Group A3 Lab 2

varsi146,aswma317,akssr921

2022-12-05

Statement of Contribution

For Lab1, we decided to split the three assignments equally, Aswath completed Assignment 1, Varun completed Assignment 2 and Akshath completed Assignment 3, after which, for verification sake, we completed each other's assignments as well and validated our findings. The report was also compiled by three of us, with each handling their respective assignments.

Assignment 1: Explicit Regularization

Sub Question 1

1 a) Question: Model fat as a linear regression, report the underlying model, report the training and test errors

```
## Model Summary:

##
## Call:
## lm(formula = Fat ~ ., data = train)
##
## 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
```

```
## Training Error: 0.005709117
```

```
## Test Error: 722.4294
```

As seen above, the test error is quite high. Hence, the model is a bad one. This is a case where the model is quite complicated or where there are a lot of features explaining the target variable. In such a model, the sample size has to be a lot more than the number of features, otherwise there would be high variance. This is confirmed by the errors above, the training error was low, but the test error was very high, indicating that the model has high variance.

```
## Number of features used in the model: 100
```

```
## Training samples used: 107
```

Another issue observed is that the target variable Fat has high variance:

```
## Variance in Fat: 158.6242
```

In this case, the solution is to either reduce the features by regularization which will make the model less complex or by increasing the sample size for training.

Sub Question 2

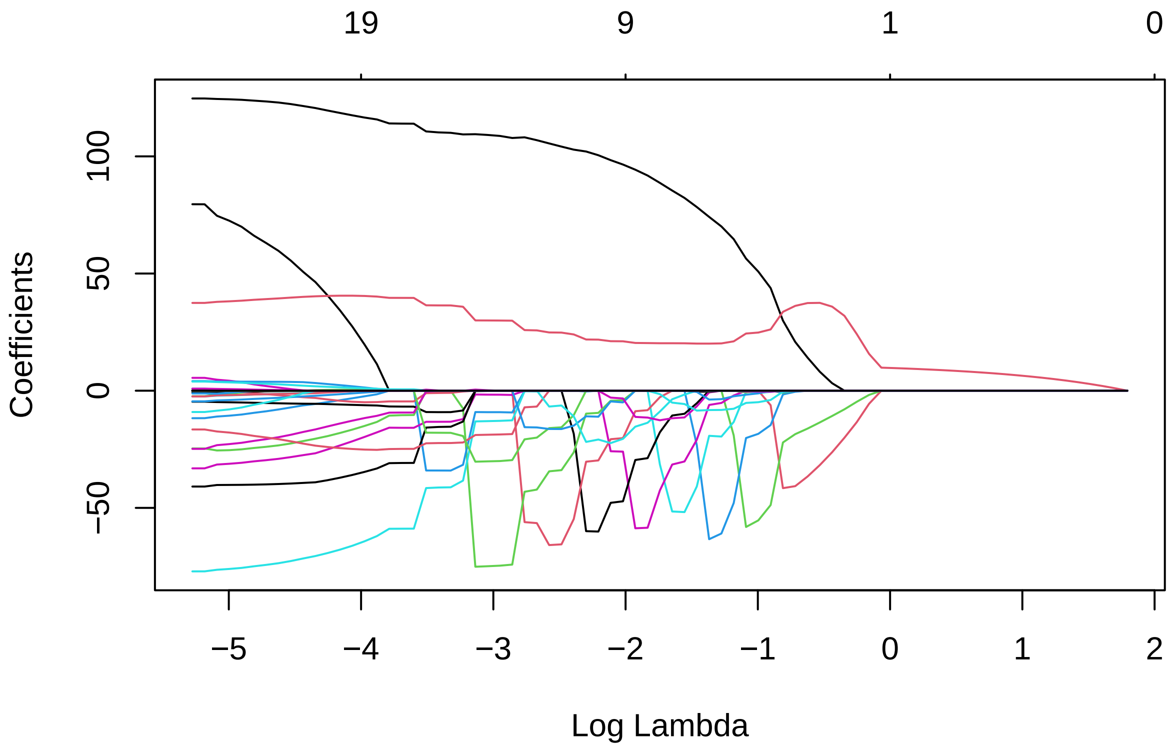
Question: Report the cost function that should be optimized for lasso regression

