Assignment 4 Chap 5

Problem 1–Exercise 3

(1-a) 1. divide the data set into k chunks, 1 for testing data and k-1 for training data 2. apply the model needed to validate, use k-1 chunks for training and 1 chunks for test, and take turns to get k test errors, then average it for the model's test error. 3. use next model (next k or other poly degree), repeat step 2 to get another test error. (1-b) i. The pro of k-fold cv relative to the validation approach is that its test error is much more deterministic whereas the estimate of validation approach is highly variable. In addition, the validation's test error may overestimate the test error. In the other hand, the con of k-fold cv relative to the validation approach is that k-fold cv is much more computationally intensive. ii.The pro of k-fold cv relative to LOOCV is that k-fold cv is less computationally intensive. In addition, k-fold cv have less variance, but however, higher bias than LOOCV.

Problem 2–Exercise 5

(2-a)

```r
library(ISLR)
attach(Default)
set.seed(1)
glm.a=glm(default~income+balance, data=Default, family = binomial)
summary(glm.a)
```

```
##
## 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
```

(2-b)

```r
#i
train = sample(dim(Default)[1], dim(Default)[1]/2)
#ii
glm.b = glm(default~income+balance, data= Default, family = binomial, subset = train)
#iii
glm.b.pred= rep("No", dim(Default)[1]/2)
glm.b.probs= predict(glm.b, Default[-train,], type="response")
glm.b.pred[glm.b.probs>0.5] = "Yes"
#iv
mean(glm.b.pred != Default[-train,]$default)
```

```
## [1] 0.0254
```

(2-c)

```r
#First
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.b = glm(default~income+balance, data= Default, family = binomial, subset = train)
glm.b.pred= rep("No", dim(Default)[1]/2)
glm.b.probs= predict(glm.b, Default[-train,], type="response")
glm.b.pred[glm.b.probs>0.5] = "Yes"
mean(glm.b.pred != Default[-train,]$default)
```

```
## [1] 0.0274
```

```r
#Second
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.b = glm(default~income+balance, data= Default, family = binomial, subset = train)
glm.b.pred= rep("No", dim(Default)[1]/2)
glm.b.probs= predict(glm.b, Default[-train,], type="response")
glm.b.pred[glm.b.probs>0.5] = "Yes"
mean(glm.b.pred != Default[-train,]$default)
```

```
## [1] 0.0244
```

```r
#Third
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.b = glm(default~income+balance, data= Default, family = binomial, subset = train)
glm.b.pred= rep("No", dim(Default)[1]/2)
glm.b.probs= predict(glm.b, Default[-train,], type="response")
glm.b.pred[glm.b.probs>0.5] = "Yes"
mean(glm.b.pred != Default[-train,]$default)
```

```
## [1] 0.0244
```

The validation set error is approximately 2.6%

(2-d)

```
train = sample(dim(Default)[1], dim(Default)[1]/2)
glm.d = glm(default~income+balance+student, data= Default, family = binomial, subset = train)
glm.d.pred= rep("No", dim(Default)[1]/2)
glm.d.probs= predict(glm.d, Default[-train,], type="response")
glm.d.pred[glm.d.probs>0.5] = "Yes"
mean(glm.d.pred != Default[-train,]$default)
```

```
## [1] 0.0278
```

It seems it doesn't matter at all. No sign of reduction in test error

Problem 3–Exercise 6

(3-a)

```
attach(Default)
```

```
## The following objects are masked from Default (pos = 3):
##
##     balance, default, income, student
```

```
set.seed(1)
glm.a=glm(default~income+balance, data=Default, family = binomial)
summary(glm.a)
```

```
##
## 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
```

(3-b)

```
boot.fn = function(data,index)
  return(coef(glm(default~income+balance, data=data, family = binomial, subset = index)))
```

(3-c)

```
library(boot)
boot(Default, boot.fn, 100)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 100)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01  8.556378e-03 4.122015e-01
## t2*  2.080898e-05 -3.993598e-07 4.186088e-06
## t3*  5.647103e-03 -4.116657e-06 2.226242e-04
```

(3-d) Only the sd of intercept have 10% difference, the other two estimate's sd don't have the significant difference.

