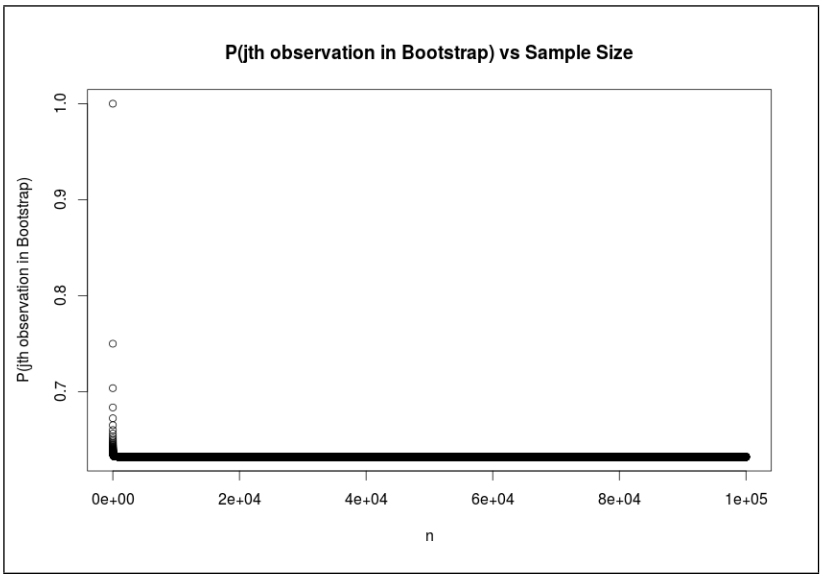
Patrick Rogan
UNI: psr2125

STAT4702 Homework 10

5.4.2

g. The probability that the jth observation will be included in the bootstrap sample is 0.6321224 for n = 100000. Increasing n by several magnitudes of ten we get 0.6321206, which appears to be converging to $\frac{e-1}{e}$. No matter how large our sample size, there will always be at least $\approx 63\%$ chance the jth observation will be in the bootstrap sample.



h. The value obtained is 0.6353. Numerically for j = 4 and n = 100, the answer agrees with what we calculated in g.

5.4.6

a. Given the logistic regression model with $\hat{\beta_1}$ corresponding to income and $\hat{\beta_2}$ corresponding to balance:
$SE(\hat{\beta_1}) = 4.985e-06$
$SE(\hat{\beta_2}) = 2.274e-04$

b. See supporting code.

c. $SE(\hat{\beta_1}) = 4.583e-06$
$SE(\hat{\beta_2}) = 2.268e-04$

```
Call:
boot(data = Default, statistic = boot.fn, R = 1000)

Bootstrap Statistics :
         original        bias     std. error
t1* -1.154047e+01 -8.008379e-03 4.239273e-01
t2*  2.080898e-05  5.870933e-08 4.582525e-06
t3*  5.647103e-03  2.299970e-06 2.267955e-04
```

d. Using a bootstrap with 1000 resamples, the values for standard error obtained are fairly close (roughly within 10% of the values produced by summary) indicating the model provides a good fit to the data.

5.4.9

a. $\hat{\mu} = 22.533$

b. $SE(\hat{\mu}) = 0.409$, given this value is fairly small with respect to the estimate of the population mean, we are fairly confident the population mean is close to the estimate provided in a.

c. Using a bootstrap with 1000 resamples $SE(\hat{\mu}) = 0.412$, this is fairly close to the value obtained in part b.

d. [21.82505, 23.45143]

e. $\hat{\mu}_{medv} = 21.2$

f. The approximate standard error of the median of medv estimated using a bootstrap with 1000 resamples is 0.380. As with the standard error estimate of the mean, this value is fairly small with respect to the estimate of the median allowing us to be fairly confident the population median is fairly close to 21.2.

