Homework 2C - Aniket Maheshwari

UB Person Number: 50360266

Clear the environment

```
rm(list = ls())
```

First, we will set the directory of the R script

```
setwd("C:/Users/anike/Desktop/Sem 1/EAS 506 Statistical Data
Mining/Homework/Homework 2")
```

Importing all the necessary libraries:

```
library(leaps)
library(MASS)
library(ISLR)
library(lattice)
library(ggplot2)
library(corrplot)

## corrplot 0.90 loaded

library(car)

## Loading required package: carData

library(caret)
library(coefplot)
library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.1-2
```

Importing Dataset and viewing it:

```
data("College")
College_data = College
dim(College_data)

## [1] 777  18

str(College_data)

## 'data.frame':    777 obs. of  18 variables:
##  $ Private    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Apps       : num  1660 2186 1428 417 193 ...
##  $ Accept     : num  1232 1924 1097 349 146 ...
##  $ Enroll     : num  721 512 336 137 55 158 103 489 227 172 ...
```

```
##  $ Top10perc  : num  23 16 22 60 16 38 17 37 30 21 ...
##  $ Top25perc  : num  52 29 50 89 44 62 45 68 63 44 ...
##  $ F.Undergrad: num  2885 2683 1036 510 249 ...
##  $ P.Undergrad: num  537 1227 99 63 869 ...
##  $ Outstate   : num  7440 12280 11250 12960 7560 ...
##  $ Room.Board : num  3300 6450 3750 5450 4120 ...
##  $ Books      : num  450 750 400 450 800 500 500 450 300 660 ...
##  $ Personal   : num  2200 1500 1165 875 1500 ...
##  $ PhD        : num  70 29 53 92 76 67 90 89 79 40 ...
##  $ Terminal   : num  78 30 66 97 72 73 93 100 84 41 ...
##  $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
##  $ perc.alumni: num  12 16 30 37 2 11 26 37 23 15 ...
##  $ Expend     : num  7041 10527 8735 19016 10922 ...
##  $ Grad.Rate  : num  60 56 54 59 15 55 63 73 80 52 ...
```

So, the college data has 777 rows and 18 columns. It just has one categorical variable Private with two levels: 'Yes' and 'No'. All the other fields are numerical. Our target Variable is Apps i.e., Number of applications.

Before starting EDA, first I'll check if there is any NULL data or missing value in dataset.

Checking if data has any NULL values:

```
indx <- apply(College_data, 2, function(x) any(is.na(x)))
indx
```

```
##     Private        Apps      Accept      Enroll   Top10perc   Top25perc
##       FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
## F.Undergrad P.Undergrad    Outstate  Room.Board       Books    Personal
##       FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
##         PhD    Terminal   S.F.Ratio perc.alumni      Expend   Grad.Rate
##       FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
```

Since all the columns return FALSE value, this means that none of them has any NULL values.

