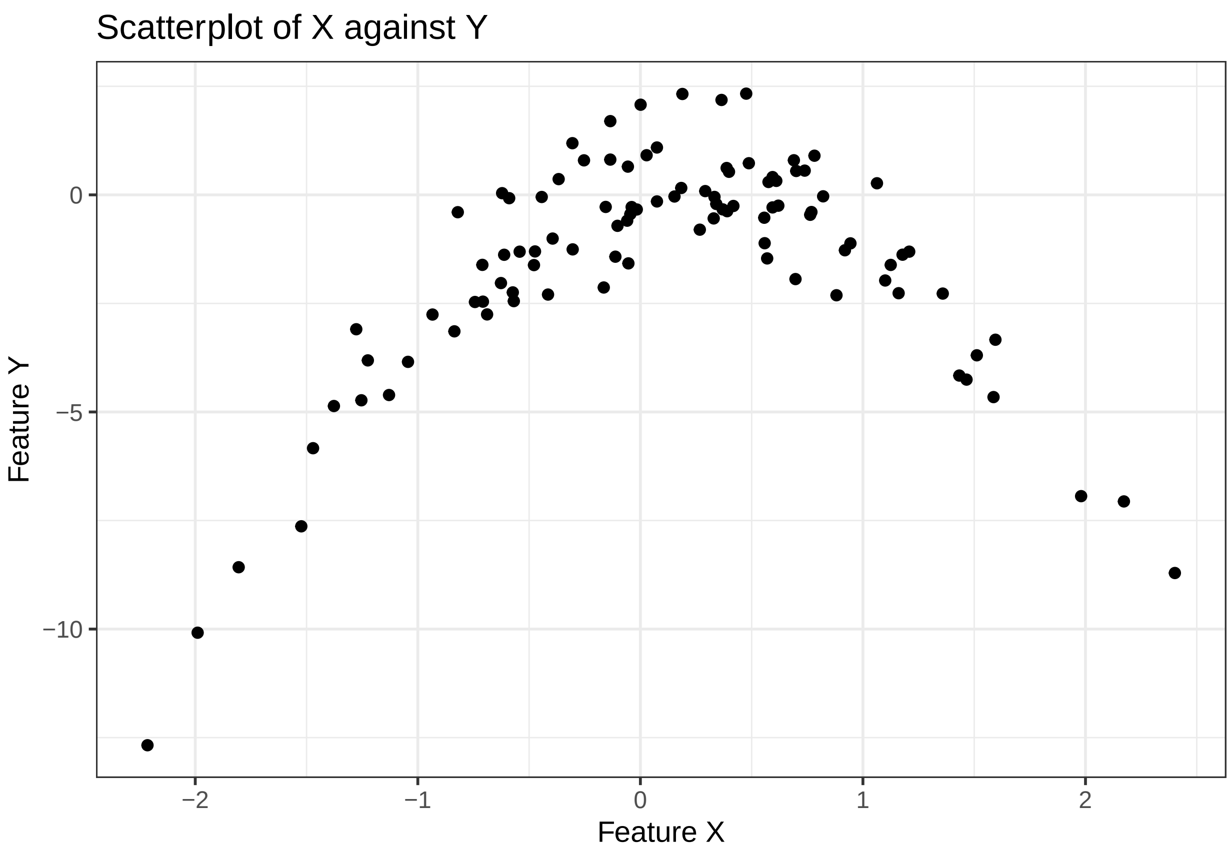
$$Y = X - 2 \times X^2 + \epsilon$$

where $\epsilon \sim N(0, 1)$

Part (b)

- We create the scatterplot of X against Y:

```
Scat.Plot = ggplot(data = DataSet,
                   aes(x = x, y = y)) + geom_point() +
  ggtitle(label = "Scatterplot of X against Y") +
  xlab(label = "Feature X") +
  ylab(label = "Feature Y") +
  theme_bw()

print(Scat.Plot)
```



