Assignment 2

Name: Kalpita Dapkekar

1St Theory:

1. Write a short paragraph about the bootstrap, how does the algorithm work and how the SE of the estimate is computed. Also explain how parametric bootstrap works. Explain pros and cons of using the bootstrap.

Bootstrap is a form of sampling from the data, which tries to 'capture' features in the distribution which the over simplified normal approximation cannot do. The bootstrap is a tool, which allows us to obtain better finite sample approximation of estimators. The bootstrap is used all over the place to estimate the variance, correct bias and construct CIs etc. There are many, many different types of bootstraps. e two simple versions of the bootstrap for constructing CIs. They can be roughly described as the nonparametric bootstrap and the parametric bootstrap.

SE

Let θ^ denote the estimate of our parameter, from the original sample.

Then let $\hat{\theta}_b$,b = 1,…., B denote the B estimates of θ from the bootstrap samples.
The bootstrap standard error for θ^ is then given by

$$SE(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\theta b - \theta)^2}$$

where θ denotes the mean of the estimates across the B bootstrap samples

Parameter bootstrap:

- For each bootstrap sample construct the bootstrap MLE $\hat{\theta}^*_{T,1}$.
  $X^*_{T,1} = (X^*_{1,1}, \ldots, X^*_{T,1})$
- Construct an estimate of the finite sample distribution using a large number of draws.
  $\hat{G}_T(x) = 1/n \sum_{k=0}^{n} I(\sqrt{T}\,\hat{\theta}^*_{T,k} - \hat{\theta}_T) \le x)$.
- Let $\hat{\xi}\alpha$ be such that $\hat{G}_T(\hat{\xi}_\alpha) = \alpha$. The 95% CI bootstrap CI of the mean θ_0 is
  $[\hat{\theta}_T + 1/\sqrt{T}\,\hat{\xi}_{\alpha/2}, \hat{\theta}_T + 1/\sqrt{T}\,\hat{\xi}_{1-\alpha/2}]$

Pros and cons:

- Pros:
  1. Parametric: Samples are drawn from finite distribution.

2. Maximum likelihood estimators are commonly used for parametric bootstrapping despite the fact that this criterion is nearly always based upon their large sample behaviour

- Cons:
  1. Non parametric: Samples are drawn from infinite samples.
  2. nonparametric bootstrap samples underestimate the amount of variation in the population you originally sampled.

#2nd Simulation

Let's compare three nonparametric methods for constructing confidence intervals for the variance of a random variable: a) the functional delta method, b) the bootstrap percentile interval, c) and the BCa interval. Conduct a simulation study to determine how the coverage probability and average interval width of these two intervals varies with the sample size n. For each of the distributions below, produce a plot of coverage probability versus sample size, with lines representing the various methods, as well as a corresponding plot for interval width.

a Carry out the above simulation with data generated from the standard normal distribution.
b Repeat using data generated from an exponential distribution with rate 1.
c Briefly, comment on the strengths and weaknesses of these three methods.

(a) Code

#Part a - Delta method

```
X1 <- 30

X2 <- 40


n1 <- 50

n2 <- 50
```

```
p1.hat <- X1/n1

p2.hat <- X2/n2


t.hat <- p2.hat - p1.hat    #tau = p2 - p1


se.hat <- sqrt((p1.hat*(1-p1.hat)/n1) + (p2.hat*(1-p2.hat)/n2))

se.hat     #the estimated standard error of t by the delta method


#90% conf interval

lower.conf <- 0.2 - 1.645*se.hat

upper.conf <- 0.2 + 1.645*se.hat


conf.intvl <- c(lower.conf,upper.conf)

conf.intvl   #90% confidence interval


#Part b - Bootstrap method


B <- 10000

tau.boot <- rep(0,B)

for (i in 1:B) {

 xx1 <- rbinom(1,n1,p1.hat)

 xx2 <- rbinom(1,n2,p2.hat)

 tau.boot[i] <- (xx2/n2) - (xx1/n1)

}
```

```
se.hat.bt <- sd(tau.boot)
#90% conf interval
lower.conf.bt <- 0.2 - 1.645*se.hat.bt
upper.conf.bt <- 0.2 + 1.645*se.hat.bt

conf.intvl.bt <- c(lower.conf.bt,upper.conf.bt)
conf.intvl.bt   #90% confidence interval
```

```
[1] 0.08944272
[1] 0.05286673 0.34713327
[1] 0.05481946 0.34518054
```

```
library(boot)
n <- 100
x <- rnorm(n)
foo <- sort(x)
my.mean = function(x,indices) {
  return( mean(x[indices]) )
}
boot.out = boot(foo,my.mean,1000)
boot.ci(boot.out)

data <- foo
```

```
mean(data)

library(boot)

my.mean = function(x,indices) {

  return( mean(x[indices]) )

}

boot.out = boot(foo,my.mean,1000)

boot.ci(boot.out)


75+2*sd(boot.out$t)

c(mean(data)-1.96*4.756461,mean(data)+1.96*4.756461)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates


CALL :

boot.ci(boot.out = boot.out)


Intervals :

Level      Normal              Basic

95%   (-0.0846,  0.3752 )   (-0.0921,  0.3684 )


Level     Percentile            BCa

95%   (-0.0749,  0.3856 )   (-0.0762,  0.3774 )

Calculations and Intervals on Original Scale

Warning message:
```

```
In boot.ci(boot.out) : bootstrap variances needed for studentized
intervals

[1] 0.1467636

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates


CALL :

boot.ci(boot.out = boot.out)


Intervals :

Level      Normal              Basic

95%   (-0.0806,  0.3666 )   (-0.0818,  0.3648 )


Level     Percentile            BCa

95%   (-0.0713,  0.3754 )   (-0.0666,  0.3765 )

Calculations and Intervals on Original Scale

Warning message:

In boot.ci(boot.out) : bootstrap variances needed for studentized
intervals

[1] 75.22815

[1] -9.175900  9.469427
```