$$\hat{\theta} = arg \min_{\theta} 1/n||X\theta - y||^2 + \lambda||\theta||$$

Sub Question 3

3 a) Fit the Lasso regression model to training data

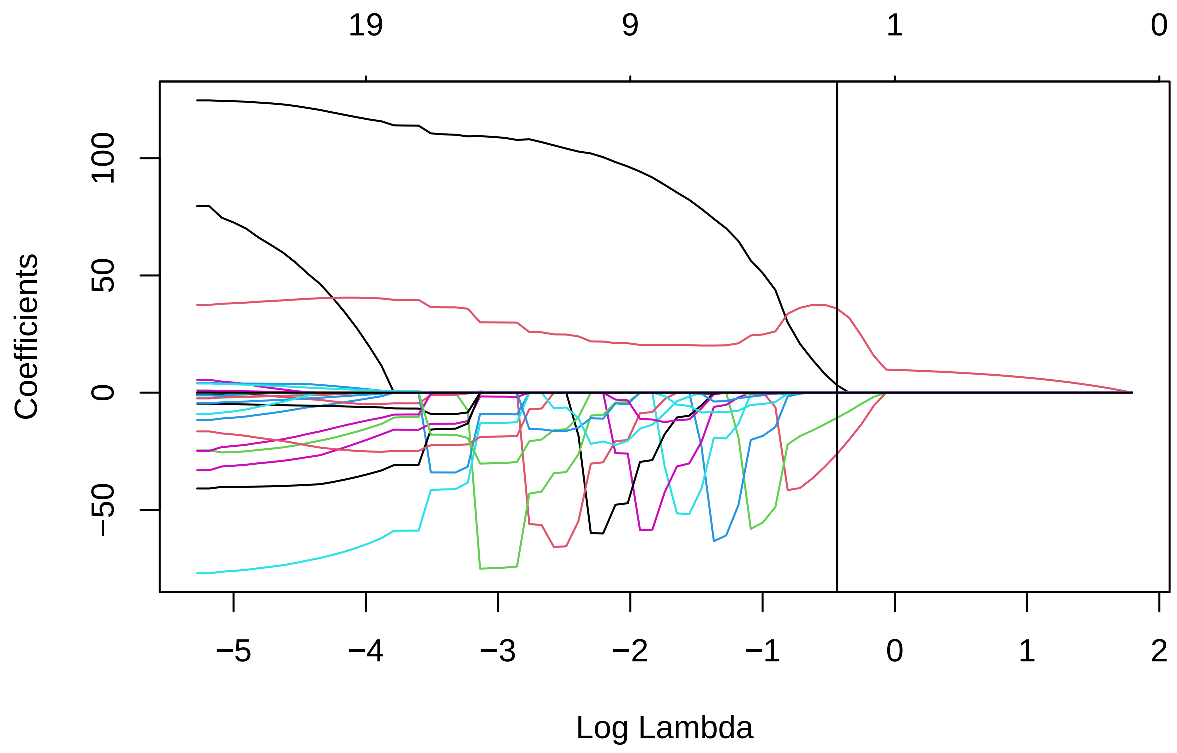
3 b) How does the regression coefficients depend on the log of penalty



At the start, when λ is small, all the features are seen explaining the model. But, as λ increases, some of these features which are of less importance or which does not explain the target much are penalized and brought to 0. While more significant features takes some increments of λ to get to 0.

Note: The expectation was that once the features become 0, it would stay 0, but as seen in the plot, some of these features come back to life.

3 c) What value of penalty factor can be chosen to select 3 features

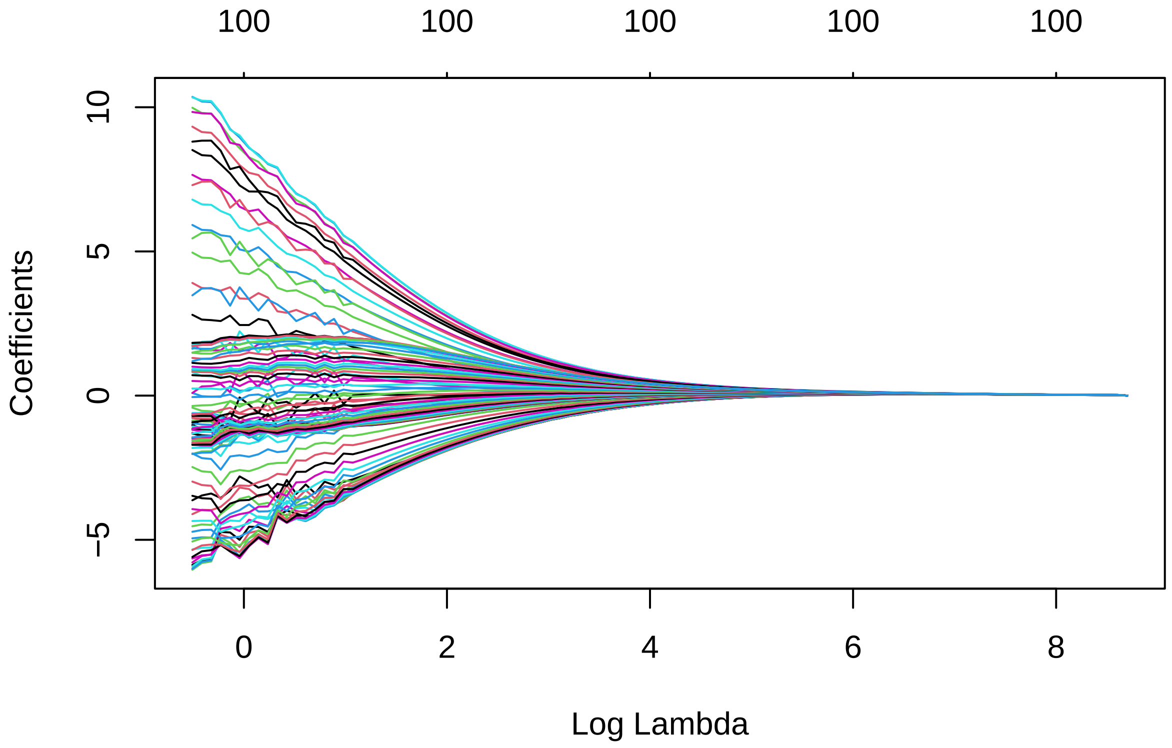


```
## Penalty value: 0.6452974 /log value: -0.438044
```