Normalize: Let's normalize the data because the features value range is very different is all the features and that will cause problem in the analysis.

```
normalize <- function(x) {
  (x -min(x)) / (max(x) - min(x))

}

college_data_norm <- as.data.frame(lapply(College_data[2:18], normalize))
head(college_data_norm , 4)
```

```
##          Apps     Accept     Enroll Top10perc Top25perc F.Undergrad
P.Undergrad
## 1 0.032886926 0.04417701 0.10791254 0.2315789 0.4725275  0.08716353
0.024547744
```

```
## 2 0.043842293 0.07053089 0.07503539 0.1578947 0.2197802  0.08075165
0.056148386
## 3 0.028054902 0.03903572 0.04734938 0.2210526 0.4505495  0.02847257
0.004488207
## 4 0.006998105 0.01054917 0.01604530 0.6210526 0.8791209  0.01177628
0.002839478
##    Outstate Room.Board     Books   Personal       PhD   Terminal S.F.Ratio
## 1 0.2634298  0.2395965 0.1577540 0.29770992 0.6526316 0.71052632 0.4182306
## 2 0.5134298  0.7361286 0.2914439 0.19083969 0.2210526 0.07894737 0.2600536
## 3 0.4602273  0.3105296 0.1354724 0.13969466 0.4736842 0.55263158 0.2788204
## 4 0.5485537  0.5784994 0.1577540 0.09541985 0.8842105 0.96052632 0.1394102
##   perc.alumni    Expend Grad.Rate
## 1    0.187500 0.0726714 0.4629630
## 2    0.250000 0.1383867 0.4259259
## 3    0.468750 0.1046053 0.4074074
## 4    0.578125 0.2984146 0.4537037
```

```
full_dataset <- cbind(College_data,college_data_norm )
full_dataset <- subset(full_dataset , select = -c(2:18))
head(full_dataset ,1)
```

```
##                               Private       Apps     Accept    Enroll
Top10perc
## Abilene Christian University     Yes 0.03288693 0.04417701 0.1079125
0.2315789
##                               Top25perc F.Undergrad P.Undergrad  Outstate
## Abilene Christian University 0.4725275  0.08716353  0.02454774 0.2634298
##                               Room.Board    Books  Personal       PhD
Terminal
## Abilene Christian University  0.2395965 0.157754 0.2977099 0.6526316
0.7105263
##                               S.F.Ratio perc.alumni    Expend Grad.Rate
## Abilene Christian University 0.4182306      0.1875 0.0726714  0.462963
```

Now all the columns are in range [0,1].

Ans a) Splitting the data into train and test:

In this data, we do have a categorical feature Private so i did a Stratified Sampling into Train and Test based on Private variable because i wanted same ratio of 'yes' and 'no' in both test and train data. If i don't do this there may be chance that almost 90% of yes's end up in training data and that will cause problem in prediction.

```
set.seed(1)
trainIndex <- createDataPartition(full_dataset$Private, p = 0.70,list =
FALSE,times = 1)
train_data <- full_dataset[trainIndex,]
test_data <- full_dataset[-trainIndex,]
dim(train_data)
```

```
## [1] 545  18
```

```
dim(test_data)
```

```
## [1] 232  18
```

Performing Linear Regression:

```
linear_model <- lm(Apps~., data=train_data)
linear_model_summary <- summary(linear_model)
linear_model_summary
```

```
##
## Call:
## lm(formula = Apps ~ ., data = train_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.075575 -0.009327 -0.000644  0.007565  0.149997
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.001864   0.008520  -0.219  0.82693
## PrivateYes  -0.010525   0.003528  -2.983  0.00298 **
## Accept       0.895154   0.025818  34.672  < 2e-16 ***
## Enroll      -0.120243   0.028710  -4.188 3.30e-05 ***
## Top10perc    0.102080   0.012988   7.859 2.19e-14 ***
## Top25perc   -0.025074   0.010527  -2.382  0.01758 *
## F.Undergrad  0.043475   0.025439   1.709  0.08804 .
## P.Undergrad  0.003654   0.023704   0.154  0.87754
## Outstate    -0.039259   0.009522  -4.123 4.35e-05 ***
## Room.Board   0.019466   0.007895   2.466  0.01399 *
## Books       -0.004339   0.013342  -0.325  0.74516
## Personal     0.002123   0.010631   0.200  0.84180
## PhD         -0.018637   0.011105  -1.678  0.09387 .
## Terminal    -0.004329   0.009806  -0.441  0.65905
## S.F.Ratio    0.008930   0.013197   0.677  0.49893
## perc.alumni  0.002854   0.006840   0.417  0.67668
## Expend       0.080301   0.015723   5.107 4.58e-07 ***
## Grad.Rate    0.018116   0.008144   2.225  0.02654 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02267 on 527 degrees of freedom
## Multiple R-squared:  0.9327, Adjusted R-squared:  0.9305
## F-statistic: 429.4 on 17 and 527 DF,  p-value: < 2.2e-16
```

Residual standard error: 0.02267. Here, Accept, enroll , Top10perc and Outstate are important features.

