Exploratory Data Analysis (S670):Assignment 6

Created by **Krishna Mahajan,0003572903**

Q1

Show that the jackknife estimate and the jackknife SE are exactly the same as sample mean $\bar{x}$ and standard error $se(\bar{x})= s/\sqrt{n}$ n when $\hat{\theta} = \bar{x}$ (sample mean).
(sol)
We know that jacknife estimate is similar to the bootstrap in that it involves resampling, but instead of sampling with replacement, the method samples without replacement.
Defination
The i_{th} jacknife replication of $\hat{\theta_{(i)}}$ of the statistic $\hat{\theta} = s(x)$ is $\hat{\theta_{(i)}} = s(x_i)$, $\forall i = 1, ...n$
Jacknife estimation of mean
$s(x_{(i)}) = \frac{1}{n-1}\sum_{j\neq i} x_j$
$= \frac{n\bar{x}-x_i}{n-1}$
$= \bar{x_i}$
now,
$\bar{x_{(.)}} = \frac{1}{n}\sum_{i=1}^{n} \bar{x_i} = \bar{x}$

Thus Jacknife estimate of sample mean is same as $\bar{x}$.

Jacknife estimate of the standard error of the mean

For $\hat{\theta} = \bar{x}$,it is easy to show that:

$\bar{x_i}= \frac{1}{n-1}\sum_{j\neq i} x_j$
$= \frac{n\bar{x}-x_i}{n-1}$
$= \bar{x_i}$ &

$\bar{x_{(.)}} = \frac{1}{n}\sum_{i=1}^{n} \bar{x_i} = \bar{x}$

Therefore:
$\hat{se_{jack}}(\bar{x})= (\sum_{i=1}^{n} \frac{x_i-\bar{x}^2}{(n-1)n})^{1/2}$
$=\frac{\bar{\sigma}}{\sqrt{(n)}}$
$=\frac{s}{\sqrt{(n)}}$

Q2

Show that the standard error of the pseudo-values (jackknife SE) is the same as the standard deviation of the " leave-out-one" values multiplied by $(n-1)/\sqrt{(n)}$ i.e. that the standard error of the pseudo-values (jackknife SE) is the same as the standard error of the “leave-out-one” values multiplied by (n -1).
(sol)
To Prove : $SE(PV's) = \frac{(n-1)}{\sqrt{(n)}}.SD(y_{(-i)})$ Here,
$y_{(-i)} = mean(xwithoutx_i) = \sum_{k\neq i} x_k/19$
$PV's = n.mean(y) - (n-1)y_{(-i)}$ (as Taught in Class)
so,
$var(PV's) = var(n.mean(y) - (n-1)y_{(-i)})$
$var(PV's) = var(n.mean(y)) + var((n-1)y_{(-i)})$
$var(PV's) = n^2 var(mean(y)) + (n-1)^2 var(y_{(-i)})$
$Sd(PV's) = n * sd(mean(y)) + (n-1) * sd(y_{(-i)})$ (i,e Taking root on both the sides to get standard deviation)

$SE(Pv's) = \frac{sd(mean(y))}{\sqrt(n)} + \frac{n-1sd(y_{(-i)})}{\sqrt(n)}$ (i,e dividing by $n^1/2$ on both sides to get SE)
Now, $\frac{sd(mean(y))}{\sqrt(n)} = 0$ as mean(y) is true population mean and it wont have any variance.
Thus,
$SE(Pv's) = \frac{n-1sd(y_{(-i)})}{\sqrt(n)}$
Alternatively we can also prove like this