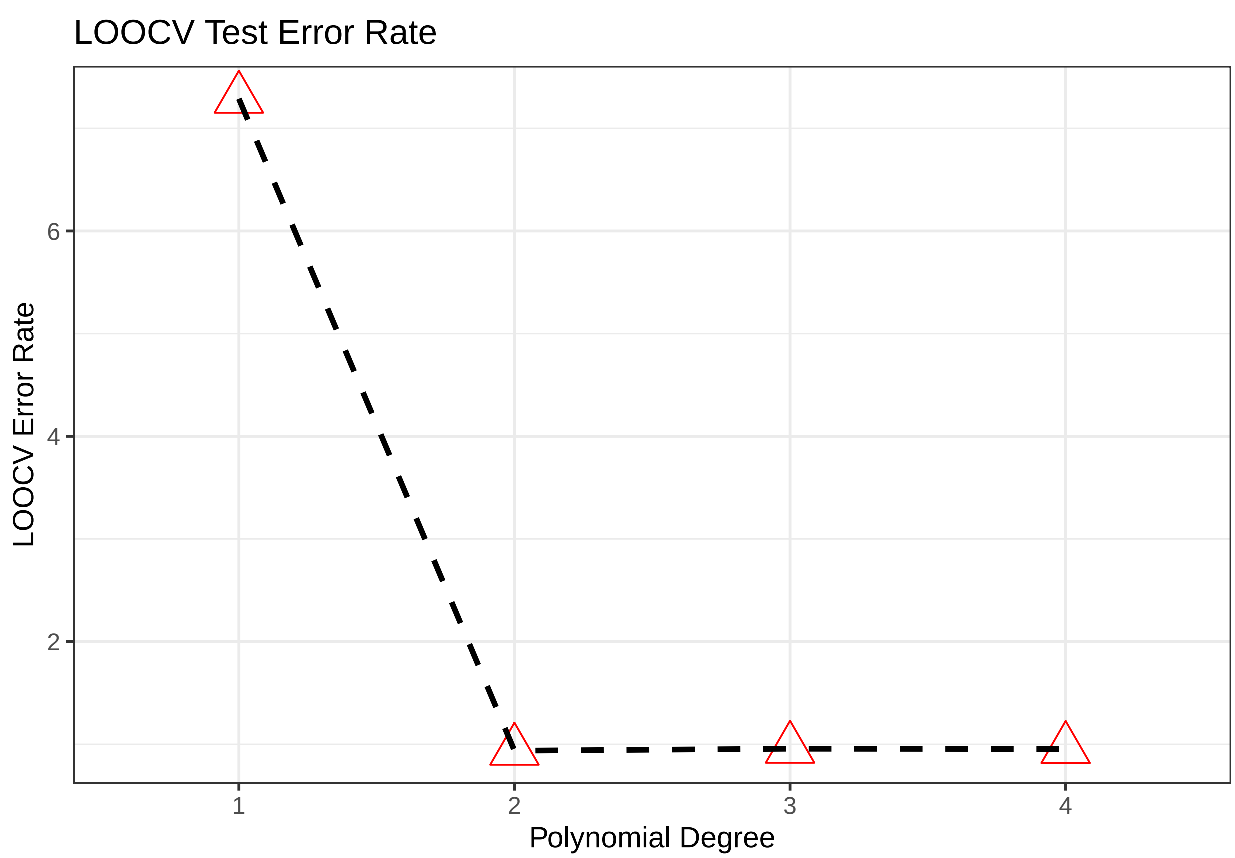
- This is non-linear relationship. The quadratic relationship seems to appear the most described relation for this data between feature X and feature Y.

Part (c)

- We compute the LOOCV errors resulting from fitting four models with polynomial degree from i = 1 to i = 4.

```
LOOCV.error = rep(NA)
deg.poly = c(1,2,3,4)
```

```r
### Perform LOOCV:

set.seed(1)
for (i in deg.poly){
  glm.fit = glm(y ~ poly(x, degree = i), data = DataSet)
  LOOCV.error[i] = cv.glm(data = DataSet, glmfit = glm.fit)$delta[1]
}

DataError.LOOCV = cbind(deg.poly, LOOCV.error)
DataError.LOOCV = as.data.frame(DataError.LOOCV)



Error.Plot = ggplot(data = DataError.LOOCV,
                    aes(x = factor(deg.poly), y = LOOCV.error)) +
  geom_point(pch = 2, color = "red", size = 6) +
  theme_bw() +
  xlab(label = "Polynomial Degree") +
  ylab(label = "LOOCV Error Rate") +
  ggtitle(label = "LOOCV Test Error Rate") +
  geom_path(mapping = aes(x = deg.poly, y = LOOCV.error),
            data = DataError.LOOCV, size = 1, lty = 2)


print(Error.Plot)
```