Predicting linear model for test data:

```
linear_model_predict <- predict(linear_model , test_data , type="response")
MSE_linear_model <- mean((test_data$Apps - linear_model_predict )^2) ## Mean
Squared Error for test set
MSE_linear_model ## MSE is 0.0003986848

## [1] 0.0003986848
```

Mean Squared Testing error came out to be 0.0003986848.

Ans b) Ridge Regression:

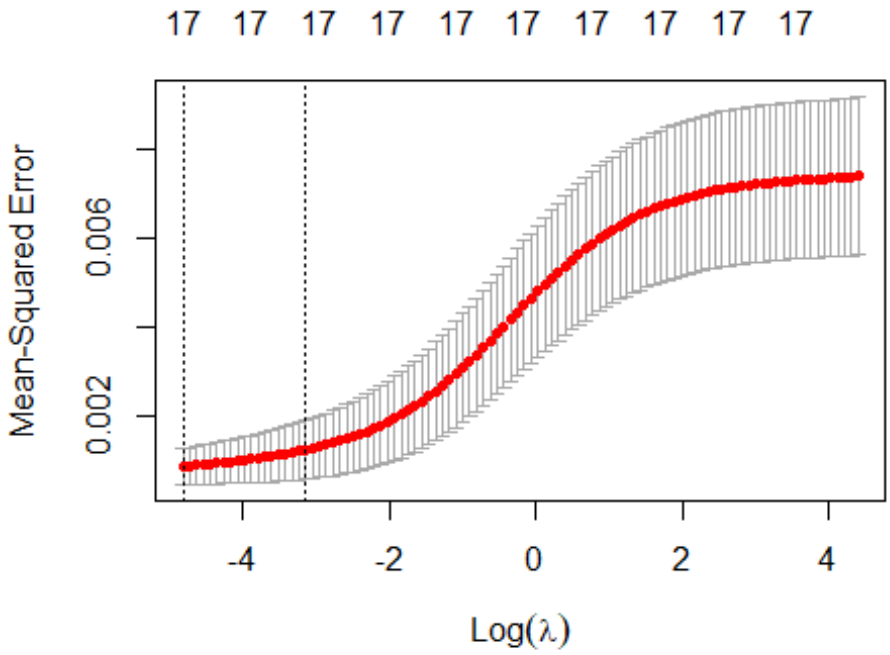
Before we start with ridge regression, we need to split test and training data according to glmnet() method.

```
set.seed(1)
train.x <- model.matrix(Apps~., train_data ) [,-1]
train.y <-  train_data$Apps
test.x <- model.matrix(Apps~.,test_data) [,-1]
test.y <- test_data$Apps
```

converting the data frames into matrix will convert the categorical variables into numeric variables.test.y and train.y stores the actual y value i.e. the target values.

Before we start with the ridge regression, we need to find the optimal lambda which is the penalty term for the ridge regression. This can be done by cross-validation.

```
set.seed (1)
cross_valid_output <- cv.glmnet(train.x,train.y,alpha =0)
plot(cross_valid_output)
```




```
bestlam <- cross_valid_output$lambda.min
bestlam

## [1] 0.008117708
```

So the best lambda value (penalty value) is 0.008117708.

Performing Ridge Regression:

```
ridge_analysis <- glmnet (train.x, train.y, alpha = 0,lambda = bestlam)

ridge_analysis_summary <- summary(ridge_analysis)
ridge_analysis_summary

##           Length Class     Mode
## a0         1     -none-    numeric
## beta      17     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       5     -none-    call
## nobs       1     -none-    numeric
```

Fitting the model to test set:

```
ridge.pred <- predict(ridge_analysis , s = bestlam ,newx = test.x ) ##Test
head(ridge.pred)
```

```
##                                                 s1
## Alderson-Broaddus College                0.009374106
## Allegheny College                        0.056870967
## Allentown Coll. of St. Francis de Sales 0.031873971
## Alma College                             0.037199868
## Amherst College                          0.073860739
## Andrews University                       0.007668084
```