(b) Code

```
library(boot)

n <- 100

x <- rexp(n)
```

```
foo <- sort(x)
my.mean = function(x,indices) {
  return( mean(x[indices]) )
}
boot.out = boot(foo,my.mean,1000)
boot.ci(boot.out)

data <- foo
mean(data)
library(boot)
my.mean = function(x,indices) {
  return( mean(x[indices]) )
}
boot.out = boot(foo,my.mean,1000)
boot.ci(boot.out)

75+2*sd(boot.out$t)
c(mean(data)-1.96*4.756461,mean(data)+1.96*4.756461)
```

Output:

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.out)
```

```
Intervals :

Level      Normal              Basic

95%   ( 0.6435,  0.9513 )   ( 0.6266,  0.9334 )


Level     Percentile            BCa

95%   ( 0.6591,  0.9659 )   ( 0.6738,  1.0044 )

Calculations and Intervals on Original Scale

Warning message:

In boot.ci(boot.out) : bootstrap variances needed for studentized
intervals

[1] 0.7962691

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates


CALL :

boot.ci(boot.out = boot.out)


Intervals :

Level      Normal              Basic

95%   ( 0.6515,  0.9489 )   ( 0.6473,  0.9495 )


Level     Percentile            BCa

95%   ( 0.6430,  0.9452 )   ( 0.6684,  0.9890 )

Calculations and Intervals on Original Scale
```

```
Some BCa intervals may be unstable

Warning message:

In boot.ci(boot.out) : bootstrap variances needed for studentized
intervals

[1] 75.15176

[1] -8.526395 10.118933
```

Suggestion is that bootstrap confidence intervals should be used whenever there is cause to doubt the assumptions underlying parametric confidence intervals. They will either validate such assumptions, or avoid misleading inferences being drawn.

The main advantage to the BCa interval is that it corrects for bias and skewness in the distribution of bootstrap estimates. The BCa interval requires that you estimate two parameters. The bias-correction parameter, z_0, is related to the proportion of bootstrap estimates that are less than the observed statistic. The acceleration parameter, *a*, is proportional to the skewness of the bootstrap distribution.

#3. From Wasserman's "All of Nonparametric Statistics". Suppose that 50 people are given a placebo and 50 are given a new treatment. Thirty placebo patients show improvement, while 40 treated patients show improvement. Let τ = p2 − p1 where p2 is the probability of improving under treatment and p1 is the probability of improvingunder placebo.

a Find the mle of τ . Find the standard error and 90 percent confidence interval using the delta method.

b Find the standard error and 90 percent confidence interval using the bootstrap.

```
#Fit a normal distribution

N <- 50

x <- rnorm(N, mean = 3, sd = 2)

mean(x)

sd(x)
```

```
#Formulate log-likelihood function
LL <- function(mu, sigma) {
  R = suppressWarnings(dnorm(x, mu, sigma))
  #
  -sum(log(R))
}

#Apply MLE to estimate 2 parameters (Mean and standard deviation)
library(stats4)
mle(LL, start=list(mu=1, sigma=1))

#CALL:
mle(minuslogl = LL, start = list(mu=1, sigma=1))
#outputs of mu and sigma follow

##--Find standard error (sample)
#standard error is standard deviation divided by square root of sample size
se <- sigma/sqrt(N)

##--Find 90% CI using delta method
delta.method(object, 0.2, var=0.90, paramterPrefix="b")

##--Find standard error (sample) & Find 90% CI using boostrap
install.packages("boot")
```