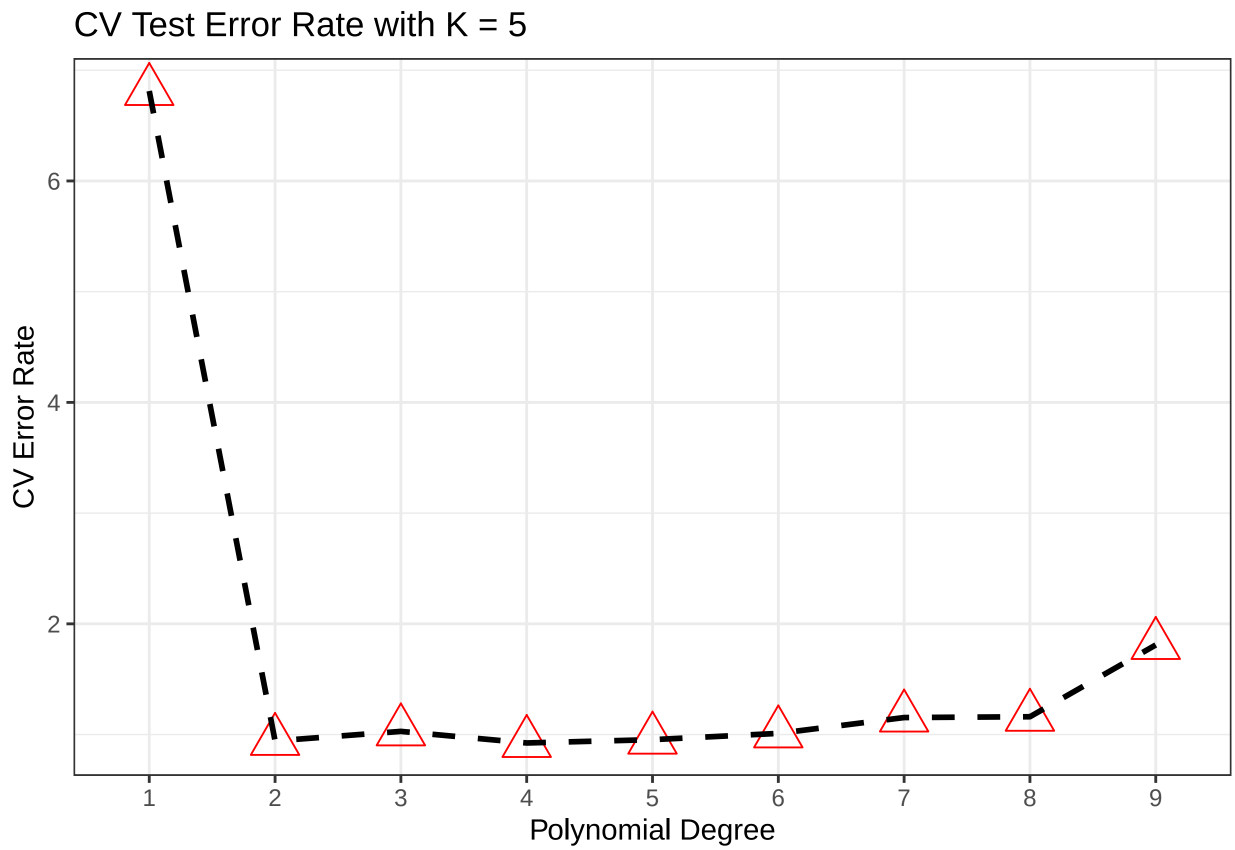


- It seems that highest LOOCV Test Error Rate is highest associated with the degree of polynomial 1 also known as the linear least square. The quadratic least square appears to be the best. There is not much improvement when fitting higher polynimal degree other than degree 2.

Part (d)

- We repeat using Cross-validation:

```
CV.error = rep(NA)
deg.poly = c(1,2,3,4,5,6,7,8,9)


### Perform LOOCV:

set.seed(46617)
for (i in deg.poly){
  glm.fit = glm(y ~ poly(x, degree = i), data = DataSet)
  CV.error[i] = cv.glm(data = DataSet, glmfit = glm.fit, K = 5)$delta[1]
}

DataError.CV = cbind(deg.poly, CV.error)
DataError.CV = as.data.frame(DataError.CV)



Error.Plot = ggplot(data = DataError.CV,
                    aes(x = factor(deg.poly), y = CV.error)) +
  geom_point(pch = 2, color = "red", size = 6) +
  theme_bw() +
  xlab(label = "Polynomial Degree") +
  ylab(label = "CV Error Rate") +
  ggtitle(label = "CV Test Error Rate with K = 5") +
  geom_path(mapping = aes(x = deg.poly, y = CV.error),
            data = DataError.CV, size = 1, lty = 2)


print(Error.Plot)
```



- The same conclusion holds with previous part using LOOCV.

Exercise 9: This question involves the data set `Boston` housing.

