# Assignment #1 – Machine Learning – Professor Haugh

## **Jaime Gacitua**

## **jg3499**

## **Monday February 15, 2016**

### Question 1

Steps 1 to 3 can be found in the R code at the Appendix.

Step 4.

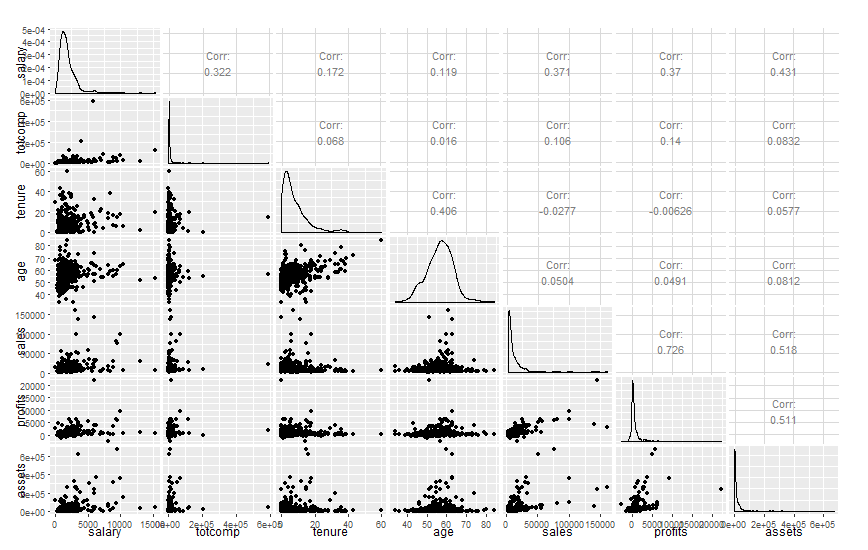
Conducting EDA to make sure the data is random enough. As a criteria for this, I will look that ‘train data’ and ‘test data’ have a similar distribution as the full dataset. In this sense, I will use pair plots for each of the 3 collections of data, and use the correlations and the distributions to check that they have similar distributions.

I ran 500000 different samples for the “training set” and “testing set”. For each subset, I would calculate their covariance matrix. I kept the subset that produced covariance matrices closest to the whole dataset.

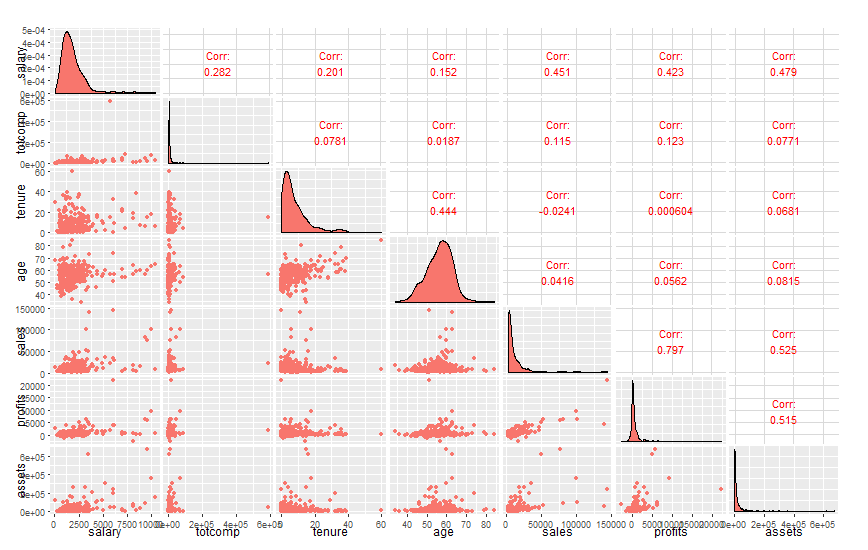
As a result,

* Pair plots indicate that the correlations and histogram of “test set” and “training set” are similar to the whole dataset.
* Box plots show that the mean and quartiles are quite aligned for all the sets.