Q3

Let x=LSAT , y= GPA. θ = 0.5 log((1+p)/(1-p)),where p is the coefficeint between x and y ,to be estimated by Pearson's correlation coefficient (r = 0.77637).
(a) Fisher showed that the distribution of $\hat{\theta}$=0.5log((1+r)/1-r) is roughly Gaussian with mean 0.5 log((1+p)/(1-p)) and variance 1/(n-3).Use Fisher's result to calculate an approximate 95% CI for θ

```
X = c(576, 635, 558, 578, 666, 580, 555, 661, 651, 605, 653, 575, 545, 572, 594)
Y = c(339, 330, 281, 303, 344, 307, 300, 343, 336, 313, 312, 274, 276, 288, 296)
Data = data.frame(X, Y)
colnames(Data) <- list("lsat", "gpa")

mean = 0.5*log(1.77637/(1-0.77637))
n = 15
var = 1/(n -3)
se = sqrt(var)/sqrt(n)
CI= mean + c(-1,1)*1.96*se
CI #95% CI for \theta
```

```
## [1] 0.8900774 1.1822569
```

(b) Use the jackknife method to estimate θ and derive an approximate 95% CIs for θ.

```
calculatePV = function(data) {
  n = length(data[[1]])
  rho = cor(data, method="pearson")[1,2]
  yall = 0.5*log((1+rho)/(1-rho))
  PV = numeric(n)
  for( i in 1:n) {
    rhominusi = cor(data[-i,], method="pearson")[1,2]
    yminusi = 0.5*log((1+rhominusi)/(1- rhominusi))
    PV[i] = n*yall - (n-1)*yminusi
  }
  PV
}
PVAll = calculatePV(Data)
JKEstimate = mean(PVAll) # Jackknife estimate is ##0.91703
varJK = sum((PVAll - JKEstimate)^2)/15*14
CI = JKEstimate + c(-1,1)*qt(0.975,df=nrow(Data)-1)*sqrt(varJK)
CI #95% CI for JackKnife Estimate
```

