Computer lab 2 block 1

Group A7

2022-12-05

Group members:Hamza (hammu144), Dinesh (dinsu875) and Umamaheswarababu (umama339)

Statement of Contribution : Assignment 1 was mostly done by Dinesh, Assignment 2 was mostly done by Umamaheswarababu and Assignment 3 was mostly done by Hamza

Assignment 1. Explicit regularization

Divide data randomly into train and test (50/50)

```
#Reading the data
data<-read.csv("tecator.csv")

#Dividing the data(50/50)
n<-dim(data)[1]
set.seed(12345)
id<-sample(1:n,floor(n*0.5))
train<-data[id,]
test<-data[-id,]

train_data <- train[,c(2:102)]
test_data <- test[,c(2:102)]
```

Question 1: Assume that Fat can be modeled as a linear regression in which absorbance characteristics (Channels) are used as features. Report the underlying probabilistic model, fit the linear regression to the training data and estimate the training and test errors. Comment on the quality of fit and prediction and therefore on the quality of model.

```
#Fit the linear regression model
mod = lm(Fat ~ ., data=train_data)
summary(mod)
```

```
##
## Call:
## lm(formula = Fat ~ ., data = train_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.201500 -0.041315 -0.001041  0.037636  0.187860
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.815e+01  5.488e+00  -3.306  0.01628 *
## Channel1     2.653e+04  1.126e+04   2.357  0.05649 .
## Channel2    -5.871e+04  3.493e+04  -1.681  0.14385
## Channel3     1.154e+05  7.373e+04   1.565  0.16852
## Channel4    -2.432e+05  1.175e+05  -2.070  0.08387 .
## Channel5     3.026e+05  1.193e+05   2.536  0.04430 *
## Channel6    -2.365e+05  8.160e+04  -2.898  0.02741 *
## Channel7     1.090e+05  3.169e+04   3.440  0.01380 *
## Channel8    -6.054e+04  1.508e+04  -4.015  0.00700 **
## Channel9     7.871e+04  2.160e+04   3.643  0.01079 *
## Channel10   -1.730e+04  1.640e+04  -1.055  0.33215
## Channel11    9.562e+04  3.529e+04   2.710  0.03512 *
## Channel12   -2.114e+05  6.198e+04  -3.410  0.01431 *
## Channel13    9.725e+04  4.424e+04   2.198  0.07026 .
## Channel14    5.296e+04  4.666e+04   1.135  0.29968
## Channel15   -7.855e+04  5.245e+04  -1.498  0.18491
## Channel16   -8.209e+03  1.893e+04  -0.434  0.67969
## Channel17    3.769e+04  1.987e+04   1.897  0.10666
## Channel18    3.306e+04  7.934e+03   4.167  0.00590 **
## Channel19   -8.405e+04  1.929e+04  -4.358  0.00478 **
## Channel20    1.510e+05  3.361e+04   4.492  0.00414 **
## Channel21   -2.069e+05  4.256e+04  -4.862  0.00282 **
## Channel22    1.348e+05  3.824e+04   3.526  0.01243 *
## Channel23   -4.094e+04  3.546e+04  -1.154  0.29222
## Channel24    2.023e+04  2.761e+04   0.733  0.49134
## Channel25    3.269e+03  1.071e+04   0.305  0.77045
## Channel26   -1.297e+04  7.636e+03  -1.699  0.14028
## Channel27    4.131e+03  1.422e+04   0.291  0.78120
## Channel28   -4.548e+03  2.988e+04  -0.152  0.88402
## Channel29    1.089e+04  1.768e+04   0.616  0.56072
## Channel30   -7.985e+04  2.653e+04  -3.010  0.02371 *
## Channel31    1.756e+05  5.279e+04   3.326  0.01589 *
## Channel32   -1.107e+05  2.904e+04  -3.813  0.00883 **
## Channel33   -6.525e+04  5.407e+04  -1.207  0.27294
## Channel34    1.007e+05  6.589e+04   1.528  0.17738
## Channel35   -2.841e+03  1.214e+04  -0.234  0.82266
## Channel36   -2.268e+04  2.295e+04  -0.988  0.36127
## Channel37   -4.479e+04  1.292e+04  -3.468  0.01334 *
## Channel38    3.209e+04  1.843e+04   1.742  0.13221
## Channel39    1.992e+04  2.067e+04   0.964  0.37246
## Channel40   -9.833e+03  2.431e+04  -0.404  0.69988
## Channel41    1.659e+04  3.648e+04   0.455  0.66531
```

```
## Channel42   -1.829e+04  3.528e+04  -0.519  0.62260
## Channel43   -2.423e+04  2.427e+04  -0.998  0.35669
## Channel44    3.246e+04  2.013e+04   1.613  0.15793
## Channel45   -8.089e+03  4.023e+04  -0.201  0.84728
## Channel46    7.065e+03  2.810e+04   0.251  0.80990
## Channel47   -4.062e+04  1.007e+04  -4.034  0.00685 **
## Channel48    9.080e+04  2.618e+04   3.469  0.01332 *
## Channel49   -6.647e+04  2.372e+04  -2.803  0.03105 *
## Channel50   -4.196e+04  2.856e+04  -1.469  0.19213
## Channel51    1.097e+05  5.572e+04   1.968  0.09661 .
## Channel52   -1.148e+05  6.376e+04  -1.800  0.12196
## Channel53    9.525e+04  7.450e+04   1.278  0.24830
## Channel54   -4.534e+04  7.363e+04  -0.616  0.56067
## Channel55   -1.535e+03  4.933e+04  -0.031  0.97618
## Channel56   -2.377e+03  2.109e+04  -0.113  0.91394
## Channel57    3.174e+04  1.005e+04   3.158  0.01961 *
## Channel58    2.221e+03  1.048e+04   0.212  0.83915
## Channel59   -8.504e+04  2.574e+04  -3.304  0.01634 *
## Channel60    6.382e+04  1.607e+04   3.972  0.00735 **
## Channel61    2.151e+04  1.234e+04   1.742  0.13211
## Channel62   -2.859e+04  1.065e+04  -2.685  0.03631 *
## Channel63    1.796e+04  9.187e+03   1.955  0.09838 .
## Channel64    5.759e+04  3.526e+04   1.633  0.15354
## Channel65   -1.470e+05  6.911e+04  -2.127  0.07752 .
## Channel66    9.121e+04  4.461e+04   2.045  0.08688 .
## Channel67   -5.733e+03  2.197e+04  -0.261  0.80288
## Channel68   -6.290e+04  2.192e+04  -2.870  0.02843 *
## Channel69    6.421e+04  2.074e+04   3.096  0.02121 *
## Channel70   -1.749e+04  1.581e+04  -1.106  0.31111
## Channel71   -7.248e+03  1.934e+04  -0.375  0.72075
## Channel72    3.406e+04  1.185e+04   2.873  0.02830 *
## Channel73   -2.100e+04  1.132e+04  -1.855  0.11308
## Channel74   -3.314e+04  1.220e+04  -2.717  0.03480 *
## Channel75    7.039e+04  2.054e+04   3.427  0.01402 *
## Channel76   -3.187e+04  1.736e+04  -1.836  0.11597
## Channel77    2.061e+04  1.810e+04   1.138  0.29832
## Channel78   -1.180e+04  2.273e+04  -0.519  0.62225
## Channel79    2.669e+04  2.997e+04   0.890  0.40750
## Channel80   -6.051e+04  1.483e+04  -4.080  0.00650 **
## Channel81    1.386e+03  2.628e+04   0.053  0.95966
## Channel82    1.020e+05  4.694e+04   2.173  0.07275 .
## Channel83   -1.706e+05  4.688e+04  -3.640  0.01083 *
## Channel84    1.097e+05  2.892e+04   3.792  0.00905 **
## Channel85   -1.294e+05  3.600e+04  -3.594  0.01145 *
## Channel86    2.130e+05  4.345e+04   4.903  0.00270 **
## Channel87   -1.198e+05  3.818e+04  -3.139  0.02011 *
## Channel88   -2.199e+04  6.085e+04  -0.361  0.73021
## Channel89    7.974e+04  5.077e+04   1.571  0.16733
## Channel90   -1.711e+05  5.499e+04  -3.112  0.02079 *
## Channel91    2.107e+05  6.406e+04   3.289  0.01663 *
## Channel92   -1.959e+05  7.171e+04  -2.733  0.03407 *
## Channel93    2.874e+05  9.937e+04   2.892  0.02762 *
```

```
## Channel94   -3.064e+05  9.601e+04  -3.191  0.01881 *
## Channel95    2.048e+05  6.220e+04   3.292  0.01656 *
## Channel96   -5.600e+04  2.929e+04  -1.912  0.10441
## Channel97   -1.318e+04  3.050e+04  -0.432  0.68065
## Channel98   -2.724e+04  2.107e+04  -1.292  0.24375
## Channel99    3.556e+04  1.382e+04   2.573  0.04218 *
## Channel100  -1.206e+04  4.264e+03  -2.828  0.03006 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3191 on 6 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:  0.9994
## F-statistic:  1651 on 100 and 6 DF,  p-value: 1.058e-09
```

```
##MSE Train & Test
train_mse <- mean(mod$residuals^2)
test_mse <- mean((test_data$Fat - predict(mod,test_data))^2)

list("Train MSEs"=train_mse,"Test MSEs"=test_mse)
```

```
## $`Train MSEs`
## [1] 0.005709117
##
## $`Test MSEs`
## [1] 722.4294
```

In Linear Regression model, Multiple-R-squared value indicates how much variation is captured by the model. Here, Multiple-R-squared is 1 says that the model explains the larger value of the variance of the model. Hence, the model is good fit.Based on the summary we can find the significant predictor values. From the summary we can see that some of the variables are less significant features as the ‘p’ value is large for them.