```
rsq <- function(formula, data, indices) {
  d <- data[indices,]      #allows boot to select sample
  fit <- lm(formula, data=d)
  return(summary(fit)$r.square)
}

#boostrapping with 1000 replications
results <- boot(data=, statistics=rsq,
              R = 1000, formula=mpg~wt+disp)

#View Results
results
plot(results)

#Get 90% CI
boot.ci(results, type="bca")
```

Output:

```
[1] 2.458878
[1] 1.886976


Call:
mle(minuslogl = LL, start = list(mu = 1, sigma = 1))


Coefficients:
```

```
      mu    sigma
2.458878 1.868010


Call:
mle(minuslogl = LL, start = list(mu = 1, sigma = 1))


Coefficients:
      mu    sigma
2.458878 1.868010
```

#3rd Implementation

#4th Question

```
jackknife = function (x, theta, ...)
{
  call = match.call()
  n = length(x)
  u = rep(0, n)
  for (i in 1:n) {
    u[i] = theta(x[-i], ...)
  }
  theta.hat = theta(x, ...)
  pseudo.values = n*theta.hat - (n-1)*u
  theta.jack = mean(pseudo.values)
  jack.se = sqrt(sum((pseudo.values - theta.jack)^2)/(n*(n-1)))
  jack.bias = (n-1)*(theta.hat - mean(u))
```

```
  return(list(theta.hat = theta.hat,
              theta.jack = theta.jack,
              jack.bias = jack.bias,
              jack.se = jack.se,
              leave.one.out.estimates = u,
              pseudo.values = pseudo.values,
              call = call))
}
```

#4th Application

#5th Question

```
jackknife = function (x, theta, ...)
{
  call = match.call()
  n = length(x)
  u = rep(0, n)
  for (i in 1:n) {
    u[i] = theta(x[-i], ...)
  }
  theta.hat = theta(x, ...)
  pseudo.values = n*theta.hat - (n-1)*u
  theta.jack = mean(pseudo.values)
  jack.se = sqrt(sum((pseudo.values - theta.jack)^2)/(n*(n-1)))
  jack.bias = (n-1)*(theta.hat - mean(u))
  return(list(theta.hat = theta.hat,
```

```
        theta.jack = theta.jack,
        jack.bias = jack.bias,
        jack.se = jack.se,
        leave.one.out.estimates = u,
        pseudo.values = pseudo.values,
        call = call))
}

library(bootstrap)
n=15;
yzdata <- as.matrix(c(576,3.39,580,3.07,653,3.12,
                635,3.30,555,3.00,575,2.74,558,2.81,661,3.43,545,2.76,
                578,3.03,651,3.36,572,2.88,666,3.44,605,3.13,594,2.96))
dim(yzdata) <- c(2,n)
indata <- t(yzdata)
corr <- function(yz,indata) { cor(indata[yz,1],indata[yz,2]) }
sampcorr <- cor(indata[1:n,1],indata[1:n,2])
sampcorr
jacklaw <- jackknife(1:n,corr,indata)
jacklaw
corrjack = sampcorr - jacklaw$jack.bias
corrjack
```

Output:

```
Attaching package: 'bootstrap'
```

```
The following object is masked _by_ '.GlobalEnv':

    jackknife

[1] 0.7763745
$theta.hat
[1] 0.7763745

$theta.jack
[1] 0.7828481

$jack.bias
[1] 0.006473623

$jack.se
[1] 0.1425186

$leave.one.out.estimates
 [1] 0.8929471 0.7799687 0.8181007 0.7637068 0.7845360 0.7857184
0.7549984
 [8] 0.7361618 0.7403509 0.7760968 0.7517391 0.7670413 0.7313197
0.7761231
[15] 0.7798725

$pseudo.values
```

```
 [1] -0.8556427  0.7260560  0.1922075  0.9537216  0.6621137
0.6455592

 [7]  1.0756402  1.3393518  1.2807048  0.7802626  1.1212703
0.9070386

[13]  1.4071420  0.7798940  0.7274020



$call

jackknife(x = 1:n, theta = corr, indata)



[1] 0.7699009
```

#4th Applications

#6th Question

```
#install.packages("bootstrap")

library(bootstrap)

#help(bootstrap)

theta <- function(i) {

 values <- eigen(var(scor[i,]), symmetric=TRUE, only.values=TRUE)$values

 values[1] / sum(values) }


scor_bootstrap <- bootstrap(1:88, 500, theta)


sd(scor_bootstrap$thetastar) # bootstrap standard error


hist(scor_bootstrap$thetastar)

abline(v=theta(1:88), col="red2")
```

abline(v=mean(scor_bootstrap$thetastar), col="blue")

For principal components analysis svd is better numerically than

eigen-decomposistion, but for bootstrapping eigen-decomposition is much faster.

Output:

```
[1] 0.04742318
```