This penalty chooses 4 features, including the intercept.

Sub Question 4

Fit ridge and compare the plots

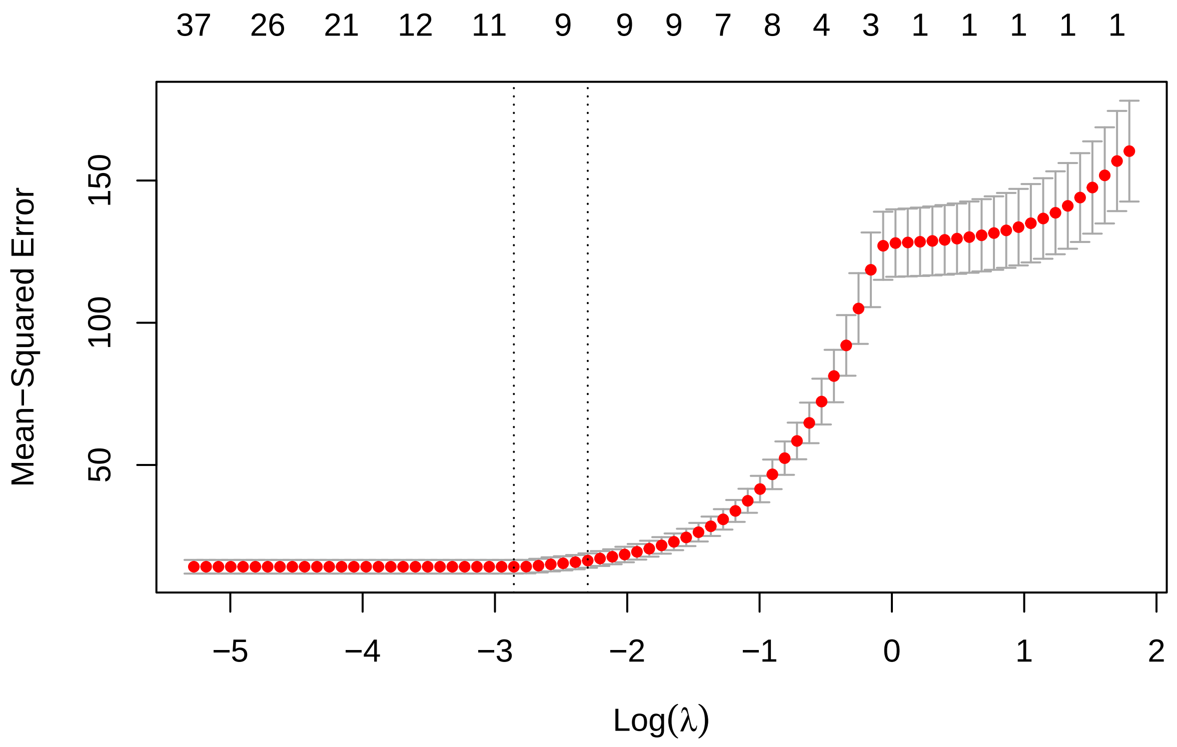


Compared to the lasso, the features are penalized in a more gradual process, it is not done abruptly. It takes high λs for the features to be penalized. Another important observation is that the features are not forced to zero, they linger around 0. This makes it difficult to pick an optimal λ with the required features.

Sub Question 5

5 a) Use cross-validation with default number of folds to compute the optimal LASSO model.

5 b) Present a plot showing the dependence of the CV score on logλ and comment how the CV score changes



with logλ

As lambda increases there are fewer and fewer features explaining the model. In other words we are making the model less complex, which means that there would be more error in training as lambda increases. However, we are generalizing the model or reducing its variance.

We also see that when the model has atlest 9 features, the error is low and stable, but as each of these 9 features are forced to 0 the error increases at a fast pace until we are left with 3 features - this means that these 9 features are significant.

5 c) Report the optimal λ and how many variables were chosen in this model.

