HW4

Ramtin Boustani - SUID# 05999261

Problem 1

Chapter 5, Exercise 3

(a)

let's say k=10 then for 10-fold cross-validation we randomly divide the entire observation to 10 sets of size n/10. Then we do 10 times training data with 9 folds and estimate error using remaining one set and averaging the 10 MSE.

```
k = 10
data = data.frame(seq(1:100))
n = nrow(data)

#randomly shuffle data
data = data.frame(data[sample(n,),])

errors = c()
k_counter = k-1
for (i in c(0:k_counter)){

  #training indexes
  current_fold = i*k
  s = 0
  e = current_fold
  training = data[s:e,]
  s = current_fold+k+1
  e = n
  training = c(training, data[s:e,])

  #test indexes
  s = current_fold
  e = current_fold+k
  test = data[s:e,]

  #call external function for training

  #call external function for test (MSE)
  err = -1
  errors = c(errors, err)
}
#averaging MSE
mean(errors)
```

(b)

Validation set approach vs CV
+ validation set approach is computationally simple
+ validation set approach is easy to implement

+ CV is less bias
- error estimate for validation set approach result is highly variable and dependent on set of observations
- validation set approach overestimates the test error because of using less data for training

LOOCV vs CV
+ CV is computationally faster
+ CV error rate estimate is better than LOOCV because of bias-variance trade-off
- LOOCV gives the most unbiased result but has more variance

Problem 2

Chapter 5, Exercise 5

(a)

```
library(ISLR)
attach(Default)
set.seed(1)
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
```

(b)

(i)

Split observations

```
n = dim(Default)[1]
train = sample(n ,n/2, replace = FALSE)
```

(ii)

fit model using only training set

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

(iii)

Predict

```
glm.pred = rep( "No", n/2)
glm.probs = predict(glm.fit, Default[-train,], type = "response")
glm.pred[glm.probs>0.5] = "Yes"

mean(glm.pred != Default[-train,]$default)
```

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

(c)

```
validationSet = function() {
  n = dim(Default)[1]
  train = sample(n ,n/2, replace = FALSE)
  glm.fit = glm(default ~income + balance, data = Default, family = binomial, subset = train)
  glm.pred = rep( "No", n/2)
  glm.probs = predict(glm.fit, Default[-train,], type = "response")
  glm.pred[glm.probs>0.5] = "Yes"
  mean(glm.pred != Default[-train,]$default)
}
result = c()
for(i in 1:5){
  res = validationSet()
  print(res)
  result = c(res, result)
}
```

```
## [1] 0.0274
## [1] 0.0244
## [1] 0.0244
## [1] 0.027
## [1] 0.0262
```

```
range(result)
```

```
## [1] 0.0244 0.0274
```

As expected variability in response

(d)

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

```
## [1] 0.0284
```

Close to the test errors without student, not much change

Problem 3

Chapter 5, Exercise 6

(a)

```
set.seed(1)
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
```

(b)

Computing statistics

```
boot.fn = function(data, index){
 glm.fit = glm( default ~ income + balance, data = data, family = binomial, subset = index )
 coefficients(glm.fit)
}
```

(c)

Use bootstrap function

```
library(boot)
set.seed(1)
```

```
boot.out = boot(Default, boot.fn, R=1000)
boot.out
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -3.945460e-02 4.344722e-01
## t2*  2.080898e-05  1.680317e-07 4.866284e-06
## t3*  5.647103e-03  1.855765e-05 2.298949e-04
```

(d)

glm SE(home) = 4.985e-06
bootstrap SE(home) = 4.866284e-06

```
4.985e-06 - 4.866284e-06
```

```
## [1] 1.18716e-07
```

glm SE(balance) = 2.274e-04
bootstrap SE(balance) = 2.298949e-04

```
2.274e-04 - 2.298949e-04
```

```
## [1] -2.4949e-06
```

very good estimate for standard error

Problem 4

Chapter 5, Exercise 9

```
library(MASS)
attach(Boston)
n = nrow(Boston)
```

(a)

```
crim.mean = mean(crim)
crim.mean
```

```
## [1] 3.613524
```

(b)

$$SE\ of\ the\ mean = \frac{Standard Deviation}{\sqrt{(n)}}$$

```
crim.error = sd(crim) / sqrt(n)
crim.error
```

```
## [1] 0.3823853
```