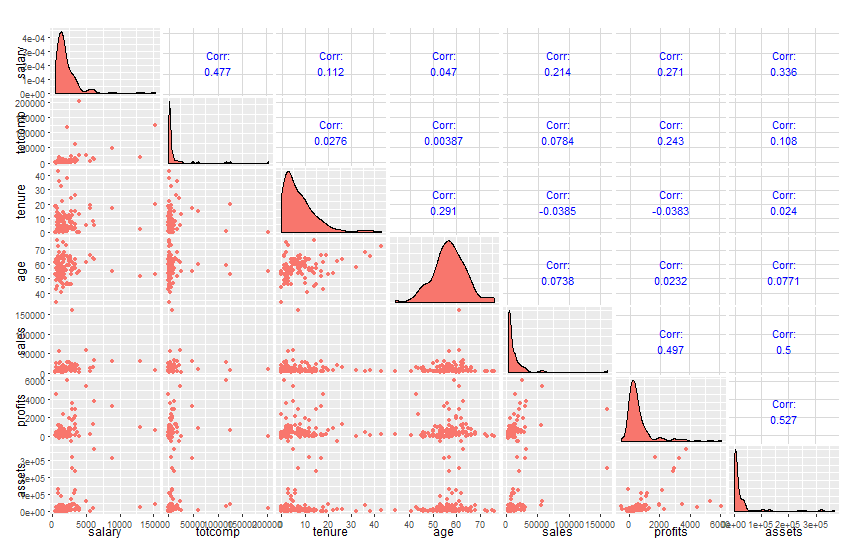
Pair Plot of Whole Dataset



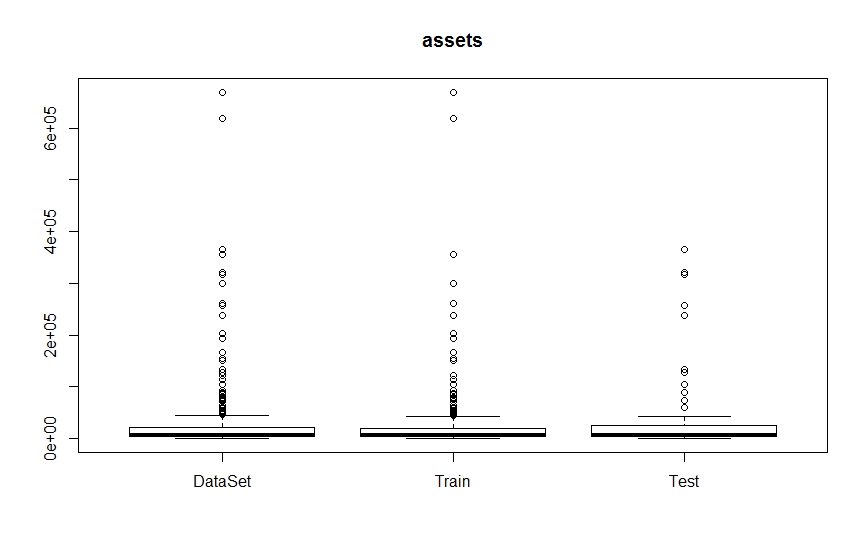