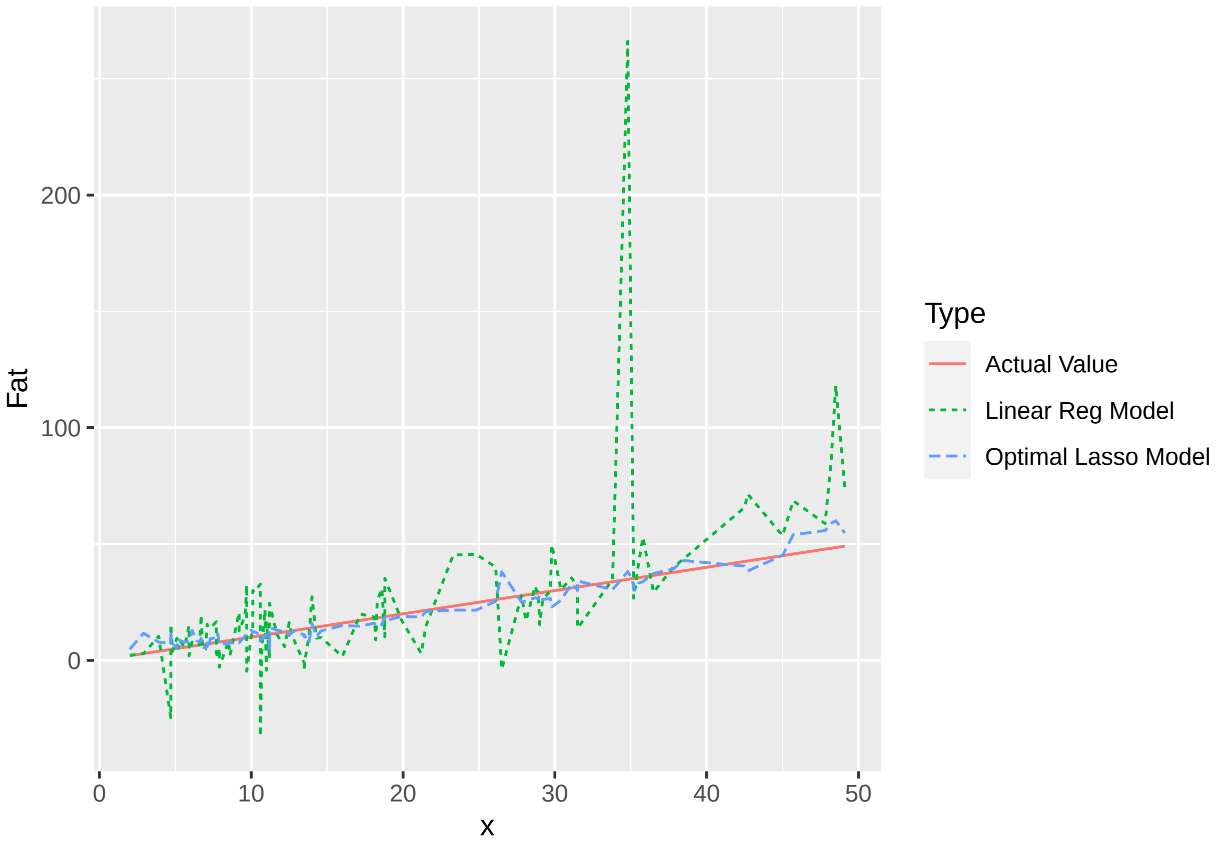
```
## Optimal lambda(in log): -2.856921
```

```
## Number of features used: 8
```

5 d) Does the information displayed in the plot suggests that the optimal λ value results in a statistically significantly better prediction than -4 $log\lambda$?

No, as the range of the mean-square error interval overlaps, it suggests that the optimal λ doesn't not indicate a statistically significant better prediction than than -4 $log\lambda$.

5 e) Scatter plot of the original test versus predicted test values and comment whether the model predictions



are good

Note: the question refers to scatter plot, but the line plot gives more clarity. As seen in the graph, the linear regression model is not able to predict the actual values properly; it could be attributed to the variance in Fat that the linear model is not able to predict properly. There is only a small section(5>Fat>10), where the linear regression model is able to predict at a decent quality, but after that the quality of prediction is bad.

While, the lasso model with optimal lambda value is able to follow the actual line quite perfectly and hence the prediction quality is quite good. Hence, the regularization has improved the model performance.

Assignment 2. Decision trees and logistic regression for bank marketing

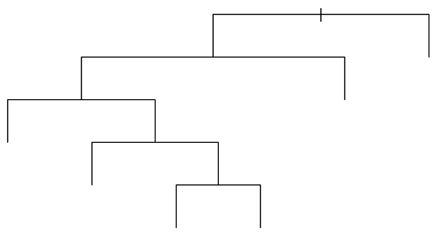
Sub Question 1

See appendix for code.

Sub Question 2

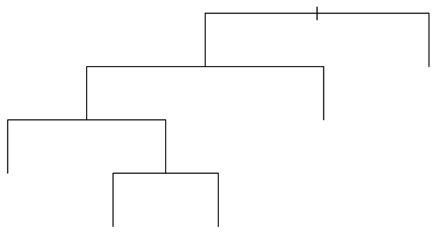
2 a). See appendix for code.

Here, we fit decision tree with default settings.