(c)

```
library(boot)
set.seed(1)
boot.fn = function(data, index){
  mean(data[index])
}
boot.out = boot(crim, boot.fn, R=1000)
boot.out
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = crim, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 3.613524 -0.01246001   0.3709464
```

Very close estimate between bootstrap and computing formula

```
crim.error -  0.3709464
```

```
## [1] 0.01143892
```

(d)

```
c(boot.out$t0 - (2*0.3709464), boot.out$t0 + (2 * 0.3709464))
```

```
## [1] 2.871631 4.355416
```

```
t.test(Boston$crim)
```

```
##
##  One Sample t-test
##
## data:  Boston$crim
## t = 9.45, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  2.862262 4.364786
## sample estimates:
## mean of x
##  3.613524
```

The t.test mean exist in the 95% confidence interval and bootstrap estimate is 0.01 different

(e)

```
crim.median = median(crim)
crim.median
```

```
## [1] 0.25651
```

(f)

```
set.seed(1)
boot.fn = function(data, index){
  median(data[index])
}
boot.out = boot(crim, boot.fn, R=1000)
boot.out
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = crim, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1*  0.25651 0.00294314  0.03580408
```

estimated median is the same as population median with small SE

(g)

```
crim.quantile = quantile(crim, probs=c(0.1))
crim.quantile
```

```
##      10%
## 0.038195
```

(h)

```
set.seed(1)
boot.fn = function(data, index){
  quantile(data[index], probs = c(0.1))
}
boot.out = boot(crim, boot.fn, R=1000)
boot.out
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = crim, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 0.038195 0.000712285 0.003123738
```

Small SE and the same tenth percentile

Problem 5

```
library(boot)
library(ISLR)
attach(USArrests)
```

(1)

```
set.seed(1)
boot.fn = function(data, index){
  pr.out = prcomp(data[index,], scale = TRUE)
  return (pr.out$rotation)
}
boot.out = boot(USArrests, boot.fn, R=1000)
print("PCAs in 1000 Bootstrap of the data")
```

```
## [1] "PCAs in 1000 Bootstrap of the data"
```

```
boot.out$t0
```

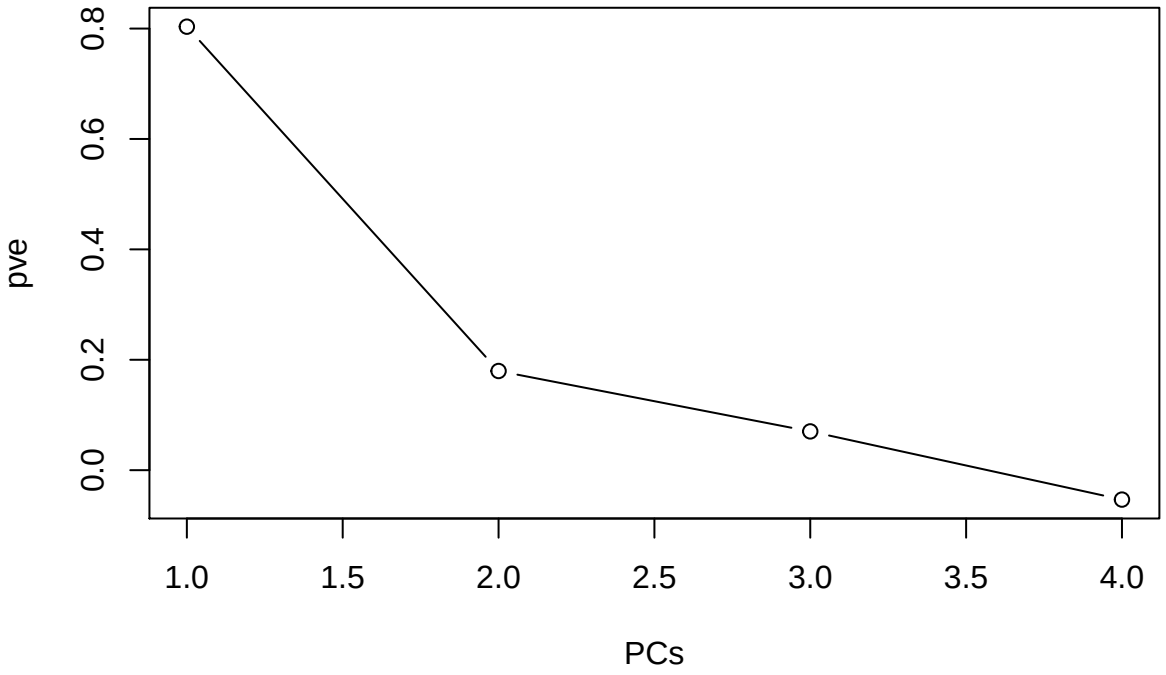
```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

Proportion of variance explained by PC1 and PC2