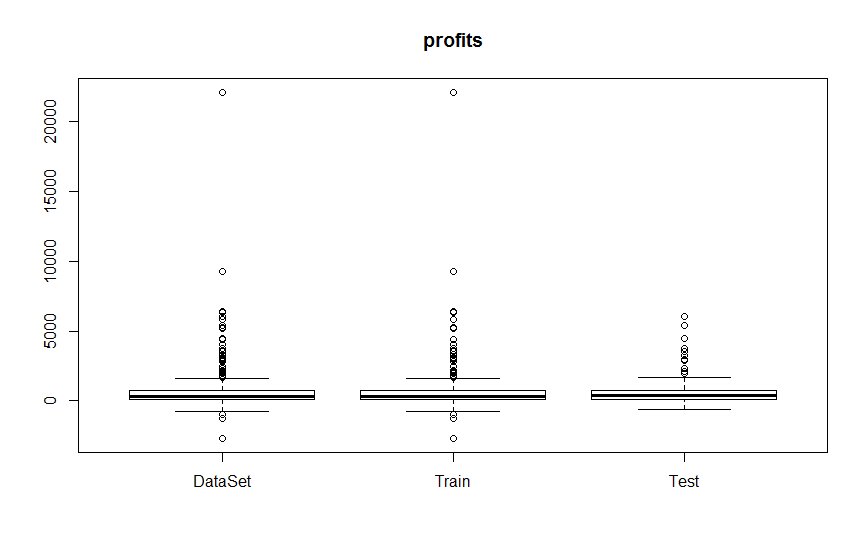
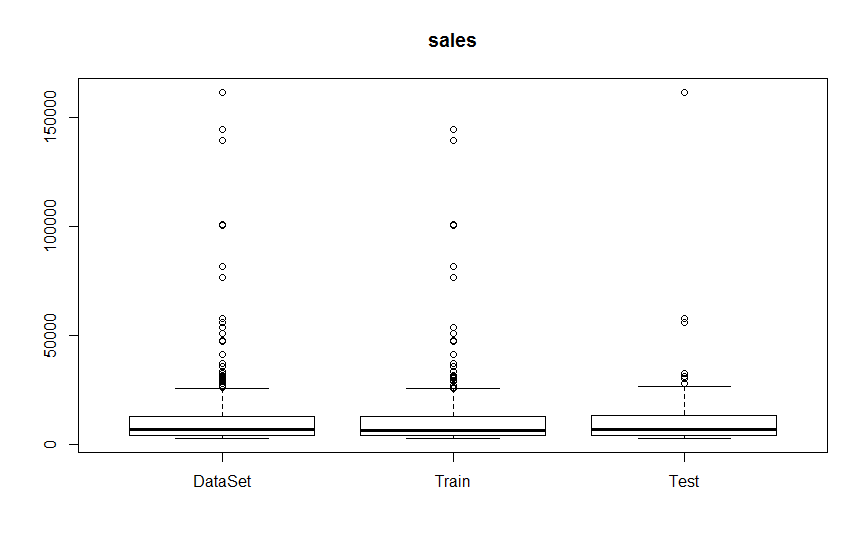
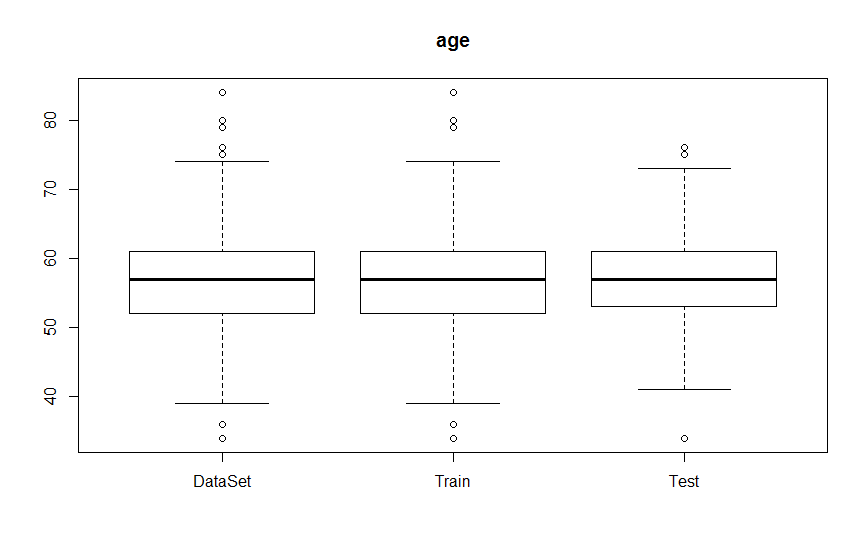
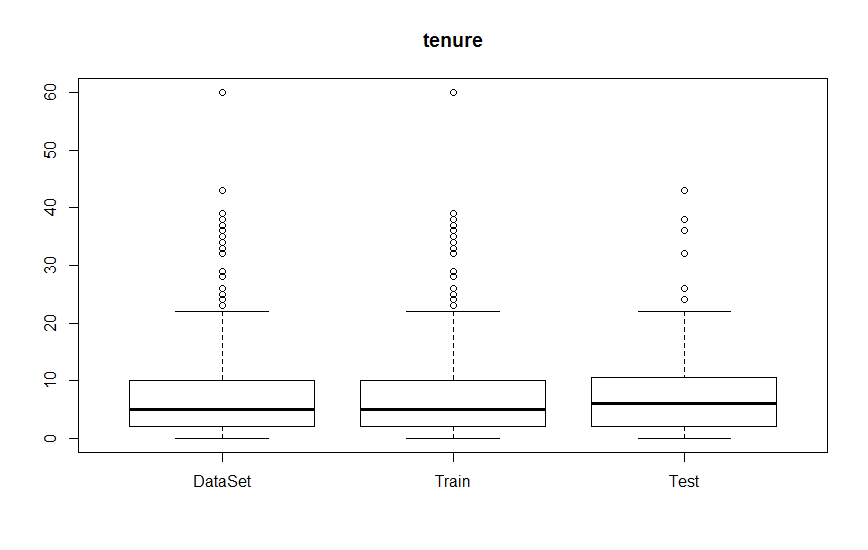