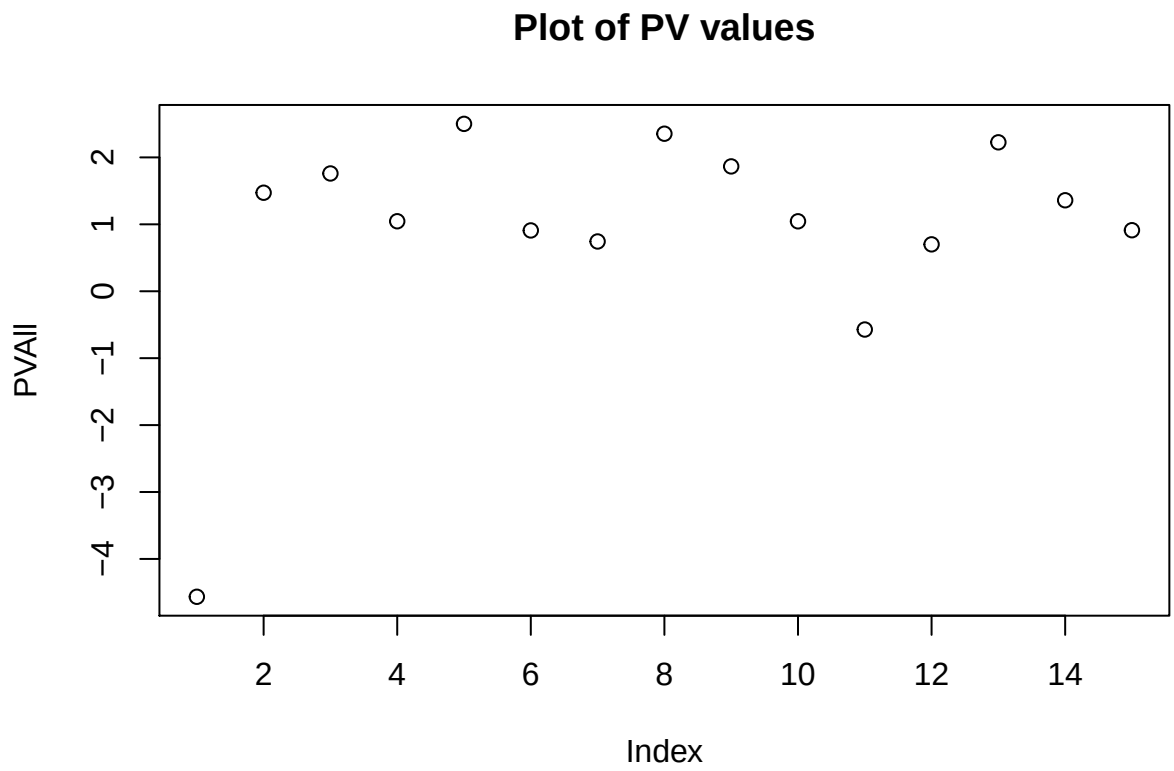
```
## [1] -12.31516  14.14923
```

(c) stem and leaf the psuedo-values.Do you see anything odd?Examine the Scatter Plot.Remove the first observation from the sample and repeat (b).

```
library(aplpack)
stem.leaf(PVAll)
```

```
## 1 | 2: represents 1.2
##  leaf unit: 0.1
##             n: 15
## LO: -4.56529576001235
##    2    -0. | 5
##         -0* |
##          0* |
##    6     0. | 7799
##   (4)    1* | 0034
##    5     1. | 78
##    3     2* | 23
##    1     2. | 5
```

```
plot(PVAll,main="Plot of PV values")
```




```
#Clearly -4.5 seems to be a outlier which corresponds to first row of the data.

PVre = calculatePV(Data[-1,])
JKEstimateRe = mean(PVre) #mean(PVre) #1.359
```

```r
varJKRe = sum((PVre - JKEstimateRe)^2)/(14*(14-1)) #0.1096
#seJKRecalc = sqrt(varJK) #0.33
CI = JKEstimateRe + c(-1,1)*qt(0.975,df=13)*sqrt(varJKRe)
CI
```

```
## [1] 0.6437573 2.0747180
```

We can observe confidence Interval changes significantly after removal of outlier Value.

(d) Use the bootstrap method to estimate θ.

```r
bootstrap = function(data, nsim) {
  theta = numeric(nsim)
  varTheta = numeric(nsim)

  n = length(data[[1]])
  index = 1:n
  for (i in 1:nsim){
    sampleindex= sample(index,n,replace=TRUE)
    PViter = calculatePV(data[sampleindex, ])
    theta[i] = mean(PViter)
    varTheta[i] = sum((PViter - theta[i])^2)/(n*(n-1))
  }

  ciLower = mean(theta) - 1.96*mean(varTheta)
  ciUpper = mean(theta) + 1.96*mean(varTheta)

  output = list(thetaBS = mean(theta), varBS = mean(varTheta),
                theta = theta, varTheta = varTheta,
                ciLower = ciLower, ciUpper = ciUpper)
  output
}
Results = bootstrap(Data, 10)
Results
```

```
## $thetaBS
## [1] 0.8009363
##
## $varBS
## [1] 0.1350431
##
## $theta
##  [1] 0.7346316 0.6366669 0.7822350 0.8513318 0.7801105 0.9319810 0.7257140
##  [8] 0.6910083 1.0082249 0.8674592
##
## $varTheta
##  [1] 0.10935954 0.08127249 0.05750318 0.18316466 0.07420528 0.15195327
##  [7] 0.06221169 0.07117708 0.22364666 0.33593685
##
## $ciLower
## [1] 0.5362519
##
## $ciUpper
## [1] 1.065621
```

(e) Derive the approximate 95% CIs for θ via the bootstrap.And compare them with the ones you obtained in (a),(b) and (d).
(sol) We see how bottstrapping and jackknifing reduce the confidence interval because in both the methods the effect of the outlier on the confidence interval is reduced. In part a, we got the 95% CI interval between 0.89, 1.182
In part b, we got between -12.31 and 14.14 & 0.6437,2.0747 after removing outlier
In part c we got 0.67,1.35 through bootstrapping
The effect of the outlier is reduced the max by bootstrapping even without removing the outlier from the data