Problem 4–Exercise 9

(4-a)

```
library(MASS)
attach(Boston)
medv.mean=mean(medv)
medv.mean
```

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

(4-b)

```
medv.error=sd(medv)/sqrt(length(medv))
medv.error
```

```
## [1] 0.4088611
```

(4-c)

```
boot.fu=function(data, index)
  return(mean(data[index]))
boot.9c=boot(medv, boot.fu, 1000)
boot.9c
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = medv, statistic = boot.fu, R = 1000)
##
##
## Bootstrap Statistics :
##     original        bias    std. error
## t1* 22.53281 -0.001814032   0.3916794
```

Pretty similar. 0.41 AND 0.408

(4-d)

```r
t.test(medv)
```

```
##
##  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
```

```r
c(boot.9c$t0 - 2*0.41, boot.9c$t0 + 2*0.41)
```

```
## [1] 21.71281 23.35281
```

t-test interval are narrower than bootstrp confidence interval by 0.36

(4-e)

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

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

(4-f)

```r
boot.fu.median=function(data, index)
  return(median(data[index]))
boot.9f=boot(medv, boot.fu.median, 1000)
boot.9f
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
```

```
##
##
## Call:
## boot(data = medv, statistic = boot.fu.median, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     21.2 -0.0199   0.3751996
```

Median is 21.2, sd is 0.37. The confidence interval of median will be pretty narrow.

(4-g)

```
medv.tenth = quantile(medv, c(0.1))
medv.tenth
```

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

(4-h)

```
boot.fu.tenth=function(data, index)
  return(quantile(data[index], c(0.1)))
boot.9h=boot(medv, boot.fu.tenth, 1000)
boot.9h
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = medv, statistic = boot.fu.tenth, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75  -0.011   0.5287898
```

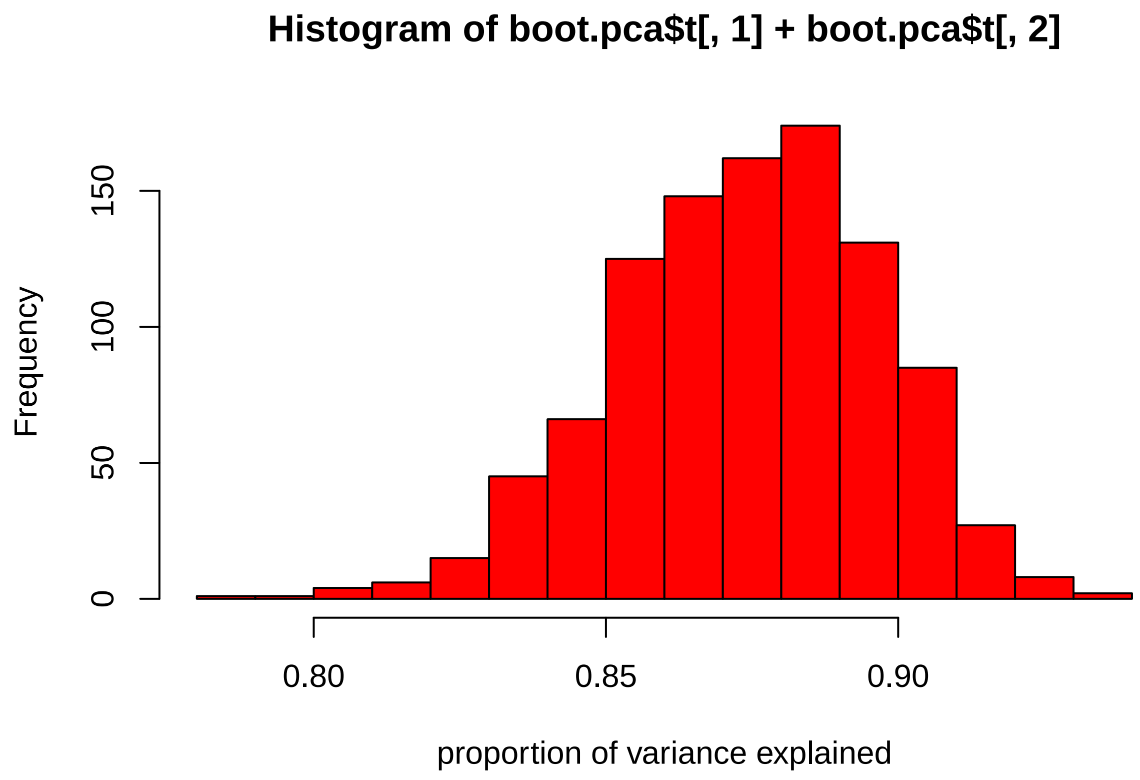
Problem 5

(5-a)

```
attach(USArrests)
pca.arrest=function(data, index){
  pr.out=prcomp(data[index,], scale=TRUE)
  pr.var=pr.out$sdev^2
  pve=pr.var/sum(pr.var)
  return(pve)
}
boot.pca=boot(USArrests, pca.arrest, 1000)
boot.pca
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = USArrests, statistic = pca.arrest, R = 1000)
##
##
## Bootstrap Statistics :
##       original       bias    std. error
## t1* 0.62006039  0.004326650 0.043760993
## t2* 0.24744129  0.002275431 0.032692346
## t3* 0.08914080 -0.003155067 0.017843864
## t4* 0.04335752 -0.003447014 0.009243234
```

```
hist(boot.pca$t[,1]+boot.pca$t[,2], xlab="proportion of variance explained", col=2)
```