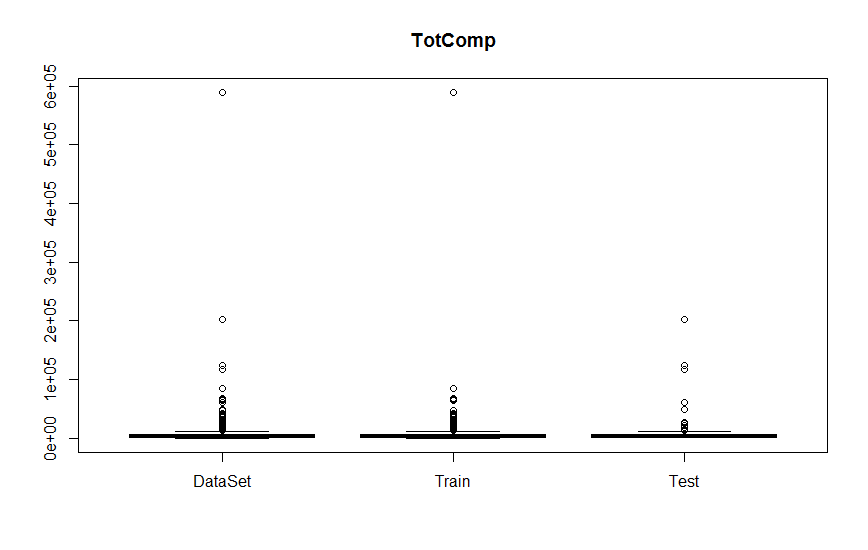
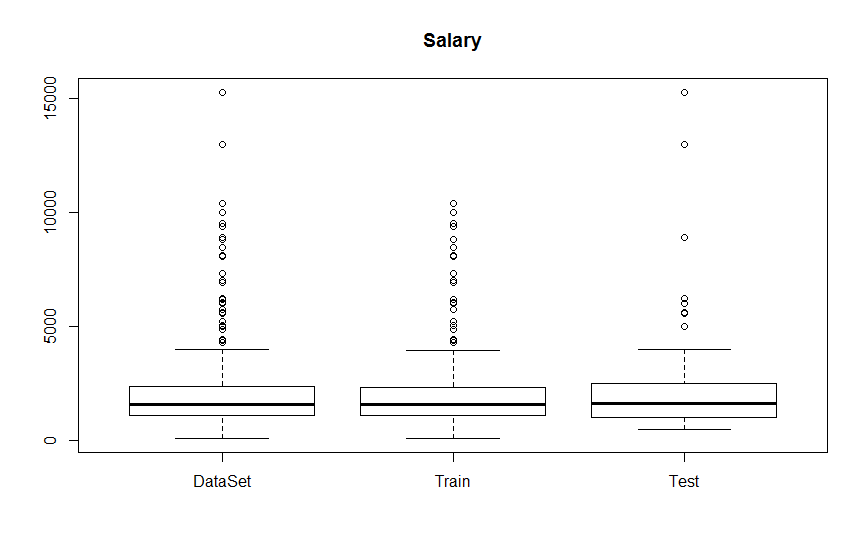
Pair Plot of Training Set