Q4

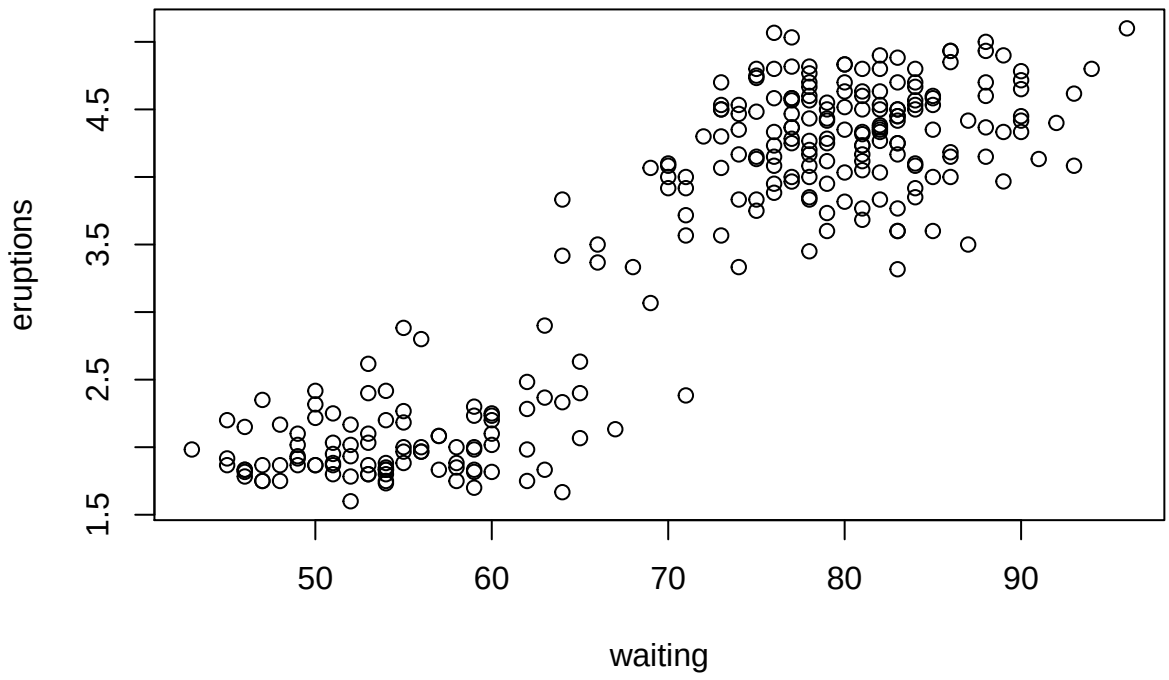
In simple linear regression we are interested in modeling the response y as a simple linear function of some predictor variable x under the model $y = \alpha + \beta x + \epsilon$ where α is the y-intercept ,β is the slope and ϵ is a vector of errors or departures from the line. In resistant regression we are interested in estimating the y-intercept and slope parameters in a resistant fashion. For example, the program run.rrline() is a program which will calculate estimators for the parameters $\hat{\theta} = \hat{\alpha}, \hat{\beta}$ in a resistant fashion,however I have no closed form estimators for the standard errors $\hat{SE}(\hat{\alpha})$ and $\hat{SE}(\hat{\beta})$,the bootstrap estimators for the bias of $\hat{\theta} = \hat{\alpha}, \hat{\beta}$ and an estimator the out of sample predictive error (or out-of-bag error) $\hat{Err}$.Your function should be constructed so that it has two arguments:a data frame and the number of bootstrap replicates, R.The input data frame should contain a column for the y variable and a column for the x variable, and the column names of the data frame can be labeled as such. Once you have constructed this function practice using this function on the "faithful" dataset in R. For this particular faithful data set, treat x-variable as waiting and the y-variable as eruptions.

```
#Code courtesy: Prof David King Lecture Notes
rrline1 <- function(x,y) {
  n3 <- floor((length(x)+1.99)/3)
  x.order <- order(x)
  medxL <- median(x[x.order][1:n3])
  medxR <- median(rev(x[x.order])[1:n3])
  medyL <- median(y[x.order][1:n3])
  medyR <- median(rev(y[x.order])[1:n3])
  slope1 <- (medyR - medyL)/(medxR - medxL)
  int1 <- median(y - slope1 * x)
  # print(c(paste("Intercept = ", format(round(int1,5))),
  #   paste("Slope = ",format(round(slope1,5)))))
  newy <- y - slope1*x - int1
  sumres <- sum(abs(newy))
  list(a=int1, b=slope1, sumres = sumres, res=newy)
}
#Code courtesy: Prof David King Lecture Notes
run.rrline <- function(x,y,iter=5) {
  out.coef <- matrix(0,iter,3)
  newy <- y
  for (i in 1:iter) {
    rr <- rrline1(x,newy)
    out.coef[i,] <- c(rr$a,rr$b,rr$sumres)
    newy <- rr$res
  }
  dimnames(out.coef) <- list(format(1:iter),c("a","b","|res|"))
  aa <- sum(out.coef[,1])
  bb <- sum(out.coef[,2])
```

```
  cc <- sum(abs(y - aa - bb*x))
  res <- y - aa - bb*x
  out.coef <- rbind(out.coef,c(aa,bb,cc))
  #print(round(out.coef,5))
  list(a = aa, b = bb, res = res, coef=out.coef)
}
#Code courtesy : Prof David King Lecture Notes
bootprog = function (x,nsim)
{
  # This program is a silly program which will be used to estimate the
  # bootstap error of the sample median statistic
  #  the input data is a vector x of data.
  #  nsim is the number of bootstrap simulations
  n = length(x)