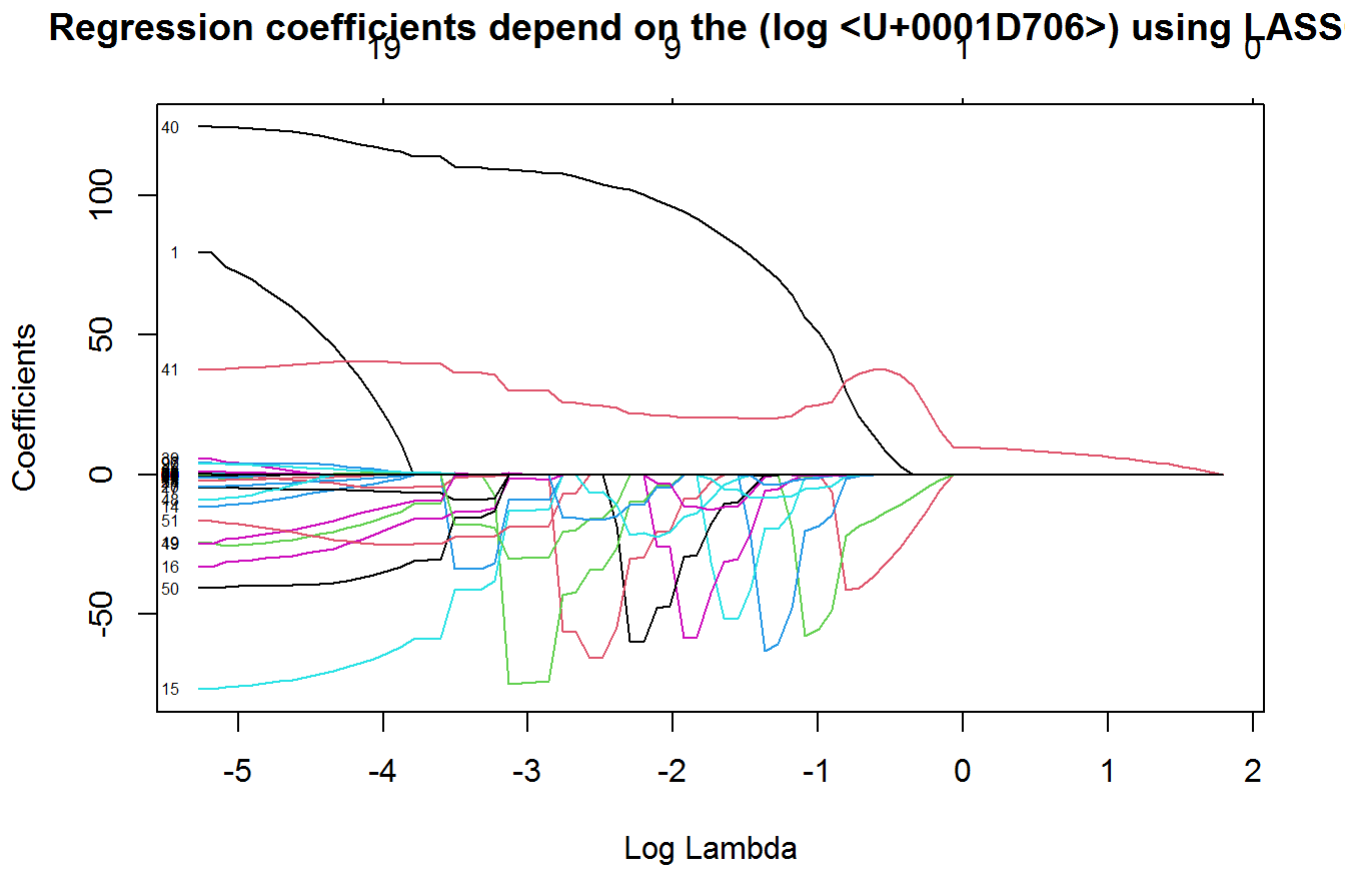
Question 2: Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features. Report the cost function that should be optimized in this scenario.

$$argmin[\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i\theta_j)^2 + \lambda\sum_{j=1}^{p}|\theta_j|]$$

```
#Cost function
lasso_cost <- function(theta, lambda){
  y <- train_data[,101]
  X <-as.matrix( train_data[,1:100] )
  n<-nrow(X)
  theta_new <- theta[1:100]
  return( (1/n)*( sum((y - X%*%theta_new)^2) +  lambda * sum(theta) ) )
}
```

Question 3: Fit the LASSO regression model to the training data. Present a plot illustrating how the regression coefficients depend on the log of penalty factor (logλ) and interpret this plot. What value of the penalty factor can be chosen if we want to select a model with only three features?

```
response <- train_data[,101]
covariates <- train_data[,c(1:100)]

library(glmnet)
Lasso_model <- glmnet(x=as.matrix(covariates),y=response , alpha=1,family="gaussian")
plot(Lasso_model, xvar="lambda", label=TRUE, main= "Regression coefficients depend on the (log
 λ) using LASSO")
```




```
s<-Lasso_model$df
t<-Lasso_model$lambda
a<-data.frame(df=s,lambda=t)

cat(paste("Optimal lambda : ", min(Lasso_model$lambda)))
```

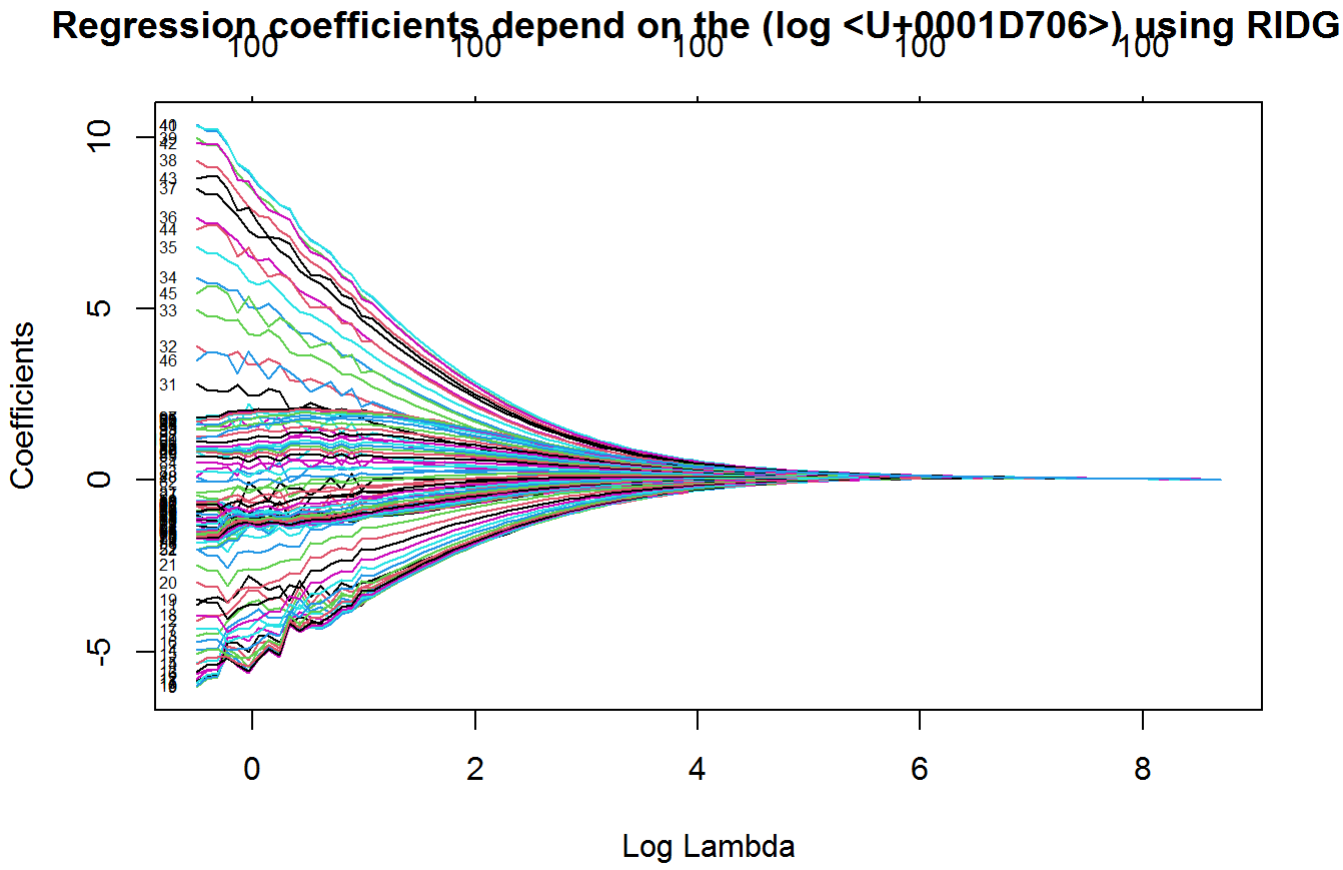
```
## Optimal lambda :  0.00511387151273923
```