Pair Plot of Test Set



Box Plots of the 7 variables in study.



Step 5.

I standardized all the independent variables of the test set, where

* Independent variables = Salary, tenure, age, sales, profits, assets
* Dependent Variable = totcomp

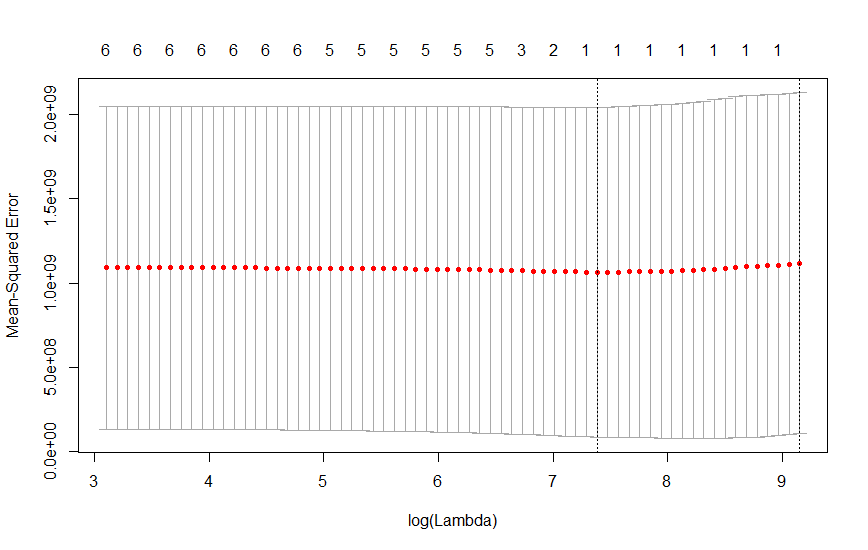
Care was taken to standardize using only the information on the test set.

Step 6.

Estimated the lasso model over **totcomp** using glmnet library, and the function cv.glmnet over the standardized data.

Below is the plot for the MSE vs :

* The mean-squared error seems very big. We are probably not fitting a good model on this data.
* For different values of lambda there is no substantial improvement for the MSE. We can make a simple model with few variables because adding complexity will not improve the prediction capacity of the model.



Regarding the linear model coefficients, the only non-zero coefficients are “intercept” and “salary”.

7 x 1 sparse Matrix of class "dgCMatrix"

1

(Intercept) 8015.260

salary 7830.159

tenure .

age .

sales .

profits .

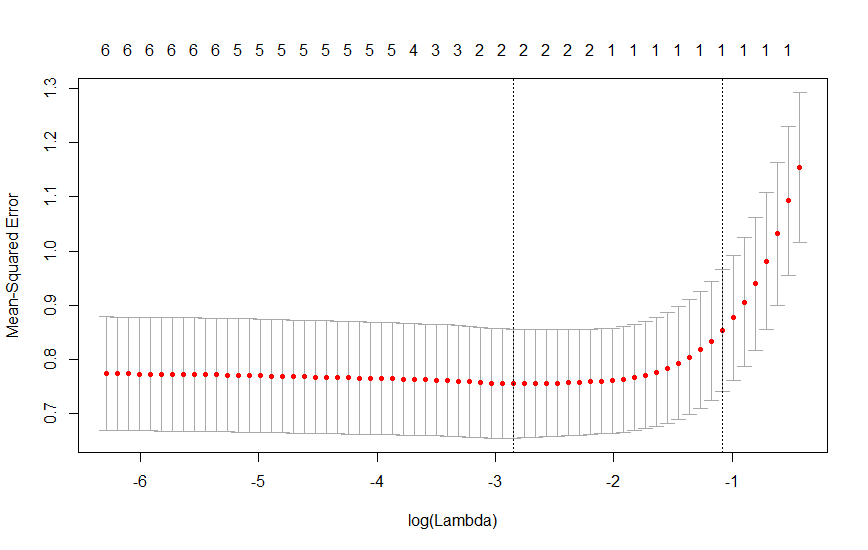
assets .

Step 7.

Estimated the lasso model over **log(totcomp)**, using glmnet library, and the function cv.glmnet over the standardized data.

Below is the plot for the MSE vs :

* The mean error is substantially lower than in Step 6. This seems to be a much better fitting model.
* There is a clear improvement when lowering lambda, increasing the number of nonzero predictors.



Regarding the linear model coefficients, all 6 of the predictor variables are used in this model.

7 x 1 sparse Matrix of class "dgCMatrix"

1

(Intercept) 8015.2597

salary 10271.0896

tenure 1309.2606

age -1347.1162

sales -603.4947

profits 1831.4061

assets -2930.8099

Step 8.