(5-b)

```
boot.pca
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
```

```
## Call:
## boot(data = USArrests, statistic = pca.arrest, R = 1000)
##
##
## Bootstrap Statistics :
##       original       bias    std. error
## t1* 0.62006039  0.004326650 0.043760993
## t2* 0.24744129  0.002275431 0.032692346
## t3* 0.08914080 -0.003155067 0.017843864
## t4* 0.04335752 -0.003447014 0.009243234
```

```
first= c(boot.pca$t0[1] - 2*0.046, boot.pca$t0[1] + 2*0.046)
second= c(boot.pca$t0[2] - 2*0.035, boot.pca$t0[2] + 2*0.035)
first
```

```
## [1] 0.5280604 0.7120604
```

```
second
```

```
## [1] 0.1774413 0.3174413
```

PC1 SD=0.046, 95% INTERVAL=0.52~0.71
PC2 SD=0.035, 95% INTERVAL=0.17~0.32

(5-c) Because the loading vector is the eigenvector, and for the eigenvector, it can have two opposite directions, positive or negative. In this way, the avaeage of 1000 vectors will be extremely variable, sd will be extremely large, since each vectors will probabally have different directions.

(5-d)

```
pca.4=function(data, index){
  pca.out=prcomp(data[index,], scale=TRUE)
  PCA.loading=pca.out$rotation[,1]
  x = max(abs(PCA.loading))
  for (i in 1:4) {
    if(x == PCA.loading[i] | x == -PCA.loading[i]){
      x.number=i
    }
  }
  sign.x = sign(PCA.loading[x.number])
  y = PCA.loading*sign.x
  return(y)
}
```

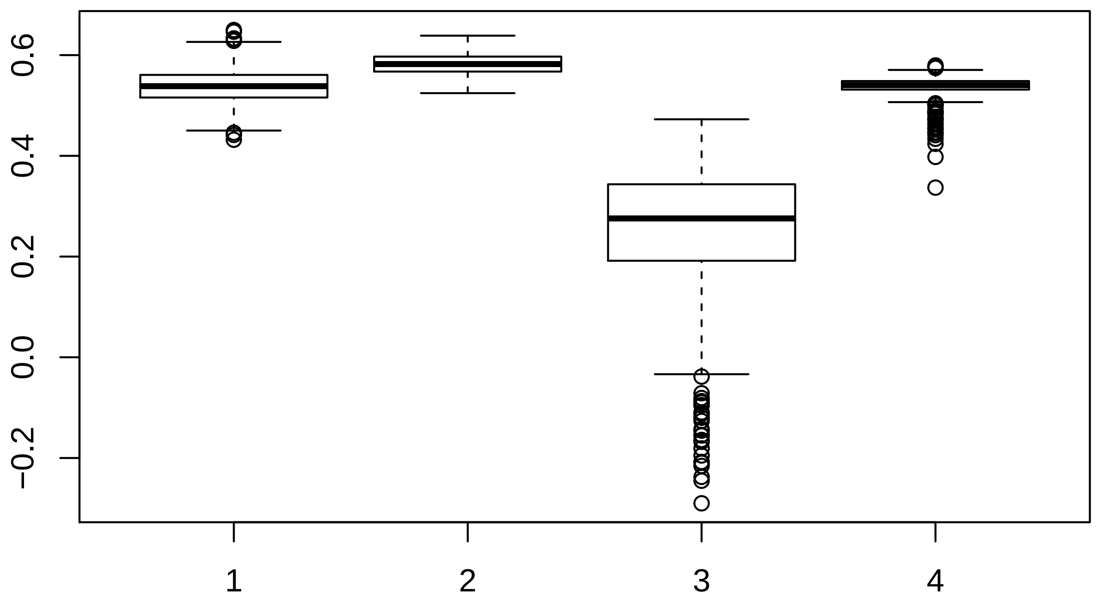
(5-e)

```
pca.boot=boot(USArrests, pca.4, 1000)
pca.boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
```

```
## Call:
## boot(data = USArrests, statistic = pca.4, R = 1000)
##
##
## Bootstrap Statistics :
##      original       bias    std. error
## t1* 0.5358995  0.002380549  0.03297718
## t2* 0.5831836 -0.000580078  0.02110578
## t3* 0.2781909 -0.024947008  0.12224830
## t4* 0.5434321 -0.005010274  0.01834177
```

```r
boxplot(pca.boot$t)
```



(5-f) If we can use signed loading, then we can change the loading vector in the same direction. At first, by sign of the maximum absolute value of the loading, we can realize the eigenvector's direction. And we wanna convert all the different vectors into the same direction, to do this, we multiple the the vector loading by the sign of the maximum absolute value of the loading, to put all the loading vector in a positive direction. And this method will surly become a good (smaller) standard error estimated for the PCA since it eliminates the direction problems of the eigenvector.