In lasso regression, the shrinkage and also the variable selection is done. A different coefficient in the model is represented by a different colored line for each value (feature). It can be observed that as the log value of lambda increases, some of the coefficients get set to zero values. The variable that most influence the model is 41, because it enters the model first, and it also affects the response variable in a positive manner. The lasso displays a definite trend in the coefficients for log lambda values.

The penalty factor 0.853045182,0.777262999, 0.708213096 can be chosen if we want to select a model with only three features.

Question 4: Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4. Conclusions?

```
# 1.4 Fit a Ridge regression model

Ridge_model <- glmnet(x=as.matrix(covariates),y=response, alpha=0,family="gaussian")
plot(Ridge_model, xvar="lambda", label=TRUE, main = "Regression coefficients depend on the (log
 λ) using RIDGE")
```




```
cat(paste("Optimal lambda : ", min(Ridge_model$lambda)))
```

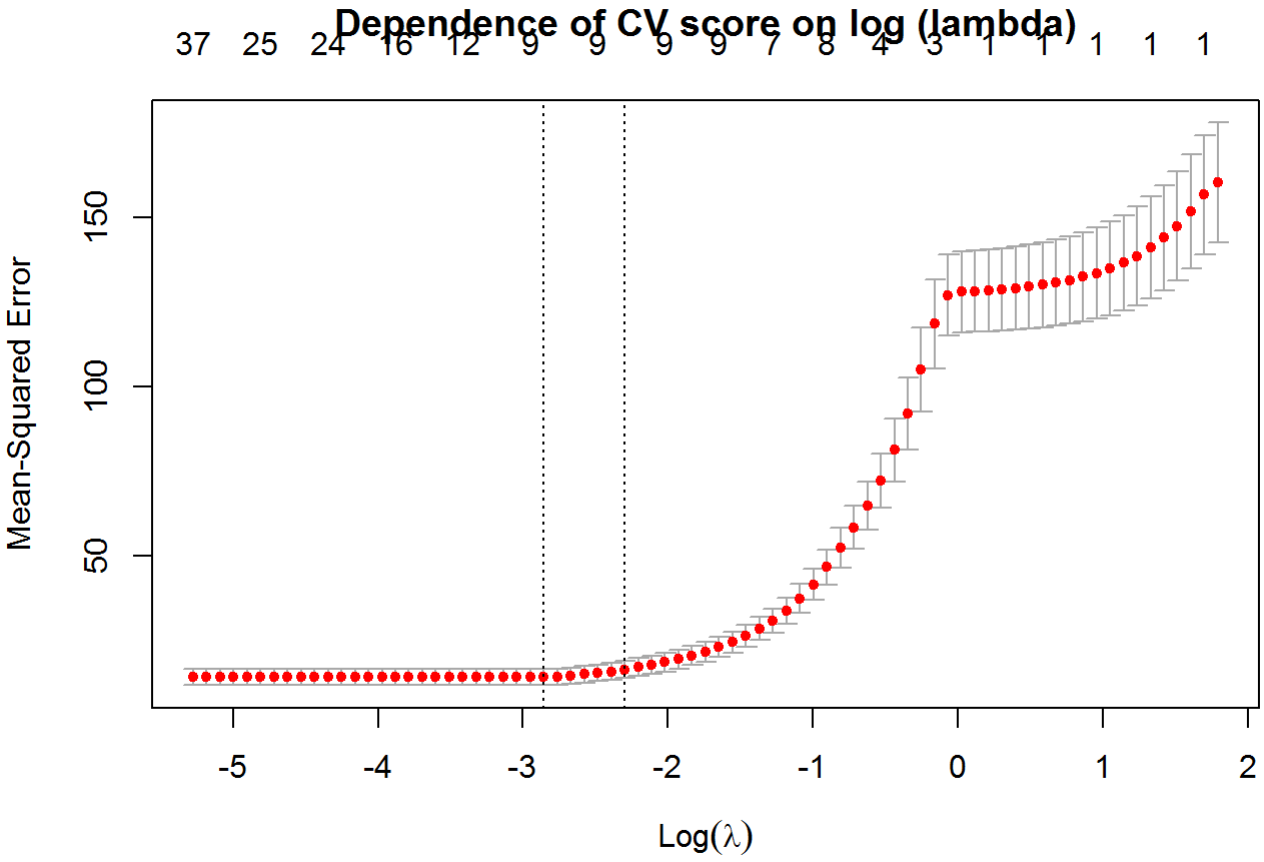
```
## Optimal lambda :  0.601806511940838
```

In ridge regression, only the beta coefficients are minimized (towards zero) by lambda (shrinkage term) but variable selection is not done. The coefficients get smaller as lambda gets bigger.When lambda effectively equals zero, the sum of squares of the coefficients has reached its maximum value. But in lasso regression, the shrinkage and also the variable selection is done. Both ridge regression and the LASSO regression have solutions and are indexed by the continuous parameter λ. Comparing with ridge, the beta values of LASSO with log λ employed have no closed form.

Question 5: Use cross-validation with default number of folds to compute the optimal LASSO model. Present a plot showing the dependence of the CV

score on log $\lambda$and comment how the CV score changes with log λ. Report the optimal $\lambda$and how many variables were chosen in this model. Does the information displayed in the plot suggests that the optimal λ value results in a statistically significantly better prediction than log $\lambda = -4$? Finally, create a scatter plot of the original test versus predicted test values for the model corresponding to optimal lambda and comment whether the model predictions are good.

```
cv_Lasso_model=cv.glmnet(x=as.matrix(covariates),y=response,alpha=1,family="gaussian")
plot(cv_Lasso_model, xvar="lambda", label=TRUE, main = "Dependence of CV score on log (lambda)")
```



```
#coef(cv_Lasso_model, s="lambda.min")

Optimal_lambda<-cv_Lasso_model$lambda.min
cat(paste("Optimal lambda : ", Optimal_lambda))
```

```
## Optimal lambda :  0.0574453477182168
```

```
cat(sep = '\n')
```