The lambdas that gave the least cross validation MSE are , . We can see that the **log(totcomp)** model has less shrinkage compared to the **totcomp** model.

I standardized the test variables using the same mean and standard deviation used in the training set. I ran the model and found out that the **totcomp** model gave a much lower MSE than the **log(totcomp)** model.

* The MSE for the first lasso model is 2.064459e+10
* The MSE for the second lasso model, within the log(totcomp) space is 4.16925e+05

### Question 2.

1. After fitting a logistic regression, the standard errors for “balance” and “income” are 2.274e-04 and 4.985e-06 respectively.

Call:

glm(formula = default ~ balance + income, 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 \*\*\*

**balance 5.647e-03 2.274e-04 24.836 < 2e-16 \*\*\***

**income 2.081e-05 4.985e-06 4.174 2.99e-05 \*\*\***

---

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

The proposed function is as follows:

boot.fn <- function(def.data, obs.index) {

data2 = def.data[obs.index,]

glm.fit = glm(default~balance+income, data=data2, family=binomial)

coefs = c(glm.fit$coefficients[2], glm.fit$coefficients[3])

return(coefs)

}

An example output for the first 10 data points is

> boot.fn(Default, c(1,2,3,4,5,6,7,8,9,10))

balance income

-1.306747e-17 6.349237e-19

1. Once combining the boot and the boot.fn functions, through the following code, I got the following errors.

boot(Default, boot.fn, R = 999)

Bootstrap Statistics :

original bias std. error

t1\* 5.647103e-03 2.172333e-06 2.241758e-04

t2\* 2.080898e-05 5.394155e-08 4.805121e-06

1. We can see that the standard error given by the bootstrap is very similar to the error in the logistic regression.

Table 1: Standard Errors of Bootstrap vs Logistic Regression

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | BootStrap | Logistic | Dif |
| Balance | 2.24E-04 | 2.27E-04 | -3.22E-06 |
| Income | 4.81E-06 | 4.99E-06 | -1.80E-07 |

We can also see that the mean values of the bootstrap (“original” column) is also very close to the coefficients of the glm() output.

### Question 3.

Before answering the questions, it is important to note that we are in a “Ridge Regression” type of model. This shrinkage model balances the bias-variance magnitudes including the 2-norm of the betas and a parameter , in the objective function. As increases, the will be steadily shrunk, and this will have different implications on the questions answered below.

1. iii. The training RSS will steadily increase.

The RSS are minimized when we minimize the sum of squares, in other words, when . As lambda increases, the second term of the ridge regression gains more and more importance, driving away the ’s from the solution that minimizes the RSS.

1. ii. The test RSS will decrease initially, and then eventually start increasing in a U Shape.

Starting at , finding the that minimizes the sum of squares in the training set, does not guarantee that it will also minimize the sum of squares tor the test set. As we increase , the shrinkage method will at some point find a combination of bias and variance that will be best suited for the test set, finding a minimizer at . After the minimizer, increasing beyond that point will just make the test RSS increase.

1. iv. The variance will steadily decrease.

As increases, the will steadily shrink to zero. This process makes the model less and less complex. The shrinking will be finished when , when we expect to have , except for , the intercept. This ultimate model will just be a constant, with variance = 0, representing the average of the ’s.

1. iii. The bias (squared) will steadily increase.

As increases, the will steadily shrink to zero. This process makes the model less and less complex. The shrinking will be finished when , when we expect to have , except for , the intercept. This ultimate model will just be a constant, extremely biased towards the average, of the ’s.

1. v. The irreducible error will remain constant.

The irreducible error is just what its name represents. This error cannot be reduced using any technique. This magnitude is related to the inherent noise in the error term and cannot be eliminated.

### Question 5.

1. The mean of the variable medv is
2. The standard error was calculated dividing the sample standard deviation by the square root of the number of observations.
3. The bootstrap statistics offer a very similar standard error as calculated in part (b). It is an indicator I would use in the future.

Bootstrap Statistics :

original bias **std. error**

t1\* 22.53281 -0.0002201087 **0.4084123**

1. Using the hint, the confidence interval using the bootstrap information is given by:

When we run the t-test over the sample, we get very similar results as the bootstrap confidence interval:

One Sample t-test

data: Boston$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

1. To calculate the median, I would use the summary() function in R. The output is the following:

Min. 1st Qu. **Median** Mean 3rd Qu. Max.