g. $\hat{\mu}_{0.1} = 12.75$

h. The approximate standard error of the lower 10^{th} percentile is 0.505. Again, we have what appears to be a high degree of confidence that the population value is fairly close to our estimate.

```
#############################################################################################
#################################### Supporting Code ########################################
#############################################################################################
# STAT4702 Homework 10
# Patrick Rogan
# UNI: psr2125

# 5.4.2 ------------------------------------------------------------------
n = 10000
1-(1-1/n)^n

# G
n = 1:100000
pNinBS = 1-(1-1/n)^n

plot(n,pNinBS, xlab = 'n', ylab = "P(jth observation in Bootstrap)",
     main = "P(jth observation in Bootstrap) vs Sample Size")

# H
store = rep(NA, 10000)
for (i in 1:10000){
  store [i]= sum (sample(1:100, rep = TRUE)==4) > 0
}
mean(store)

# 5.4.6 ------------------------------------------------------------------
library(ISLR)
library(boot)
attach(Default)

# Part A
glm.fit = glm(default ~ income + balance, family = binomial, data = Default)
summary(glm.fit)

# Part B
boot.fn <- function(Default,index){
  glm.fit = glm(default ~ income + balance, family = binomial,
                data = Default, subset = index)
  return(coefficients(glm.fit))
}

# Part C
set.seed(1)
boot(Default,boot.fn,1000)

# 5.4.9 ------------------------------------------------------------------
```

```
library(MASS)
attach(Boston)

# Part A
mu_hat = mean(medv)

# Part B
SE_mu_hat = sqrt(var(medv))/sqrt(length(medv))

# Part C
boot.fn <- function(Boston,index){
  meanMedv = mean(medv[index])
  return(meanMedv)
}

set.seed(1)
result = boot(Boston,boot.fn,1000)

# Part D
ci = cat(boot.ci(result, type="bca")[[4]][4], boot.ci(result, type="bca")[[4]][5])

# Part E
median(medv)

# Part F
boot.fn <- function(Boston,index){
  seMed = median(medv[index])
  return(seMed)
}

set.seed(1)
result = boot(Boston, boot.fn,1000)

# Part G
unname(quantile(medv,0.1))

# Part H
boot.fn <- function(Boston,index){
  seMed = unname(quantile(medv[index],0.1))
  return(seMed)
}

set.seed(1)
result = boot(Boston, boot.fn,1000)
```

```
# STAT4702 Homework 10
# Patrick Rogan
# UNI: psr2125
# Note that this code is derived from work presented by Robert V. Hogg, Joseph W. McKean and
# Allen T. Craig in Introduction to Mathematical Statistics, 7th edition on pages 651 and
# 652.

# 4.9.3 ----------------------------------------------------------------
percentciboot<-function(x,b,alpha){
  # x is a vector containing the original sample.
  # b is the desired number of bootstraps.
  # alpha: (1 - alpha) is the confidence coefficient.
  #
  # theta is the point estimate.
  # lower is the lower end of the percentile confidence interval.
  # upper is the upper end of the percentile confidence interval.
  # thetastar is the vector of bootstrapped theta^*s.
  #
  theta<-median(x)
  thetastar<-rep(0,b)
  n<-length(x)
  for(i in 1:b){xstar<-sample(x,n,replace=T)
    thetastar[i]<-median(xstar)
  }
  thetastar<-sort(thetastar)
  pick<-round((alpha/2)*(b+1))
  lower<-thetastar[pick]
  upper<-thetastar[b-pick+1]
  return(list(theta=theta,lower=lower,upper=upper))#,thetastar=thetastar))
  #list(theta=theta,lower=lower,upper=upper)
}
x = c(131.7, 4.3, 182.7, 265.6, 73.3, 61.9, 10.7, 10.8, 150.4,
      48.8, 42.3, 22.5, 22.2, 8.8, 17.9, 150.6, 264.0, 103.0,
      154.4, 85.9)

percentciboot(x,3000,0.1)

# 4.9.5 ----------------------------------------------------------------
percentcibootstand<-function(x,b,alpha){
  # x is a vector containing the original sample.
  # b is the desired number of bootstraps.
  # alpha: (1 - alpha) is the confidence coefficient.
  #
  # theta is the point estimate.
  # lower is the lower end of the percentile confidence interval.
  # upper is the upper end of the percentile confidence interval.
  # thetastar is the vector of bootstrapped theta^*s.
  #
  theta<-mean(x)
  sdx = sd(x)
  thetastar<-rep(0,b)
  n<-length(x)
  for(i in 1:b){xstar<-sample(x,n,replace=T)
    bootMean = mean(xstar)
    sdBoot = sd(xstar)
    thetastar[i]<-(bootMean - theta)/(sdBoot/sqrt(n))
  }
  thetastar<-sort(thetastar)
  pick<-round((alpha/2)*(b+1))
  lower<-thetastar[pick]
  upper<-thetastar[b-pick+1]
```

```
  lower2 = theta - upper*sdx/sqrt(n)
  upper2 = theta - lower*sdx/sqrt(n)
  return(list(theta=theta,lower=lower2,upper=upper2))#,thetastar=thetastar))
  #list(theta=theta,lower=lower,upper=upper)
}

x = c(119.7, 95.4, 104.1, 77.2, 92.8, 100.0, 85.4, 114.2, 108.6, 150.3, 93.4,
      102.3, 67.1, 105.8, 88.4, 107.5, 101.0, 0.9, 97.2, 94.1)

percentcibootstand(x, 3000, 0.1)

percentciboot<-function(x,b,alpha){
  # x is a vector containing the original sample.
  # b is the desired number of bootstraps.
  # alpha: (1 - alpha) is the confidence coefficient.
  #
  # theta is the point estimate.
  # lower is the lower end of the percentile confidence interval.
  # upper is the upper end of the percentile confidence interval.
  # thetastar is the vector of bootstrapped theta^*s.
  #
  theta<-mean(x)
  thetastar<-rep(0,b)
  n<-length(x)
  for(i in 1:b){xstar<-sample(x,n,replace=T)
  thetastar[i]<-mean(xstar)
  }
  thetastar<-sort(thetastar)
  pick<-round((alpha/2)*(b+1))
  lower<-thetastar[pick]
  upper<-thetastar[b-pick+1]
  return(list(theta=theta,lower=lower,upper=upper))#,thetastar=thetastar))
  #list(theta=theta,lower=lower,upper=upper)
}

percentciboot(x, 3000, 0.1)

# 4.9.13 ----------------------------------------------------------------
pairboot <- function(x,y,b){
  delta = x - mean(x) - y + mean(y)
  n = length(delta)
  ts = mean(x) - mean(y)
  tstar = rep(0,b)
  counter = 0
  for(i in 1:b){xstar<-sample(delta,n,replace=T)
    tstar[i]<-mean(xstar)
    if (tstar[i] >= ts){
      counter = counter + 1
    }
  }
  counter = counter/b
  return(list(counter = counter))
}

x = c(23.500, 12.000, 21.000, 22.000, 19.125, 21.500, 22.125, 20.375,
      18.250, 21.625, 23.250, 21.000, 22.125, 23.000, 12.000) # Cross

y = c(17.375, 20.375, 20.000, 20.000, 18.375, 18.625, 18.625, 15.250,
      16.500, 18.000, 16.250, 18.000, 12.750, 15.500, 18.000) # Self

pairboot(x,y,3000)
```