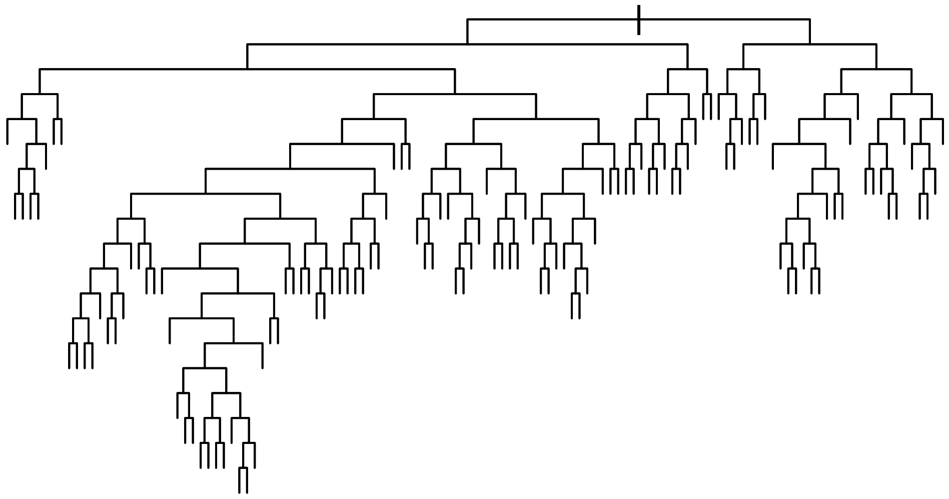
2 b). See appendix for code.

Here, we fit decision tree with smallest allowed node size equal to 7000.



2 c). See appendix for code.

Here, we fit a decision tree with minimum deviance set to 0.0005



We can see below, the misclassification rates for the three models trained above, for training and validation data.

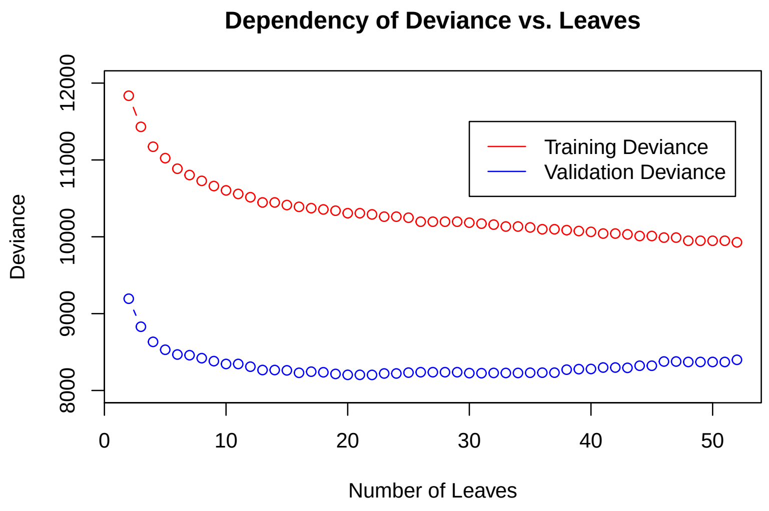
```
##         Model    DataSet MisclassRate
## 1 default_fit bank_train   0.10484406
## 2 default_fit bank_valid   0.10926786
## 3  tree_fit_2 bank_train   0.10484406
## 4  tree_fit_2 bank_valid   0.10926786
## 5  tree_fit_3 bank_train   0.09361867
## 6  tree_fit_3 bank_valid   0.11170095
```

We can say that the 2nd decision tree model is the best one, since it is a much simpler model as compared to the first model although their misclassification rates on test data are identical.

Upon changing the deviance of the third decision tree, that is, the mindev argument to 0.0005, it can be seen that the tree depth has increased considerably and it now has 122 terminal nodes. This is because, when a tree is being constructed the mindev argument determines how much error the potential node should reduce for the tree to grow. Of course, when that threshold is set very low, naturally there would be more terminal nodes thereby increasing tree depth.

As compared to the first tree, on setting the smallest allowed nodesize, that is, the minsize argument in tree function to 7000, it reduces the number of terminal nodes of the second tree from 6 to 5 meaning that it has reduced the tree size. When we look at the summary of the two trees, it can be observed that, without the argument minsize mentioned, ‘month’ is further split to two terminal nodes of ‘housing’. However, when minsize is mentioned, it means that for a node to be split, it must contain at least the number of observations as specified by minsize argument.