5.00 17.02 **21.20** 22.53 25.00 50.00

1. To estimate the error, I would use bootstrap over this statistic. We can see that the standard error for the median is lower than the standard error for the mean.

Bootstrap Statistics :

original bias **std. error**

t1\* 21.2 -0.012119 **0.3798412**

The estimated error would be

1. Using the quantile() function in R, we get the following value for the 10% percentile:
2. Using the bootstrap, I estimate the standard error of this statistic. It is interesting to note that the standard error is higher than the median and the mean.

Bootstrap Statistics :

original bias **std. error**

t1\* 12.75 0.008456 **0.5027847**

### Question 6.

I will develop the term including the , and demonstrate it takes the same form of the minimization function for the ridge regression.

* The first term can be recognized as the RSS.
* The second term cancels out, because
* The third term, we will be left only with the quadratic terms, because all of the cross terms have the form with , and since they are independent, we are left with

If we set , we reach the ridge regression function, that we want to minimize:

## Code for Q1

require(ggplot2)

require(GGally)

require(glmnet)

## Step 1

## Read the data into a data frame

ceo\_data <- read.csv(".\\ceo.csv")

ceo\_data<-as.data.frame(ceo\_data)

names(ceo\_data)

summary(ceo\_data)

## Step 2

## Remove last column of de data frame

ceo\_data <- ceo\_data[,0:7]

names(ceo\_data)

summary(ceo\_data)

## Step 3 and 4

## Selecting sufficiently random training and test sets.

## For now deleted using the if false statement. Takes a lot of time.

if(FALSE){

minDif <- 999999999999

count <- 0

iter <- 500000

while(count < iter){

#Generate random sample for training and testing

train\_index <- sample(nrow(ceo\_data), floor(nrow(ceo\_data) \* 0.75))

trainData <- ceo\_data[train\_index, ]

testData <- ceo\_data[-train\_index, ]

difTrain <- abs(cov(ceo\_data) - cov(trainData))

difTest <- abs(cov(ceo\_data) - cov(testData))

totalDif <- sum(difTrain) + sum(difTest)

if(totalDif < minDif){

train\_index\_min <- train\_index

minDif <- totalDif

}

count <- count + 1

if(count - floor(count/1000)\*1000 == 0){

print(count)

}

}

# Use the best set

trainData <- ceo\_data[train\_index\_min, ]

testData <- ceo\_data[-train\_index\_min, ]

}

### Step 4.

## EDA to check if the split is random enough

## Pair Plots

ggpairs(ceo\_data, diag=list(continuous="densityDiag", discrete="barDiag"), axisLabels="show")

ggpairs(trainData, diag=list(continuous="densityDiag", discrete="barDiag"), axisLabels="show", aes(colour='red'))

ggpairs(testData, diag=list(continuous="densityDiag", discrete="barDiag"), axisLabels="show", aes(colour='blue'))

## Box Plots

boxplot(ceo\_data$salary, trainData$salary, testData$salary,

names = list("DataSet", "Train", 'Test'), main = "Salary")

boxplot(ceo\_data$totcomp, trainData$totcomp, testData$totcomp,

names = list("DataSet", "Train", 'Test'), main = "TotComp")

boxplot(ceo\_data$tenure, trainData$tenure, testData$tenure,

names = list("DataSet", "Train", 'Test'), main = "tenure")

boxplot(ceo\_data$age, trainData$age, testData$age,

names = list("DataSet", "Train", 'Test'), main = "age")

boxplot(ceo\_data$sales, trainData$sales, testData$sales,

names = list("DataSet", "Train", 'Test'), main = "sales")

boxplot(ceo\_data$profits, trainData$profits, testData$profits,

names = list("DataSet", "Train", 'Test'), main = "profits")

boxplot(ceo\_data$assets, trainData$assets, testData$assets,

names = list("DataSet", "Train", 'Test'), main = "assets")

### Step 5

# Standarizing independent variables

y.var = names(trainData) %in% c('totcomp')

trainData.x <- scale(trainData[!y.var])

trainData.y <- trainData[y.var]

colMeans(trainData.x)

sd(trainData.x[,2])