```
var = apply(boot.out$t0, 2, sum)
pve = var / sum(var)
pve
```

```
##         PC1         PC2         PC3         PC4
##  0.80341834  0.17965116  0.07021931 -0.05328881
```

```
plot(pve, xlab = "PCs" , type = "b")
```



(2)

95% confidence interval for first principal component

```
set.seed(1)
boot.fn = function(data, index){
  pr.out = prcomp(data[index,], scale = TRUE)
  return (abs(pr.out$rotation[,1]))
}
boot.out = boot(USArrests, boot.fn, R=1000)
boot.ci(boot.out, type = c("bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.out, type = c("bca"))
##
## Intervals :
## Level       BCa
## 95%   ( 0.4787,  0.6023 )
## Calculations and Intervals on Original Scale
```

95% confidence interval for second principal component

```
set.seed(1)
boot.fn = function(data, index){
  pr.out = prcomp(data[index,], scale = TRUE)
  return (abs(pr.out$rotation[,2]))
}
boot.out = boot(USArrests, boot.fn, R=1000)
boot.ci(boot.out, type = c("bca"))
```

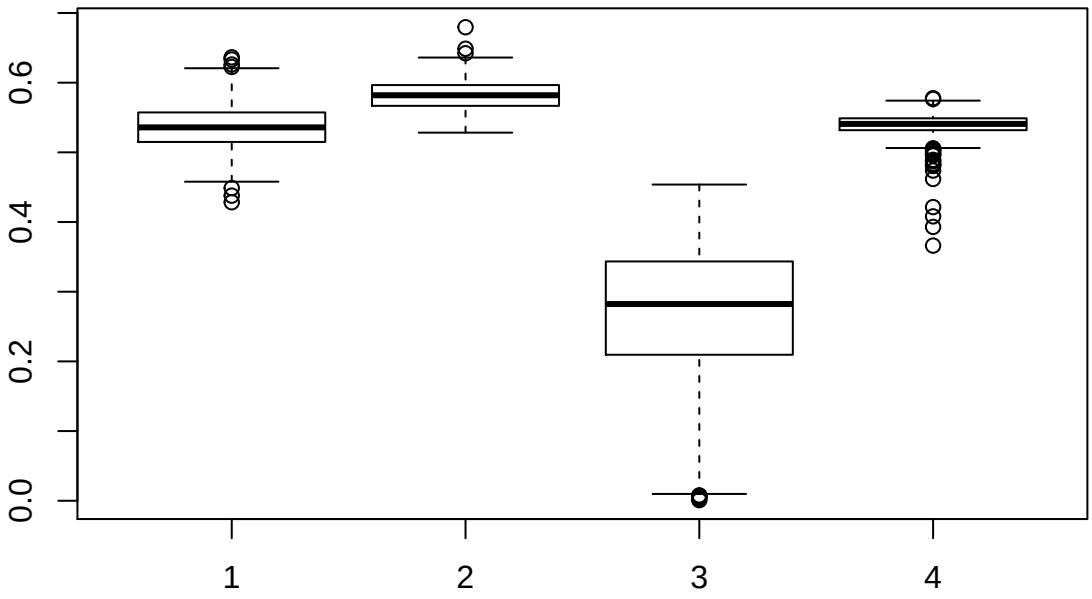
```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
```