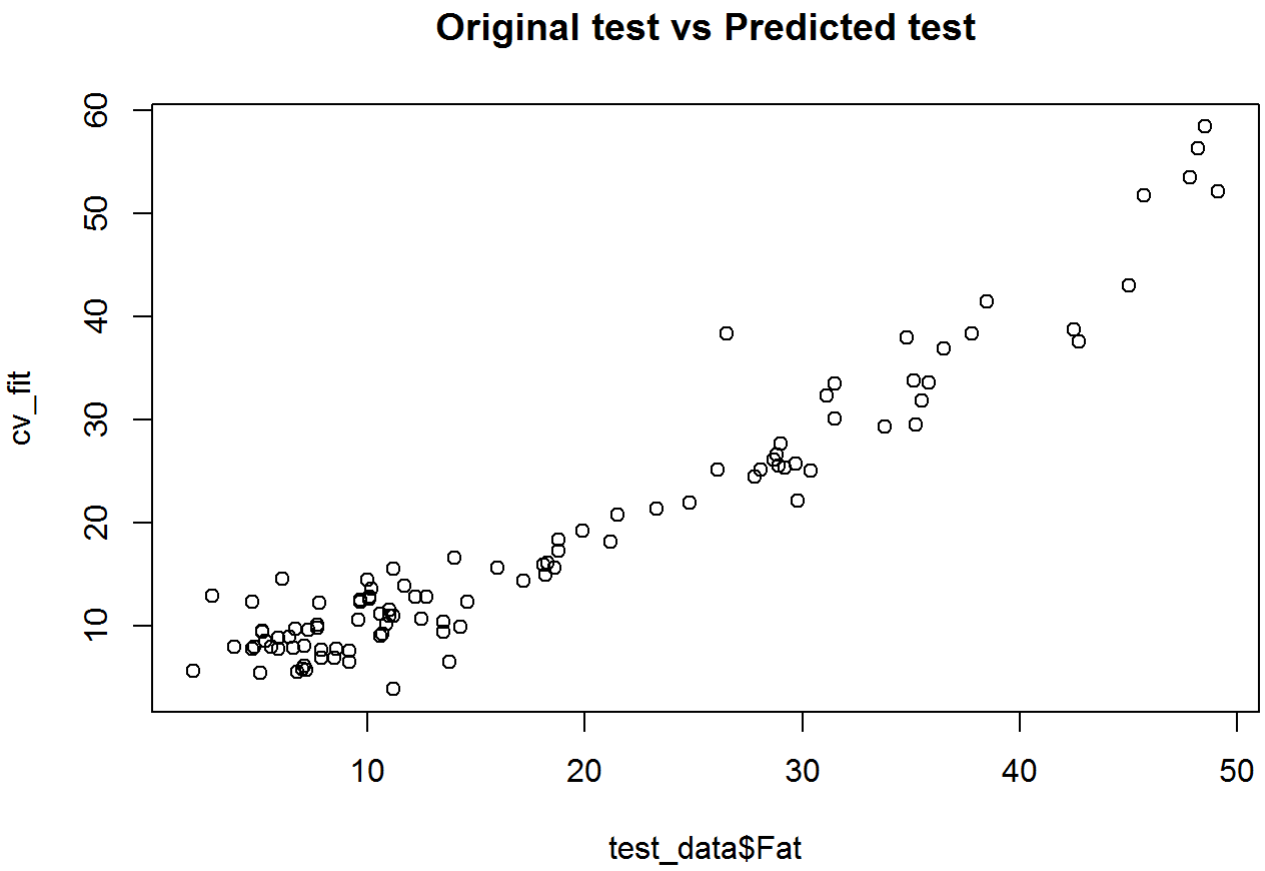
```
cat(paste("Number of variables chosen by the model : ",length(coef(cv_Lasso_model, s="lambda.mi
n")) - 1))
```

```
## Number of variables chosen by the model :  100
```

To find the optimal Lasso model we are using cross validation technique. Here, the red dotted represents the algorithm process for finding significant subset features, number of variables that have been chosen were all 100 variables. The MSE increases when the value of lambda increases.Initially it starts with the stable and increases in the range log lambda.Hence, optimal lambda shows no statistics difference with log lambda=-4.

create a scatter plot of the original test versus predicted test values for the model corresponding to optimal lambda

```
x_test_covariates <- test_data[,1:100]
cv_fit<-predict(cv_Lasso_model,as.matrix(x_test_covariates))

plot(x=test_data$Fat ,y= cv_fit, main = "Original test vs Predicted test")
```



A scatter plot is used to visualize the linear relationship between the dependent (response) variable and independent (predictor) variables. From the obtained scatter plot, we can see that the test value increases, as well as there is a gradual increase in predicted test. Therefore, in this plot, a best-fitting is clearly obtained through the points. Hence, the model predictions are good.

Assignment 2 : Decision trees and logistic regression for bank marketing

Q1.Import the data to R, remove variable "duration" and divide into training/validation/test as 40/30/30: use data partitioning code specified in Lecture 2a.

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v ggplot2 3.3.6      v dplyr   1.0.10
## v tibble  3.1.8      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## v purrr   0.3.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

```
data<-read.csv("bank-full.csv")
col_names<-colnames(data)
col_names<- unlist(str_split(col_names,"[.]"))
data1<-separate(data,1,col_names,sep = ";")

data1<-data1[,-6]  #removing "balance" which is not present in input variables

data1$age<-as.numeric(data1$age)
data1$job<-as.factor(data1$job)
data1$marital<-as.factor(data1$marital)
data1$education<-as.factor(data1$education)
data1$default<-as.factor(data1$default)
data1$housing<-as.factor(data1$housing)
data1$loan<-as.factor(data1$loan)
data1$contact<-as.factor(data1$contact)
data1$month<-as.factor(data1$month)
data1$day<-as.factor(data1$day)
data1$duration<-as.numeric(data1$duration)
data1$campaign<-as.numeric(data1$campaign)
data1$pdays<-as.numeric(data1$pdays)
data1$previous<-as.numeric(data1$previous)
data1$poutcome<-as.factor(data1$poutcome)
data1$y<-ifelse(data1$y=="yes",1,0) #1 means "yes", 0 means "no"
data1$y<-as.factor(data1$y)

#removing the column "duration" and loading the data
cleaned_data<- data1[ , -which(names(data1) %in% c("duration"))]

#splitting the data into training/validation/test
n=dim(cleaned_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=cleaned_data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=cleaned_data[id2,]
id3=setdiff(id1,id2)
test=cleaned_data[id3,]
```

Q2.Fit decision trees to the training data so that you change the default settings one by one (i.e. not simultaneously):

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.2.2
```

```
#a. Decision Tree with default settings.
tree_default<-tree(y~., data = train)

#b. Decision Tree with smallest allowed node size equal to 7000.
tree_smallest<-tree(y~.,data = train, minsize=7000)

#c. Decision trees minimum deviance to 0.0005.
tree_min_dev<-tree(y~., data = train,mindev=0.0005)


#predictions with training data
pred_train_default <- predict(tree_default, train, type = "class")
pred_train_smallest <- predict(tree_smallest, train, type="class")
pred_train_min_dev <- predict(tree_min_dev, train, type="class")

#prediction with validation data
pred_valid_default <- predict(tree_default, valid, type = "class")
pred_valid_smallest <- predict(tree_smallest, valid, type = "class")
pred_valid_min_dev <- predict(tree_min_dev, valid, type = "class")

#A function to calculate misclassification error
missclass<- function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#calculating misclassification rate on training data

d1<-missclass(train$y,pred_train_default)
s1<-missclass(train$y,pred_train_smallest)
m1<-missclass(train$y,pred_train_min_dev)

#calculating misclassification rate on validation data

d2<-missclass(valid$y,pred_valid_default)
s2<-missclass(valid$y,pred_valid_smallest)
m2<-missclass(valid$y,pred_valid_min_dev)


misclass_rate<- data.frame("Default"=c(d1,d2),"Smallest_Node"=c(s1,s2),"Min_dev"=c(m1,m2))
rownames(misclass_rate)<-c("Train","Valid")
```

Misclassification rates are given below

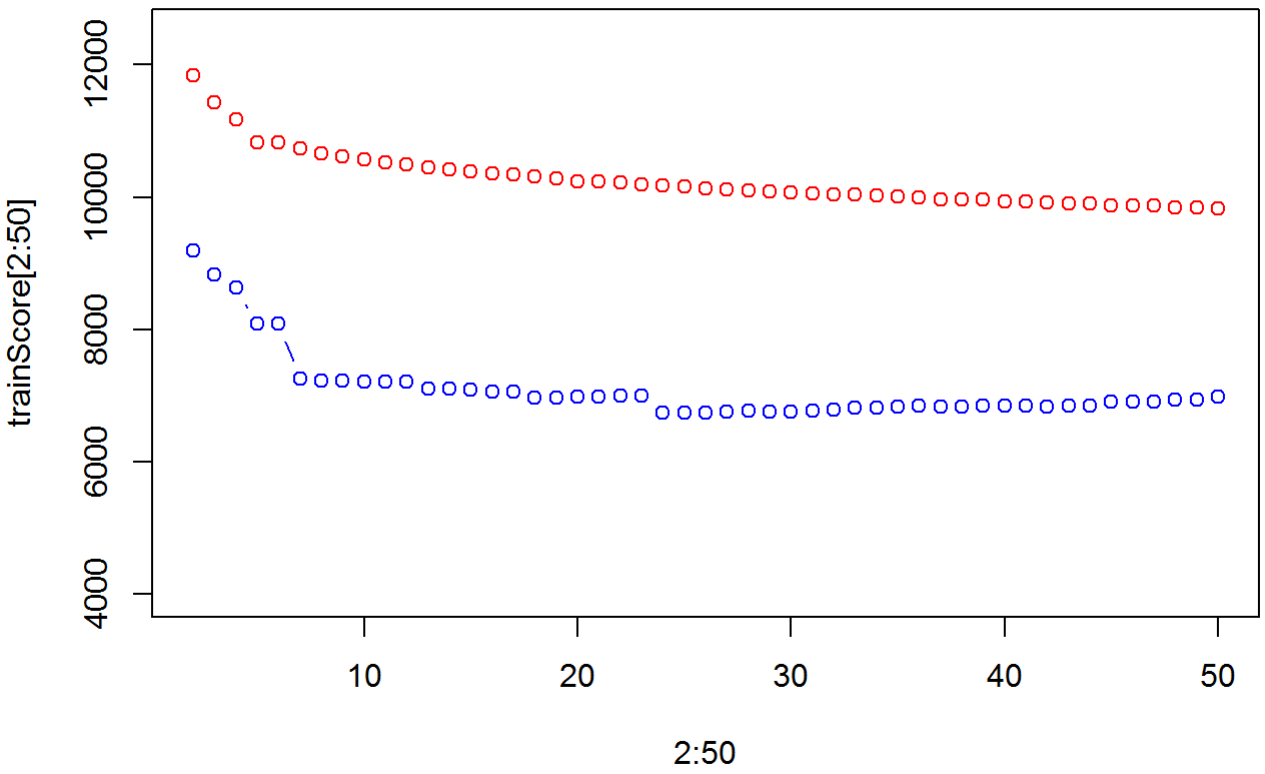
```
misclass_rate
```