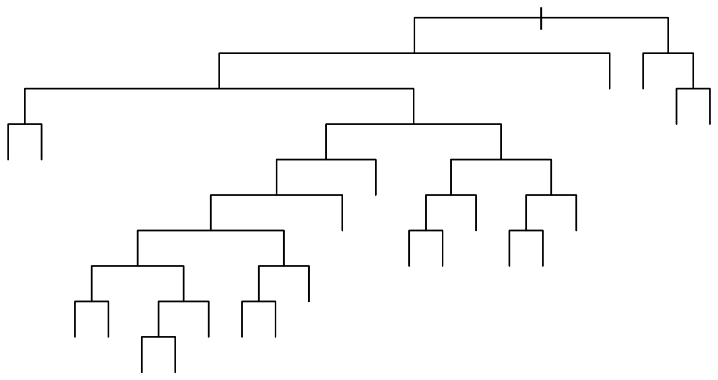
Sub Question 3



In the above graph, the X axis is represented as Number of leaves and can be interpreted in terms of model complexity. Then, we can make the inference that, as the model becomes more complex, it is being over fitted to the data since the training error is reducing but the validation error increases. That is, the bias is decreasing but the variance is increasing and hence the model begins to perform poorly on new unseen data.

The optimal amount of leaves is shown below:

```
## [1] 22
```



```
##
## Classification tree:
## snip.tree(tree = tree_fit_3, nodes = c(581L, 17L, 577L, 79L,
## 37L, 77L, 14L, 576L, 153L, 580L, 6L, 1157L, 15L, 16L, 5L, 1156L,
## 156L, 152L, 579L))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"
## Number of terminal nodes:  22
## Residual mean deviance:  0.5698 = 10290 / 18060
## Misclassification error rate: 0.1039 = 1879 / 18084
```

From the summary above, we can see that the variables that are most important in decision making are:

poutcome, month, contact, pdays, age, day, balance, housing and job

Sub Question 4

The mis-classification rate for the test data using the pruned tree model is:

```
##          Model    DataSet MisclassRate
## 7 opt_tree_new bank_valid    0.1123645
```

```
##          Model   DataSet MisclassRate
## 8 opt_tree_new bank_test    0.1089649
```

The confusion matrix estimation of test data:

```
##      opt_pred_test
##          no   yes
##   no  11872   107
##   yes  1371   214
```

The model metrics when fit on test data are determined to be:

```
##         TPR         FPR Precision    Recall  Accuracy  F1Score
## 1 0.1350158 0.008932298 0.6666667 0.1350158 0.8910351 0.224554
```

We can see from the above model metrics that the accuracy of the model is at around 89.1% and we can say that the model has good predictive score since its performance(based on misclassification rates) on test data is better than on validation data.

Generally, it is better to prefer F1-score over accuracy since accuracy does not take into account how the data is distributed and also since we have a lot of false-negatives, improving the F1 score would indirectly mean that it penalizes the model that has too many false negatives.

Sub Question 5

The confusion matrix after performing classification with the provided loss-matrix:

```
##      new_pred
##          no   yes
##   no  11030   949
##   yes   771   814
```

Change in misclassification rate:

```
## [1] 0.1268063
```

Change in model metrics after inclusion of the loss matrix:

```
##         TPR        FPR Precision    Recall  Accuracy   F1Score
## 2 0.5135647 0.07922197  0.461713 0.5135647 0.8731937 0.4862605
```

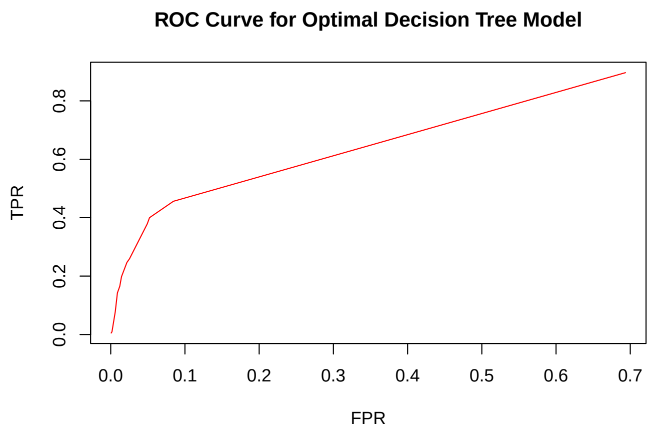
After performing a classification by including a loss matrix, we see that there is a much better balance between precision and recall and also that the F1-score has improved by 2-fold. This is because, after the inclusion of loss matrix, we have penalized those predictions that were false negatives and by doing so, our F1 score has improved considerably improved while accuracy has reduced due to the increase in misclassification rates, albeit only by a margin of approximately 2%.

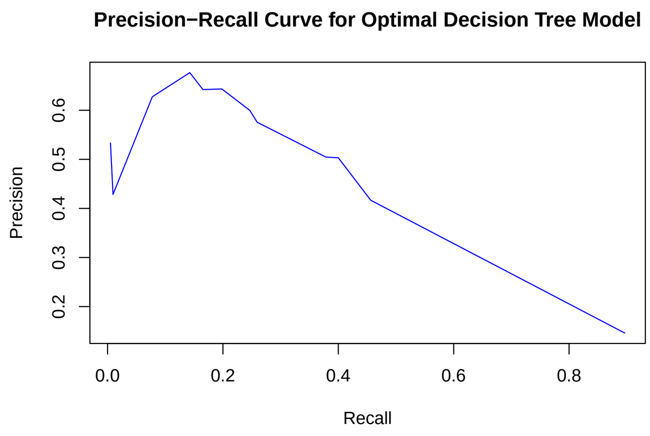
Sub Question 6

Here, we perform the classification of test data by the following principle:

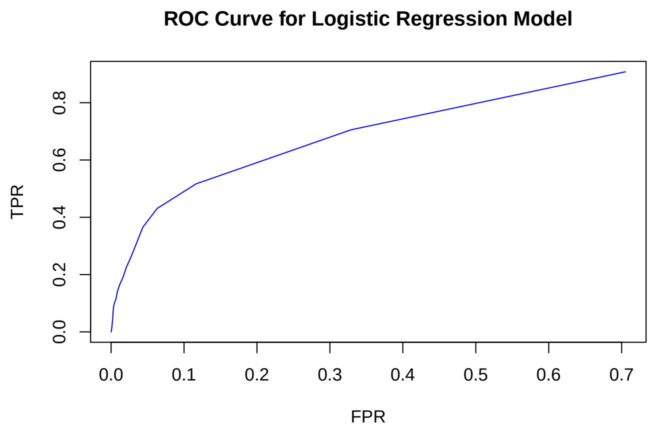
$\hat{Y}$ = yes if $p(Y =' yes'|X) > \pi$, else $\hat{Y}$ = no, where $\pi = 0.05, 0.1, 0.15, \ldots, 0.9, 0.95$

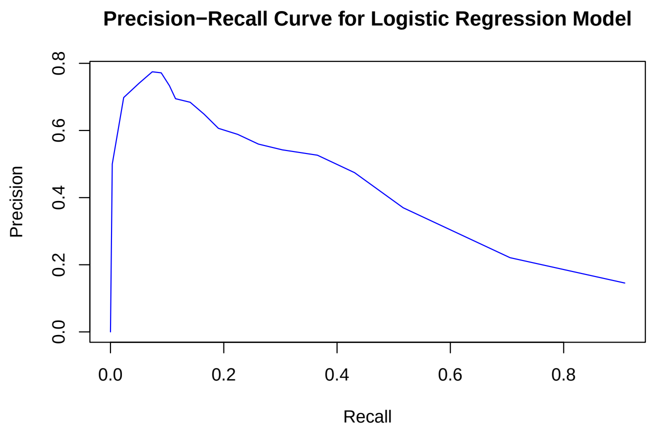
The ROC curve after using the optimal tree and performing classification based on the above principle:





The ROC curve after using the logistic regression model and performing classification based on the above principle:





The conclusion we can draw from the ROC curves is that, for any given FPR the TPR is higher which means that we have a good classifier.

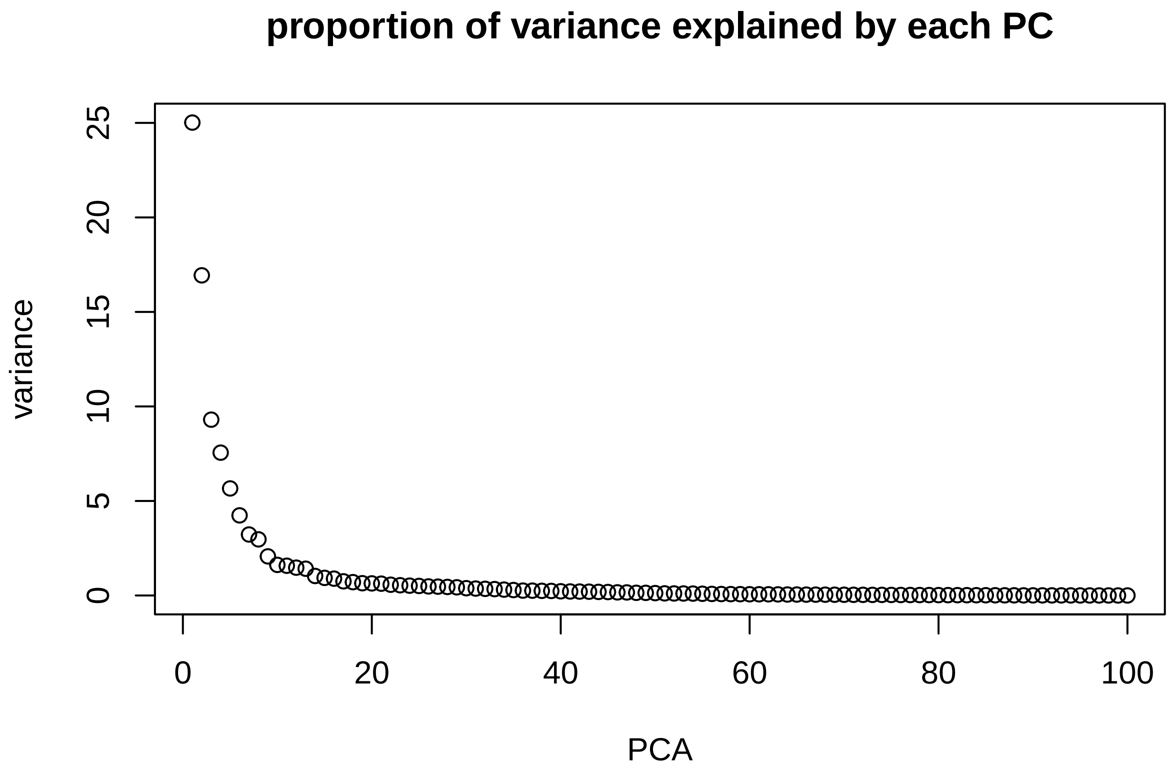
The precision-recall curve in this case would be a better option because they are more insensitive to the changing classification threshold as compared to ROC curve whose shape changes dramatically with changes in threshold values since changes in classification threshold directly impacts TPR and FPR values. Hence, the precision recall curve provides a more stable view of the model's performance.