  index = 1:n
  m = median(x)
  stat = numeric(nsim)
  ooberr = numeric(nsim)
  for (i in 1:nsim){
    sampleindex= sample(index,n,replace=TRUE)
    stat[i] = median(x[sampleindex])
    oobindex = setdiff(index,unique(sampleindex))
    oobdat = x[oobindex]
    ooberr[i] = sum((oobdat-stat[i])^2)/length(oobindex)
  }
  bias = m - mean(stat)
  variance = var(stat)
  se = sqrt(variance)
  avgooberr = mean(ooberr)
  output = list(bias=bias,var=variance,se=se,avgooberr=avgooberr)
  output
}

#Code courtesy: Prof David King Lecture Notes
cvprog = function (x,nfold)
{
  # This program is a silly program which will be used to estimate the
  # crossvalidation error of the sample median statistic
  # the input data is a vector x of data.
  #  nfold is the number of folds you want to divide your data up into
  n = length(x)
  m = floor(n/nfold)
  # Generally speaking n/nfold would be an integer, however if it is not
  # and the remainder of n/nfold is k then we will take the extra k datapoints
  # and give them to the first k folds.
  folds = rep(1:nfold,m)
  k = n - length(folds)
  if(k>0){folds = c(folds,1:k)}
  # now folds is of length n and we can randomly permute the indicies
  foldindicies = sample(folds,n,replace=FALSE)
  m = median(x)
  stat = numeric(nfold)
  cverr = numeric(nfold)
```

```
  for (i in 1:nfold){
    b = foldindicies == i
    stat[i] = median(x[!b])
    cverr[i] = sum((x[b]-stat[i])^2)/length(x[b])
  }
  bias = m - mean(stat)
  variance = var(stat)
  se = sqrt(variance)
  avgcverr = mean(cverr)
  output = list(bias=bias,var=variance,se=se,avgcverr=avgcverr)
  output
}

getOOBforBootstrap = function(oobdata) {
  # Based on the class slides and hints from the professor,
  # this function calculates rrline for
  # the oob data to get a and b, shuffles the residuals,
  # adds them to the original data
  # calculate rr line again to get new a and b, this is repeated till we have
  # n estimates of a and b where n is the number of oob samples
  # oobdata
  originalData = oobdata
  n = length(originalData[[1]])
  aOutofBag = numeric(n)
  bOutofBag = numeric(n)
  for (q in 1: n) {
    results = run.rrline(originalData[[1]], originalData[[2]])
    residuals = results$res
    aOutofBag[q] = results$a
    bOutofBag[q] = results$b
    shuffledResiduals = sample(residuals)
    originalData[[2]] = oobdata[[2]] + shuffledResiduals
  }
  list(a0 = aOutofBag, b0 = bOutofBag)
}



rrlineWithBootstrap = function(data, nsim) {
  # This function runs rrline with bootstrapping
  n = length(data[[1]])
  index = 1:n

  # We maintain 2 different stats and oob for each a and b
  statA = numeric(nsim)