```
##         Default Smallest_Node    Min_dev
## Train 0.1048441     0.1048441 0.09544349
## Valid 0.1092679     0.1092679 0.11398658
```

From the above results we can say that there is no much difference between misclassification rates of two data sets(train & valid). We can observe that the model with minimum deviance 0.0005 is classifying comparatively better than the models with default settings and smallest node is equal to 7000.

Q3.Use training and validation sets to choose the optimal tree depth in the model 2c: study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and the validation data on the number of leaves and interpret this graph in terms of bias-variance tradeoff. Report the optimal amount of leaves and which variables seem to be most important for decision making in this tree. Interpret the information provided by the tree structure (not everything but most important findings).

```
#decision trees minimum deviance to 0.0005.
tree_min_dev<-tree(y~., data = train,mindev=0.0005)

trainScore <- rep(0,50)
validScore <- rep(0,50)

for(i in 2:50) {
  prunedTrain=prune.tree(tree_min_dev,best=i)
  pred_train=predict(prunedTrain, newdata=valid, type="tree")
  trainScore[i]=deviance(prunedTrain)
  validScore[i]=deviance(pred_train)
}
plot(2:50, trainScore[2:50], type="b", col="red", ylim=c(4000,12500))
points(2:50, validScore[2:50], type="b", col="blue")
```



The above graph shows the dependence of deviances for the training and the validation data on the number of leaves for the model with minimum deviance equal to 0.0005. If the number of leaves are less(simple model) then the model has high bias and low variance(underfit) and if the number of leaves are model(more complex model) then the model has low bias and high variance(overfit)

Optimal amount of leaves is given by

```
optimal_leaves <-which.min(validScore[-1])
optimal_leaves
```

```
## [1] 23
```

This means this model with 23 leaves gives the least deviance when evaluated with validation data.

The most important information provided by the model and most important variables that are used to make decision making are given below

```
optimal_tree <- prune.tree(tree_min_dev,best = optimal_leaves)
summary(optimal_tree)
```

```
##
## Classification tree:
## snip.tree(tree = tree_min_dev, nodes = c(154L, 298L, 288L, 79L,
## 305L, 11L, 75L, 6L, 304L, 10L, 7L, 289L, 153L, 598L, 78L, 148L,
## 16L))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "day"      "age"      "pdays"    "job"
## [8] "campaign" "housing"
## Number of terminal nodes:  23
## Residual mean deviance:  0.5644 = 10190 / 18060
## Misclassification error rate: 0.1027 = 1857 / 18084
```

The following variables seem to be most important for decision making in the tree.

“poutcome” “month” “contact” “day” “age” “pdays” “job” “campaign” “housing”

Q4. Estimate the confusion matrix, accuracy and F1 score for the test data by using the optimal model from step 3. Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

```
#prediction with test data
pred_test <-predict(optimal_tree, test, type = "class")
```

Confusion matrix is given by

```
#confusion matrix
t<-table(true=test$y,pred_test)
t
```

```
##     pred_test
## true     0     1
##    0 11782   197
##    1  1251   334
```

Accuracy is given by

```
#accuracy
1-missclass(test$y,pred_test)
```

```
## [1] 0.8932468
```

The accuracy of the model with test data is 89.32%

F1 score is given by $F1 = \frac{2.precision.recall}{precision+recall}$

```
recall <-t[2,2]/sum(t[,2])
precision<- t[2,2]/(sum(t[2,]))
F1<-2*precision*recall/(precision+recall)
F1
```

```
## [1] 0.31569
```

Since the data has uneven class distribution, F1 score has to be considered to determine the predictive power of the model. The F1 score of our model is 0.315 which means our classifier has high number of false positives. From this we can say that our model has no good predictive power.

Q5. Perform a decision tree classification of the test data with the following loss matrix, and report the confusion matrix for the test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.2.2
```

```
loss_matrix <-matrix(c(0,5,1,0), nrow = 2)
tree_test<-rpart(y~., data=train, method="class",parms=list(loss=loss_matrix))
loss_mat_tree <-predict(tree_test, test, type = "class")
#Reference:https://stackoverflow.com/questions/49646377/loss-matrix-in-rs-package-rpart
```

The confusion matrix is given by

```
t_loss<-table(true=test$y,predicted=loss_mat_tree)
t_loss
```

```
##     predicted
## true     0     1
##    0 10910  1069
##    1   836   749
```

Accuracy is given by

```
#accuracy
1-missclass(test$y,loss_mat_tree)
```

```
## [1] 0.8595547
```

The accuracy of this model is 85.95%

F1 score is given by

```
recall_loss <-t_loss[2,2]/sum(t_loss[,2])
precision_loss<- t_loss[2,2]/(sum(t_loss[2,]))
F1_loss<-2*precision_loss*recall_loss/(precision_loss+recall_loss)
F1_loss
```

```
## [1] 0.4401998
```

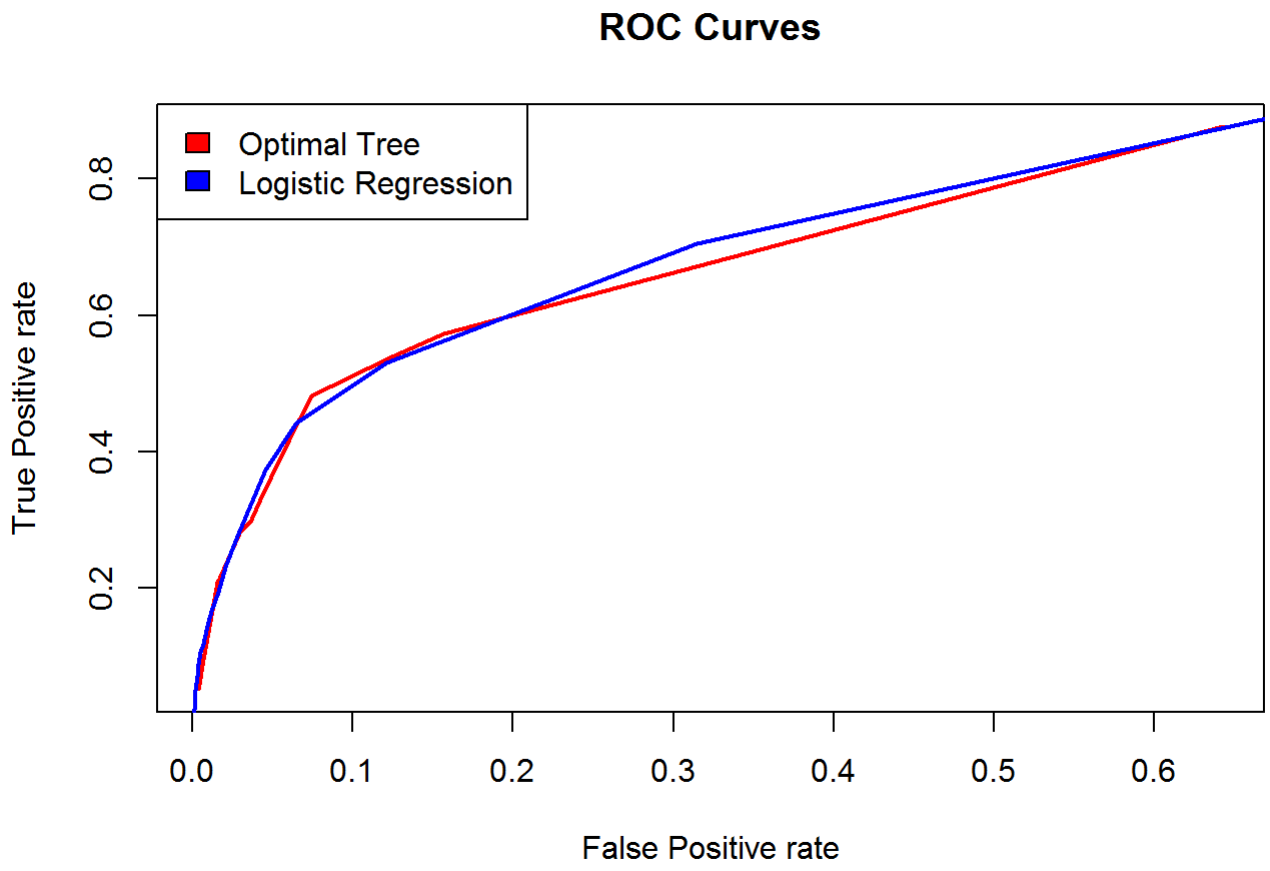
In comparison with the results in step 4, the accuracy of the model has come down to 85.95% and F1 score is increased to 0.440 from 0.315. Here a loss matrix is used to weight the misclassifications to penalize each misclassification. We penalize false positives five times more than the false negatives which changes the way of splitting such that loss of making incorrect predictions is reduced.