# REMEMBER When I standarize the test set, I have to use the same mean and stdev to center and scale.

### Step 6 Fitting a Lasso model with 10-fold CV

lasso.cv=cv.glmnet(trainData.x, unlist(trainData.y), alpha = 1)

plot(lasso.cv)

bestlam = lasso.cv$lambda.min

out = glmnet(trainData.x, unlist(trainData.y), alpha=1)

lasso.coef = predict(out,type = "coefficients", s=bestlam)

lasso.coef

### Step 7 Fitting a Lasso model with 10-fold CV to log(totcomp)

trainData.y.log <- log(trainData.y)

lasso.cv.log=cv.glmnet(trainData.x, unlist(trainData.y.log), alpha = 1)

plot(lasso.cv.log)

lasso.cv.log

bestlam.log = lasso.cv.log$lambda.min

out.log = glmnet(trainData.x, unlist(trainData.y.log), alpha=1)

lasso.coef.log = predict(out,type = "coefficients", s=bestlam.log)

lasso.coef.log

### Step 8

testData.y <- testData$totcomp

testData.x <- testData[!y.var]

# Get Mean and Stdev of training set to standarize the X's in the test set

means.x <- colMeans(trainData[,!y.var])

stdev.x <- sqrt(diag(var(trainData[,!y.var])))

testData.x <- as.matrix((testData.x - means.x) / stdev.x)

## First Lasso Model

lasso.pred <- predict(out,newx=testData.x,s=bestlam)

lasso.error <- mean((lasso.pred-testData.y)^2)

lasso.error

## Second Lasso Model (log)

lasso.pred.log <- predict(out.log, newx=testData.x, s=bestlam.log)

lasso.error.log <- mean((exp(lasso.pred.log)-testData.y)^2)

lasso.error.log

matrix(c(lasso.pred.log, exp(lasso.pred.log), testData.y, exp(lasso.pred.log) - log(testData.y) ), nrow=length(lasso.pred.log))

### Code for Q2

#ISLR Chapter 5 question 6.

names(city)

city

require(ISLR)

require(boot)

summary(Default)

names(Default)

set.seed(1)

## Part a

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

summary(glm.fit)

Default[c(1,2,3,4),]

## Part b

boot.fn <- function(def.data, obs.index) {

data2 = def.data[obs.index,]

glm.fit = glm(default~balance+income, data=data2, family=binomial)

coefs = c(glm.fit$coefficients[2], glm.fit$coefficients[3])

return(coefs)

}

boot.fn(Default, c(1,2,3,4,5,6,7,8,9,10))

## Part c

boot(Default, boot.fn, R = 999)

### Code for Q5

require(MASS)

require(boot)

## checking there are no NA values

Boston$medv[is.na(Boston$medv)]

## Part a

medv.mean <- mean(Boston$medv)

## Part b

medv.se <- sd(Boston$medv) / sqrt(length(Boston$medv))

## Part c

# Define the function to call with boot

se.function = function(data,index){ # INput args are: (i) the data and (ii) an index vector defining what observations to use

data2 = data[index]

return(mean(data2))

}

se.function(Boston$medv, sample(length(Boston$medv), 500)) # test for one realization

se.boot <- boot(Boston$medv, se.function, R=100000)

se.boot

## Part d

se.boot$t0[1] + 2\*sd(se.boot$t)

se.boot$t0[1] - 2\*sd(se.boot$t)

t.test(Boston$medv)

## Part e

summary(Boston$medv)[3]

## Part f

# Define the function to call with boot

med.function = function(data,index){ # INput args are: (i) the data and (ii) an index vector defining what observations to use

data2 = data[index]

return(summary(data2)[3])

}

med.boot <- boot(Boston$medv, med.function, R=100000)

med.boot

## Part g

?quantile

med.percentile10 <- quantile(Boston$medv, 0.1)

med.percentile10

## Part h

# Define the function to call with boot

perc10.function = function(data,index){ # INput args are: (i) the data and (ii) an index vector defining what observations to use

data2 = data[index]

return(quantile(data2, 0.1))

}

med.boot.perc10 <- boot(Boston$medv, perc10.function, R=100000)

med.boot.perc10