Mean - Squared Test Error:

```
MSE_ridge_regression <- mean((ridge.pred - test.y )^2)
MSE_ridge_regression
```

```
## [1] 0.0004046323
```

Residual Sum of Square:

```
RSS_ridge_regression <- sum((ridge.pred - test.y)^2)
RSS_ridge_regression
```

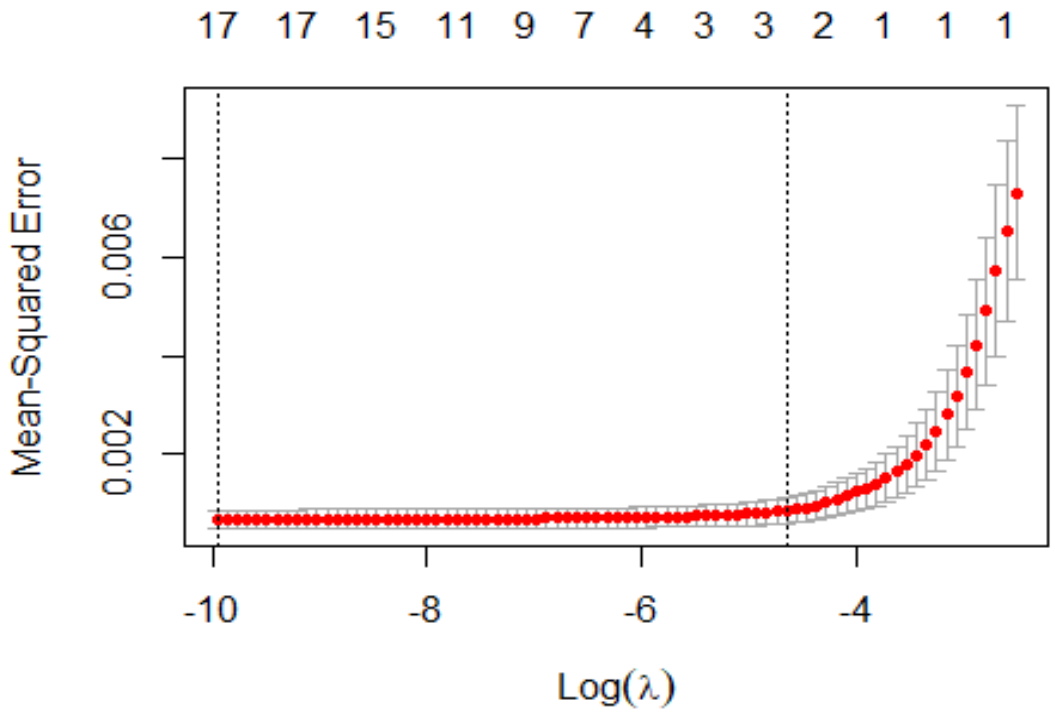
```
## [1] 0.09387468
```

So Mean Squared test error and Residual Sum of Square for test data came out to be 0.0004046323 and 0.09387468 respectively.

Ans c) Lasso Regression: Here also, just like ridge regression, first we need to find the penalty variable. Here instead of L2 regularization we use L1 regularization.

The other difference between ridge and lasso regression is ridge try to bring the non correlated variables close to zero but never sets the value of coefficient to absolute zero whereas lasso regression tends to make coefficients to absolute zero and remove them.

```
set.seed(1)
cross_valid_output_lasso <- cv.glmnet(train.x,train.y,alpha =1)
plot(cross_valid_output_lasso)
```




```
bestlam_lasso <- cross_valid_output_lasso$lambda.min
bestlam_lasso

## [1] 4.754559e-05
```