Q6. Use the optimal tree and a logistic regression model to classify the test data by using the following principle: Y_hat= yes if p(Y=yes|X)>π,otherwise Y_hat = no where π=0.05,0.1,0.15, … 0.9,0.95. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion? Why precision recall curve could be a better option here?

```
#optimal tree
tree_final <- tree(y~., train, mindev=0.0005)
optimal_tree_final <-prune.tree(tree_final, best=23)
optimal_pred<-predict(optimal_tree_final, test, type = "vector")

#Logistic regression
logistic<- glm(y~., data=train, family = "binomial")
logistic_pred<-predict(logistic, test, type= "response")

#computing TPR and FPR values for different values of threshold(pi)
TPR_tree<-c()
FPR_tree<-c()
TPR_log<-c()
FPR_log<-c()

j<-1

for (i in seq(0.05,0.95,0.05)){
  temp_tree<-ifelse(optimal_pred[,2]>i,1,0)
  temp_log<-ifelse(logistic_pred>i,1,0)
  c1<-table(test$y,temp_tree)
  if(dim(c1)[2]>1){
    TPR_tree[j]<-c1[2,2]/sum(c1[2,])
    FPR_tree[j]<-c1[1,2]/sum(c1[1,])
  }
  c2<-table(test$y,temp_log)
  if(dim(c2)[2]>1){
    TPR_log[j]<-c2[2,2]/sum(c2[2,])
    FPR_log[j]<-c2[1,2]/sum(c2[1,])
  }
  j=j+1
}
```

Plotting ROC curves

```
plot(FPR_tree, TPR_tree, type = "l",col="red", lwd=2,xlab = "False Positive rate",ylab = "True P
ositive rate", main = "ROC Curves")
lines(FPR_log,TPR_log, type="l", col="blue",lwd=2)
legend("topleft",c("Optimal Tree","Logistic Regression"), fill=c("red","blue"))
```



we can observe that the area under the curve is more for logistic regression model than optimal tree model. So we can conclude that logistic regression is the best model in this case.

Precision-recall curve could be a better option here because the data has unbalanced classes.

Assignment 3 : Principal components and implicit regularization

Question-1

we load crime data first. Then scaled data. we scale it without ViolentCrimesPerPop(101 col) as mentioned. for (PCA) we have to maximize variance of projected data. for that we need sample covarince matrix. which is

$$S = \frac{1}{n}\mathbf{X}^T X$$

with the help of eigenvalue decomposition we will get optimal solution by maximizing lamda.

$$S\mu = \mu\lambda$$

After getting sample covariance matrix we need to find eigen values and eigen vector which we will find with the help of eigen() function.

In above matrix each value in diagonal represent eigenvalue. so we have to check for at least 95% of variance of data for that we will check each eigenvalue of matrix.

```
crime_data <- read.csv("communities.csv")
sacled_data <- scale(crime_data[ ,-101])
s_c_matrix <- cov(sacled_data )
eigen <- eigen(s_c_matrix)


e_values <- eigen$values

report <-  function(e_values){
  vs <- 0
  counter = 0
  for(i in e_values){

    vs = vs + (i / sum(e_values))
    counter = counter + 1

    if(vs > 0.95){
      break
    }

  }


   comp.1 = e_values[1]/sum(e_values)
   comp.2 = e_values[2]/sum(e_values)
  res <- list("Total componets" = counter,  "proportion of first componenet" = comp.1,
              "proportion of second componenet" = comp.2)

  return(res)
}


report(e_values)
```

```
## $`Total componets`
## [1] 35
##
## $`proportion of first componenet`
## [1] 0.2501699
##
## $`proportion of second componenet`
## [1] 0.1693597
```

Above list gives us total components having at least 95% variance and proportion for first two components we were asked for.

Question-2

Now we did analysis using princomp() function and we plot it. we use loading as it holds all the information for commutative and proportion variance which we used for analysis. we take first component as we asked in assignment. from plot we get variation of component one between (-0.15 to 0.15) we selected first five absoulte feature of comp.1

which are:

state:US

population:population for community

householdsize:mean people per household

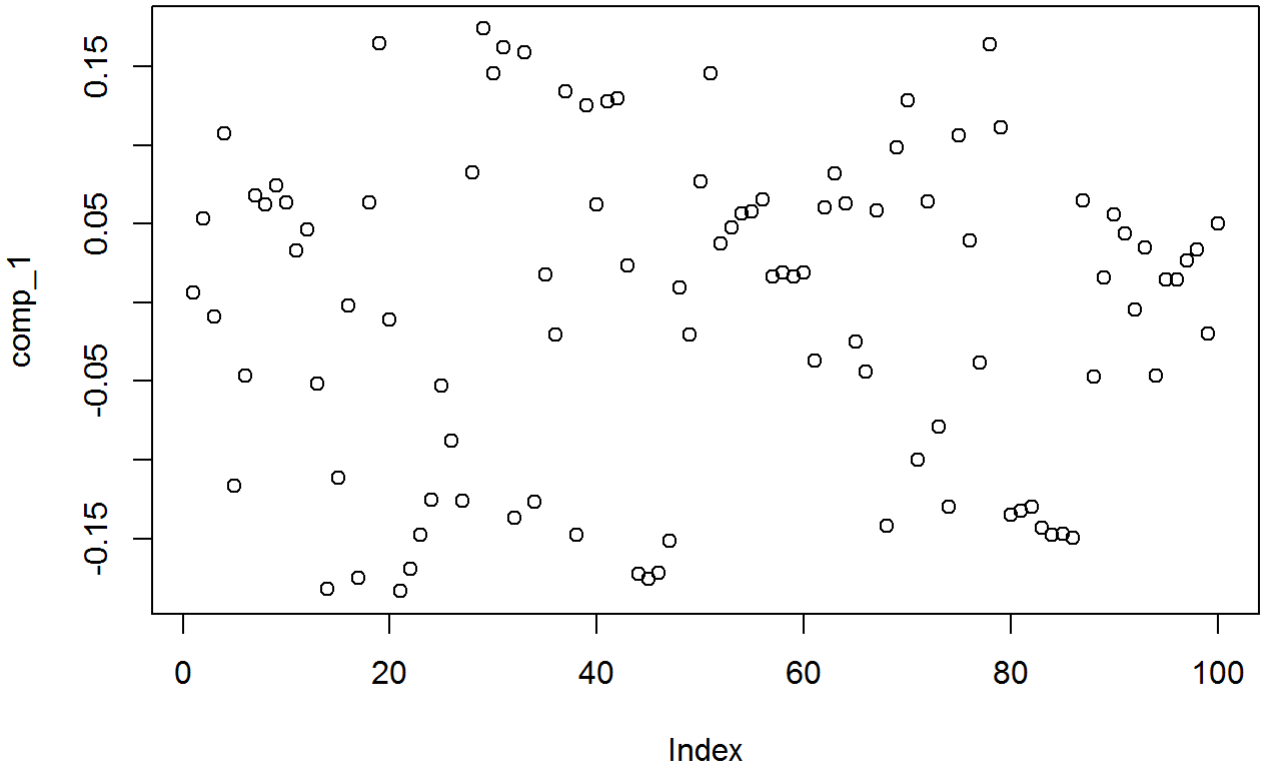
racepctblack:percentage of population that is african american

racePctWhite:percentage of population that is caucasian

These are attributes description which we collect from link. from above attributes of first component we can say that they have common thing in them is population because of increasing population and state not able to manage them as we can see that increase of population not ability of government to manage things leads to cause of crime. Maybe increase of crime is because of increase of population.

we get data frame to draw plot between first two component and compared their score with eachother.