```
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.out, type = c("bca"))
##
## Intervals :
## Level       BCa
## 95%   ( 0.1874,  0.6067 )
## Calculations and Intervals on Original Scale
```

(3)

The sign (+/-) of PCs are not important but causes the wrong result while averaging their values and endup with wrong range for standard error and confidence interval.
The following boxplot shows the problem:

```
boot.fn = function(data, index) {
  pr.out = prcomp(data[index,], scale = TRUE )
  abs(pr.out$rotation[,1])
}
boot.out = boot(USArrests, boot.fn, R=1000)
boxplot(boot.out$t,use.cols = TRUE)
```

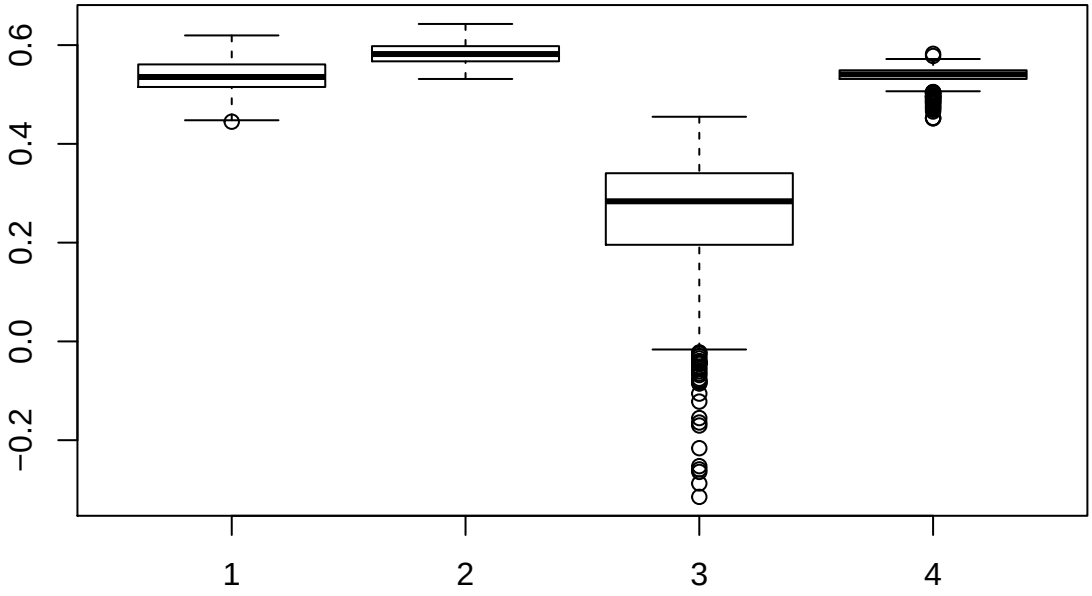


(4)

```
boot.fn = function(data, index) {
  pr.out = prcomp(data[index,], scale = TRUE )
  pc1 = pr.out$rotation[,1]
  pos = which(abs(pc1) == max(abs(pc1)))
  pos.sign = sign(pc1[pos])
  pc1 * pos.sign
}
```

(5)

```
boot.out = boot(USArrests, boot.fn, R=1000)
boxplot(boot.out$t,use.cols = TRUE)
```



(6)

The predicator that has the most variance won't change in all bootstrap samples so the function in part 4 tries to fix the sign issue for that predicator that has the highest scores in all samples.
Also we are trying to solve the problem only for the first principal component so that solution cannot be used to simultaneously fix other PCs sign issue.