  statB = numeric(nsim)
  ooberrA = numeric(nsim)
  ooberrB = numeric(nsim)

  # Run rrline to get initial statistic on entire data,
  # for confirmatory purposes only
  results = run.rrline(data[[1]], data[[2]])
  a = results$a
```

```
  b = results$b

  #Run nsim times
  for (i in 1:nsim){
    sampleindex= sample(index,n,replace=TRUE)
    results = run.rrline(data[[1]][sampleindex], data[[2]][sampleindex])
    statA[i] = results$a
    statB[i] = results$b

    oobindex = setdiff(index,unique(sampleindex))
    oobResults = getOOBforBootstrap(data[oobindex,])

    ooberrA[i] = sum((oobResults$a0-statA[i])^2)/length(oobindex)
    ooberrB[i] = sum((oobResults$b0-statB[i])^2)/length(oobindex)
  }

  # Calculate bias and variance
  biasA = a - mean(statA)
  varianceA = var(statA)
  biasB = b - mean(statB)
  varianceB = var(statB)

  # Calculate standard error and average oob error
  seA = sqrt(varianceA)
  seB = sqrt(varianceB)
  avgooberrA = mean(ooberrA)
  avgooberrB = mean(ooberrB)

  output = list(a= mean(statA), b = mean(statB),
                biasA=biasA,varA=varianceA, biasB = biasB, varB = varianceB,
                seA=seA,seB = seB, ooberrA = ooberrA, ooberrB = ooberrB,
                avgooberrA=avgooberrA, avgooberrB = avgooberrB)
  output
}

q4Data = data.frame(faithful$waiting, faithful$eruptions)
colnames(q4Data) <- c("waiting", "eruptions")
plot(q4Data)
```




```
resultsq4 = rrlineWithBootstrap(q4Data, 5)

resultsq4
```

```
## $a
## [1] -2.024288
##
## $b
## [1] 0.07792406
##
## $biasA
## [1] 0.01688714
##
## $varA
## [1] 0.06122154
##
## $biasB
## [1] -0.0001240512
##
## $varB
## [1] 1.084102e-05
##
## $seA
## [1] 0.2474299
##
```

```
## $seB
## [1] 0.00329257
##
## $ooberrA
## [1] 3.495000 3.557046 3.899154 4.311185 5.051458
##
## $ooberrB
## [1] 0.0007244419 0.0007442197 0.0007704949 0.0007939759 0.0010260165
##
## $avgooberrA
## [1] 4.062769
##
## $avgooberrB
## [1] 0.0008118298
```

Q5

In this problem, I will ask you to do something similar as the previous problem, except this time use k-fold cross-validation rather than bootstrapping.That is to say, for the program run.rrline() construct a function which will calculate the k-fold cross-validation estimator for the out-of-sample error $\hat{err}$.Your function should be constructed so that it has two arguments: a data frame and the number of desired CV folds k.Like the previous problem, the input data frame should contain a column for the y variable and a column for the x variable, and the column names of the data frame can be labeled as such. Once you have constructed this function practice using this function on the "faithful" dataset in R, treating the x-variable as waiting and the y-variable as eruptions.