Performing Lasso Regression:

```
lasso_analysis <- glmnet (train.x, train.y, alpha = 1,lambda = bestlam_lasso)

lasso_analysis_summary <- summary(lasso_analysis)
lasso_analysis_summary

##           Length Class     Mode
## a0         1     -none-    numeric
## beta      17     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       5     -none-    call
## nobs       1     -none-    numeric
```

```
lasso_analysis$beta
```

```
## 17 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## PrivateYes  -0.010425516
## Accept       0.886749752
## Enroll      -0.105113172
## Top10perc    0.098585091
## Top25perc   -0.022252755
## F.Undergrad  0.033912198
## P.Undergrad  0.003356053
## Outstate    -0.038028685
## Room.Board   0.018987223
## Books       -0.003411501
## Personal     0.001531970
## PhD         -0.017991793
## Terminal    -0.004214277
## S.F.Ratio    0.007849825
## perc.alumni  0.001599007
## Expend       0.079464048
## Grad.Rate    0.017594142
```

Fitting the model to test set now:

```
lasso.pred <- predict(lasso_analysis, s = bestlam_lasso , newx = test.x )
head(lasso.pred)
```

```
##                                              s1
## Alderson-Broaddus College              0.01150574
## Allegheny College                      0.06003785
## Allentown Coll. of St. Francis de Sales 0.03427472
## Alma College                           0.03875999
## Amherst College                        0.07782797
## Andrews University                     0.01073702
```

Mean Squared test error for lasso regression:

```
MSE_lasso_regression <- mean((lasso.pred - test.y )^2)
MSE_lasso_regression
```

```
## [1] 0.0003982019
```

Residual sum of squared for test data in lasso regression:

```
RSS_lasso_regression <- sum((lasso.pred - test.y)^2)
RSS_lasso_regression
```

```
## [1] 0.09238284
```

Now one of the main properties of lasso regression is to make coefficients to absolute zero and remove them. Let's check what all features are zero and non-zeros.

```
coef(lasso_analysis)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -0.002239857
## PrivateYes  -0.010425516
## Accept       0.886749752
## Enroll      -0.105113172
## Top10perc    0.098585091
## Top25perc   -0.022252755
## F.Undergrad  0.033912198
## P.Undergrad  0.003356053
## Outstate    -0.038028685
## Room.Board   0.018987223
## Books       -0.003411501
## Personal     0.001531970
## PhD         -0.017991793
## Terminal    -0.004214277
## S.F.Ratio    0.007849825
## perc.alumni  0.001599007
## Expend       0.079464048
## Grad.Rate    0.017594142
```

This shows that non of the features are zero. I'll try some other method to check it.

```
outcome <- coef(lasso_analysis, s=bestlam_lasso)
outcome
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -0.002239857
## PrivateYes  -0.010425516
## Accept       0.886749752
## Enroll      -0.105113172
## Top10perc    0.098585091
## Top25perc   -0.022252755
## F.Undergrad  0.033912198
## P.Undergrad  0.003356053
## Outstate    -0.038028685
## Room.Board   0.018987223
## Books       -0.003411501
## Personal     0.001531970
## PhD         -0.017991793
## Terminal    -0.004214277
## S.F.Ratio    0.007849825
## perc.alumni  0.001599007
## Expend       0.079464048
## Grad.Rate    0.017594142
```

```
outcome[outcome[,1]!=0,]
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) -0.002239857
## PrivateYes  -0.010425516
## Accept       0.886749752
## Enroll      -0.105113172
## Top10perc    0.098585091
## Top25perc   -0.022252755
## F.Undergrad  0.033912198
## P.Undergrad  0.003356053
## Outstate    -0.038028685
## Room.Board   0.018987223
## Books       -0.003411501
## Personal     0.001531970
## PhD         -0.017991793
## Terminal    -0.004214277
## S.F.Ratio    0.007849825
## perc.alumni  0.001599007
## Expend       0.079464048
## Grad.Rate    0.017594142
```

This also don't show us any feature to be zero so for this data all the features are non-zeroes.

Ans D)

Citations:

https://stackoverflow.com/questions/17200114/how-to-split-data-into-training-testing-sets-using-sample-function

https://www.listendata.com/2015/02/splitting-data-into-training-and-test.html

https://discuss.analyticsvidhya.com/t/how-can-i-check-whether-my-data-frame-contains-na-inf-values-in-some-column-or-not-in-r/1647