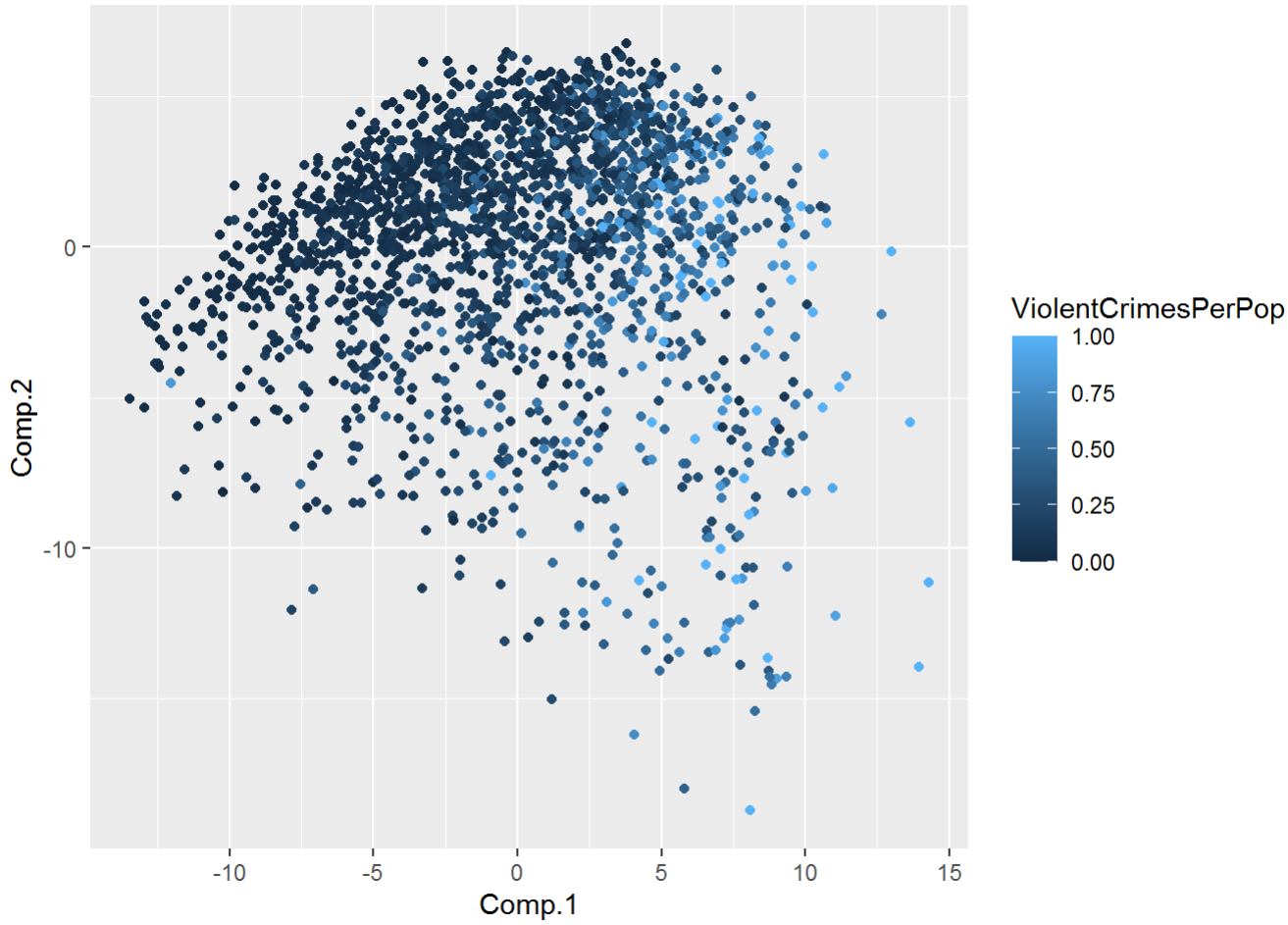
```
repaet_pca <- princomp(sacled_data)

comp_1 <- repaet_pca$loadings[, "Comp.1"]
plot(comp_1)
```




```
obs <- abs(comp_1)[1:5]
obs
```

```
##         state    population householdsize  racepctblack  racePctWhite
##   0.006204668   0.053178417   0.009010122   0.107496674   0.116370033
```

```
pc_score <- as.data.frame(repaet_pca$scores[, 1:2])
ViolentCrimesPerPop = crime_data[ ,101]


library(ggplot2)
ggplot(pc_score, aes(Comp.1, Comp.2))+
  geom_point(aes(color = ViolentCrimesPerPop))
```



from above plot we can see that values of component 2 is variation between -10 to 0 and variation of component 1 is -10 to 15 so we can say that high score of crime is in component 1 but on other hand crime score of second component is less.

Question-3

We have asked with dividing data into train and test then scale we did it with the help of set seed by fixing data and then scaled it.

```
n=dim(crime_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
test=crime_data[id,]
scale_test = as.data.frame(scale(test))

id1=setdiff(1:n, id)
id2=sample(id1, floor(n*0.5))
train=crime_data[id2,]
scale_train = as.data.frame(scale(train))

lm_m <- lm(ViolentCrimesPerPop ~ ., data = scale_train)

predicted_data <- predict(lm_m, scale_train)
actual_data <- crime_data[,"ViolentCrimesPerPop"]
n <- length(crime_data[,"ViolentCrimesPerPop"])
mse_train <- sum((actual_data - predicted_data) ^2)/n

predicted_data <- predict(lm_m, scale_test)
actual_data <- crime_data[,"ViolentCrimesPerPop"]
n <- length(crime_data[,"ViolentCrimesPerPop"])
mse_test <- sum((actual_data - predicted_data) ^2)/n

mse_test
```

```
## [1] 0.7965122
```

```
mse_train
```

```
## [1] 0.8348073
```

we used trained scaled data as we asked and took ViolentCrimesPerPop as target variable with the help of this we made a model and then we asked for MSE for both test and train which we calculated with the help of taking mean of square difference of actual and predicted data. above two values represent required results.

Question-4

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

#Reading the data
data<-read.csv("tecator.csv")

#Dividing the data(50/50)
n<-dim(data)[1]
set.seed(12345)
id<-sample(1:n,floor(n*0.5))
train<-data[id,]
test<-data[-id,]

train_data <- train[,c(2:102)]
test_data <- test[,c(2:102)]


#Fit the linear regression model
mod = lm(Fat ~ ., data=train_data)
summary(mod)

##MSE Train & Test
train_mse <- mean(mod$residuals^2)
test_mse <- mean((test_data$Fat - predict(mod,test_data))^2)

list("Train MSEs"=train_mse,"Test MSEs"=test_mse)

#Cost function
lasso_cost <- function(theta, lambda){
  y <- train_data[,101]
  X <-as.matrix( train_data[,1:100] )
  n<-nrow(X)
  theta_new <- theta[1:100]
  return( (1/n)*( sum((y - X%*%theta_new)^2) +  lambda * sum(theta) ) )
}

response <- train_data[,101]
covariates <- train_data[,c(1:100)]

library(glmnet)
Lasso_model <- glmnet(x=as.matrix(covariates),y=response , alpha=1,family="gaussian")
plot(Lasso_model, xvar="lambda", label=TRUE, main= "Regression coefficients depend on the (log
 λ) using LASSO")

s<-Lasso_model$df
t<-Lasso_model$lambda
a<-data.frame(df=s,lambda=t)

cat(paste("Optimal lambda : ", min(Lasso_model$lambda)))


# 1.4 Fit a Ridge regression model
```

```
Ridge_model <- glmnet(x=as.matrix(covariates),y=response, alpha=0,family="gaussian")
plot(Ridge_model, xvar="lambda", label=TRUE, main = "Regression coefficients depend on the (log
 λ) using RIDGE")

cat(paste("Optimal lambda : ", min(Ridge_model$lambda)))

cv_Lasso_model=cv.glmnet(x=as.matrix(covariates),y=response,alpha=1,family="gaussian")
plot(cv_Lasso_model, xvar="lambda", label=TRUE, main = "Dependence of CV score on log (lambda)")
#coef(cv_Lasso_model, s="lambda.min")