Part (a)

- We estimate the population mean of medv. Denote this estimate $\hat{\mu}$

```
attach(Boston)
medv = Boston[,c("medv")]

mu.hat = mean(medv)

print(list(
  estimate.mu.hat = mu.hat
))
```

```
## $estimate.mu.hat
## [1] 22.53281
```

Part (b)

- We compute the estimate of the standard error of $\hat{\mu}$:

```
sample.std = sd(medv)
SE.mu.hat = sample.std/sqrt(length(medv))

print(list(
```

```
  SE.mu.hat = SE.mu.hat
))
```

```
## $SE.mu.hat
## [1] 0.4088611
```

Part (c)

- We estimate the standard error of $\hat{\mu}$ using the boostrap:

```
boot.fn = function(data, index){
  DataSet  = data[index]
  mean.val = mean(DataSet)

  return(mean.val)
}
### We check the function for boostrap:
boot.fn(data = medv, index = sample(100,10))
```

```
## [1] 22.08
```

```
boot.fn(data = medv, index = 1:length(medv))
```

```
## [1] 22.53281
```

```
R.boot = 1000

boot.strap = boot(data = medv, statistic = boot.fn, R = R.boot)

boot.strap
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = medv, statistic = boot.fn, R = R.boot)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 22.53281 -0.02654862   0.4072954
```

- The estimate of SE of $\hat{\mu}$ using boostrap is 0.4078936 which is very close to the estimated SE from part (b).

Part (d)

- The 95% Confidence interval for bootstrap estimate:

```
CI.boot.mu.hat = boot.ci(boot.strap, conf = 0.95)
print(CI.boot.mu.hat)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.strap, conf = 0.95)
##
```

```
## Intervals :
## Level      Normal              Basic
## 95%   (21.76, 23.36 )   (21.76, 23.38 )
##
## Level     Percentile            BCa
## 95%   (21.69, 23.31 )   (21.79, 23.35 )
## Calculations and Intervals on Original Scale
```

- The 95% Confidence Interval for the mean of medv using bootstrap is $(21.73, 23.33)$.
- The 95% Confidence interval for the estimate of mean medv using `t.test()`:

```r
CI.mu.hat = t.test(medv, conf.level = 0.95)
print(CI.mu.hat)
```

```
##
##  One Sample t-test
##
## data:  medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

- The 95% Confidence interval using `t.test()` is $(21.72953, 23.33608)$

Part (e)

- Now we consider the estimate of median for *medv*:

```r
med.hat = median(medv)
print(med.hat)
```

```
## [1] 21.2
```

Part (f)

- We now estimate the standard error for median using bootstrap:

```r
n = length(medv)

boot.fn = function(data, index){
  DataSet = data[index]
  med.val = median(DataSet)

  return(med.val)

}

### Test the function that we write:
boot.fn(data =medv, index = sample(n, 100))
```

```
## [1] 21.4
```

```r
boot.fn(data = medv, index = sample(n,10))
```

```
## [1] 19.8
```

```r
Median.Bootstrap = boot(data = medv, statistic = boot.fn,
                        R = 1000)

print(Median.Bootstrap)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = medv, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     21.2   0.003   0.3686691
```

```r
CI.med.boot = boot.ci(Median.Bootstrap, conf = 0.95)

print(CI.med.boot)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Median.Bootstrap, conf = 0.95)
##
## Intervals :
## Level      Normal              Basic
## 95%   (20.47, 21.92 )   (20.50, 21.85 )
##
## Level     Percentile            BCa
## 95%   (20.55, 21.90 )   (20.44, 21.75 )
## Calculations and Intervals on Original Scale
```

- The standard error of estimated median value using bootstrap is 0.362986. The 95% Confidence interval of $\hat{\mu}_{med}$ is $(20.44, 21.94)$

Part (g)

- We now consider estimating the tenth percentile of medv in the Boston surburbs:

```r
tenth.quant = quantile(medv, probs = 0.10 )
tenth.quant
```

```
##   10%
## 12.75
```

- The value of estimated tenth percentile for `medv` is 12.75

Part (h)

- We now estimate the tenth percentile using bootstrap:

```r
boot.fn =  function(data, index)
{
  DataSet = data[index]
  tenth.quant = quantile(DataSet,
                         probs = 0.10)
```

```r
  return(tenth.quant)
}

### Test the function:
boot.fn(data = medv, index = sample(n, 100))
```

```
##   10%
## 14.08
```

```r
Quant.bootstrap = boot(data = medv, statistic = boot.fn, R = 1000)
print(Quant.bootstrap)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = medv, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75  0.0161   0.4986376
```

```r
boot.ci(Quant.bootstrap, conf = 0.95, type = c("norm"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Quant.bootstrap, conf = 0.95, type = c("norm"))
##
## Intervals :
## Level      Normal
## 95%   (11.76, 13.71 )
## Calculations and Intervals on Original Scale
```

- The standard error of estimated tenth percentile is \$ 0.4980154\$. The 95% confidence interval of bootstrap tenth percentile value is $(11.76, 13.69)$.