Optimal_lambda<-cv_Lasso_model$lambda.min
cat(paste("Optimal lambda : ", Optimal_lambda))
cat(sep = '\n')
cat(paste("Number of variables chosen by the model : ",length(coef(cv_Lasso_model, s="lambda.mi
n")) - 1))

x_test_covariates <- test_data[,1:100]
cv_fit<-predict(cv_Lasso_model,as.matrix(x_test_covariates))

plot(x=test_data$Fat ,y= cv_fit, main = "Original test vs Predicted test")

library(tidyr)
library(tidyverse)
data<-read.csv("bank-full.csv")
col_names<-colnames(data)
col_names<- unlist(str_split(col_names,"[.]"))
data1<-separate(data,1,col_names,sep = ";")

data1<-data1[,-6]  #removing "balance" which is not present in input variables

data1$age<-as.numeric(data1$age)
data1$job<-as.factor(data1$job)
data1$marital<-as.factor(data1$marital)
data1$education<-as.factor(data1$education)
data1$default<-as.factor(data1$default)
data1$housing<-as.factor(data1$housing)
data1$loan<-as.factor(data1$loan)
data1$contact<-as.factor(data1$contact)
data1$month<-as.factor(data1$month)
data1$day<-as.factor(data1$day)
data1$duration<-as.numeric(data1$duration)
data1$campaign<-as.numeric(data1$campaign)
data1$pdays<-as.numeric(data1$pdays)
data1$previous<-as.numeric(data1$previous)
data1$poutcome<-as.factor(data1$poutcome)
data1$y<-ifelse(data1$y=="yes",1,0) #1 means "yes", 0 means "no"
data1$y<-as.factor(data1$y)

#removing the column "duration" and loading the data
cleaned_data<- data1[ , -which(names(data1) %in% c("duration"))]

#splitting the data into training/validation/test
```

```
n=dim(cleaned_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=cleaned_data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=cleaned_data[id2,]
id3=setdiff(id1,id2)
test=cleaned_data[id3,]
library(tree)

#a. Decision Tree with default settings.
tree_default<-tree(y~., data = train)

#b. Decision Tree with smallest allowed node size equal to 7000.
tree_smallest<-tree(y~.,data = train, minsize=7000)

#c. Decision trees minimum deviance to 0.0005.
tree_min_dev<-tree(y~., data = train,mindev=0.0005)


#predictions with training data
pred_train_default <- predict(tree_default, train, type = "class")
pred_train_smallest <- predict(tree_smallest, train, type="class")
pred_train_min_dev <- predict(tree_min_dev, train, type="class")

#prediction with validation data
pred_valid_default <- predict(tree_default, valid, type = "class")
pred_valid_smallest <- predict(tree_smallest, valid, type = "class")
pred_valid_min_dev <- predict(tree_min_dev, valid, type = "class")

#A function to calculate misclassification error
missclass<- function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#calculating misclassification rate on training data

d1<-missclass(train$y,pred_train_default)
s1<-missclass(train$y,pred_train_smallest)
m1<-missclass(train$y,pred_train_min_dev)

#calculating misclassification rate on validation data

d2<-missclass(valid$y,pred_valid_default)
s2<-missclass(valid$y,pred_valid_smallest)
m2<-missclass(valid$y,pred_valid_min_dev)


misclass_rate<- data.frame("Default"=c(d1,d2),"Smallest_Node"=c(s1,s2),"Min_dev"=c(m1,m2))
```

```
rownames(misclass_rate)<-c("Train","Valid")
misclass_rate
#decision trees minimum deviance to 0.0005.
tree_min_dev<-tree(y~., data = train,mindev=0.0005)

trainScore <- rep(0,50)
validScore <- rep(0,50)

for(i in 2:50) {
  prunedTrain=prune.tree(tree_min_dev,best=i)
  pred_train=predict(prunedTrain, newdata=valid, type="tree")
  trainScore[i]=deviance(prunedTrain)
  validScore[i]=deviance(pred_train)
}
plot(2:50, trainScore[2:50], type="b", col="red", ylim=c(4000,12500))
points(2:50, validScore[2:50], type="b", col="blue")

optimal_leaves <-which.min(validScore[-1])
optimal_leaves
optimal_tree <- prune.tree(tree_min_dev,best = optimal_leaves)
summary(optimal_tree)
#prediction with test data
pred_test <-predict(optimal_tree, test, type = "class")
#confusion matrix
t<-table(true=test$y,pred_test)
t
#accuracy
1-missclass(test$y,pred_test)
recall <-t[2,2]/sum(t[,2])
precision<- t[2,2]/(sum(t[2,]))
F1<-2*precision*recall/(precision+recall)
F1
library(rpart)
loss_matrix <-matrix(c(0,5,1,0), nrow = 2)
tree_test<-rpart(y~., data=train, method="class",parms=list(loss=loss_matrix))
loss_mat_tree <-predict(tree_test, test, type = "class")
#Reference:https://stackoverflow.com/questions/49646377/loss-matrix-in-rs-package-rpart
t_loss<-table(true=test$y,predicted=loss_mat_tree)
t_loss
#accuracy
1-missclass(test$y,loss_mat_tree)
recall_loss <-t_loss[2,2]/sum(t_loss[,2])
precision_loss<- t_loss[2,2]/(sum(t_loss[2,]))
F1_loss<-2*precision_loss*recall_loss/(precision_loss+recall_loss)
F1_loss
#optimal tree
tree_final <- tree(y~., train, mindev=0.0005)
optimal_tree_final <-prune.tree(tree_final, best=23)
optimal_pred<-predict(optimal_tree_final, test, type = "vector")

#Logistic regression
logistic<- glm(y~., data=train, family = "binomial")
```

```
logistic_pred<-predict(logistic, test, type= "response")

#computing TPR and FPR values for different values of threshold(pi)
TPR_tree<-c()
FPR_tree<-c()
TPR_log<-c()
FPR_log<-c()

j<-1

for (i in seq(0.05,0.95,0.05)){
  temp_tree<-ifelse(optimal_pred[,2]>i,1,0)
  temp_log<-ifelse(logistic_pred>i,1,0)
  c1<-table(test$y,temp_tree)
  if(dim(c1)[2]>1){
    TPR_tree[j]<-c1[2,2]/sum(c1[2,])
    FPR_tree[j]<-c1[1,2]/sum(c1[1,])
  }
  c2<-table(test$y,temp_log)
  if(dim(c2)[2]>1){
    TPR_log[j]<-c2[2,2]/sum(c2[2,])
    FPR_log[j]<-c2[1,2]/sum(c2[1,])
  }
  j=j+1
}
plot(FPR_tree, TPR_tree, type = "l",col="red", lwd=2,xlab = "False Positive rate",ylab = "True P
ositive rate", main = "ROC Curves")
lines(FPR_log,TPR_log, type="l", col="blue",lwd=2)
legend("topleft",c("Optimal Tree","Logistic Regression"), fill=c("red","blue"))
crime_data <- read.csv("communities.csv")
sacled_data <- scale(crime_data[ ,-101])
s_c_matrix <- cov(sacled_data )
eigen <- eigen(s_c_matrix)


e_values <- eigen$values

report <-  function(e_values){
  vs <- 0
  counter = 0
  for(i in e_values){

    vs = vs + (i / sum(e_values))
    counter = counter + 1

    if(vs > 0.95){
      break
    }

  }
```

```
   comp.1 = e_values[1]/sum(e_values)
   comp.2 = e_values[2]/sum(e_values)
  res <- list("Total componets" = counter,  "proportion of first componenet" = comp.1,
              "proportion of second componenet" = comp.2)

  return(res)
}


report(e_values)



repaet_pca <- princomp(sacled_data)

comp_1 <- repaet_pca$loadings[, "Comp.1"]
plot(comp_1)


obs <- abs(comp_1)[1:5]
obs

pc_score <- as.data.frame(repaet_pca$scores[, 1:2])
ViolentCrimesPerPop = crime_data[ ,101]


library(ggplot2)
ggplot(pc_score, aes(Comp.1, Comp.2))+
  geom_point(aes(color = ViolentCrimesPerPop))


n=dim(crime_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
test=crime_data[id,]
scale_test = as.data.frame(scale(test))

id1=setdiff(1:n, id)
id2=sample(id1, floor(n*0.5))
train=crime_data[id2,]
scale_train = as.data.frame(scale(train))

lm_m <- lm(ViolentCrimesPerPop ~ ., data = scale_train)

predicted_data <- predict(lm_m, scale_train)
actual_data <- crime_data[,"ViolentCrimesPerPop"]
n <- length(crime_data[,"ViolentCrimesPerPop"])
mse_train <- sum((actual_data - predicted_data) ^2)/n

predicted_data <- predict(lm_m, scale_test)
actual_data <- crime_data[,"ViolentCrimesPerPop"]
n <- length(crime_data[,"ViolentCrimesPerPop"])
```

```
mse_test <- sum((actual_data - predicted_data) ^2)/n

mse_test
mse_train
```