Assignment 3. Principal components and implicit regularization

Question 1

Sub question 1

The proportion of variance explained by each of the first two principle components is 0.2501699(25%) and 0.1693597(16.9%)



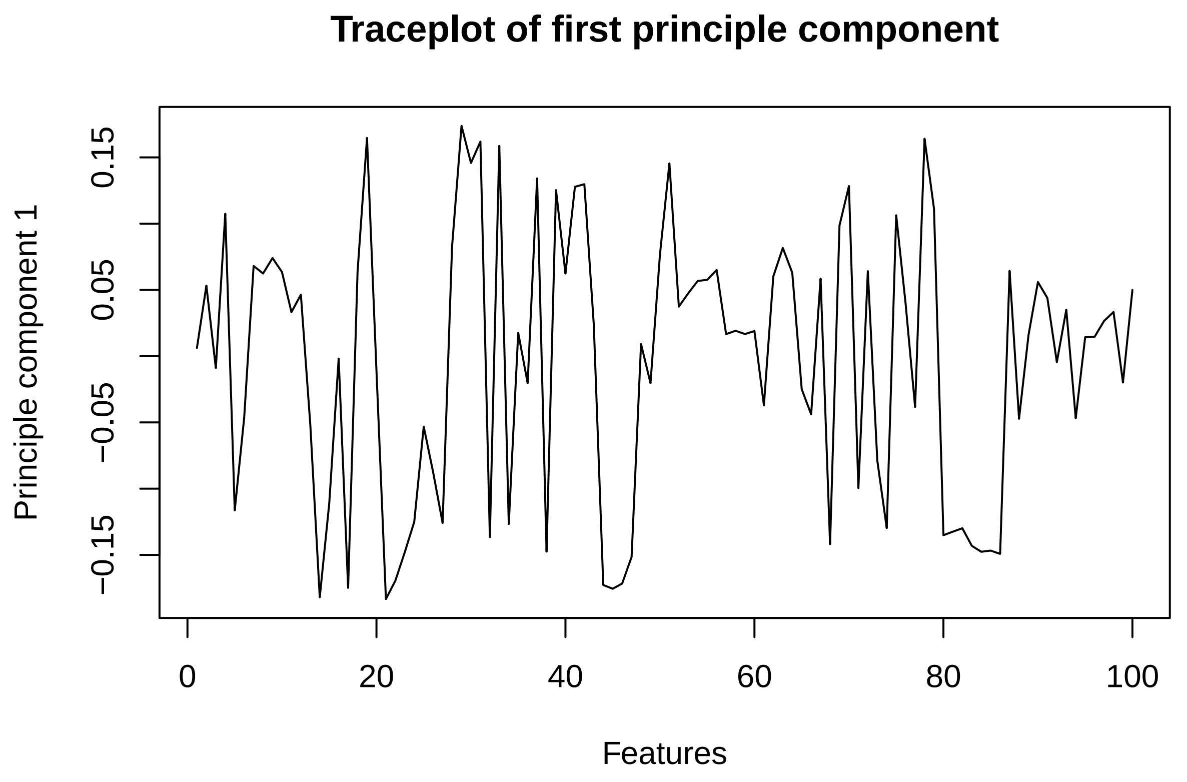
```
## Components needed for 95% variance is : 34

## Proportion of variation explained by first component is : 0.2501699

## Proportion of variation explained by second components is : 0.1693597
```

Sub question 2

From the trace plot of PC1 below, we can say that few features have significant contribution to this component.



The 5 features which contribute mostly to principle component 1 are shown below.

All the 5 features have logical relationship to the crime rate

medFamInc: median family income (differs from household income for non-family households)

The family income has dependency on violent crime. Children of parents with lowest income are being convicted of violent criminality compared to the children with parents earning high income.

medIncome: median household income

When the household income is less, the person will look for a easy money which may lead to violent crimes.

PctKids2Par: percentage of kids in family housing with two parents

Family conflicts will lead to violent crimes. Youth experiencing homelessness and runaway youth consistently identify family conflict as the primary reason for homelessness. These youth experience higher rate of serious violence.

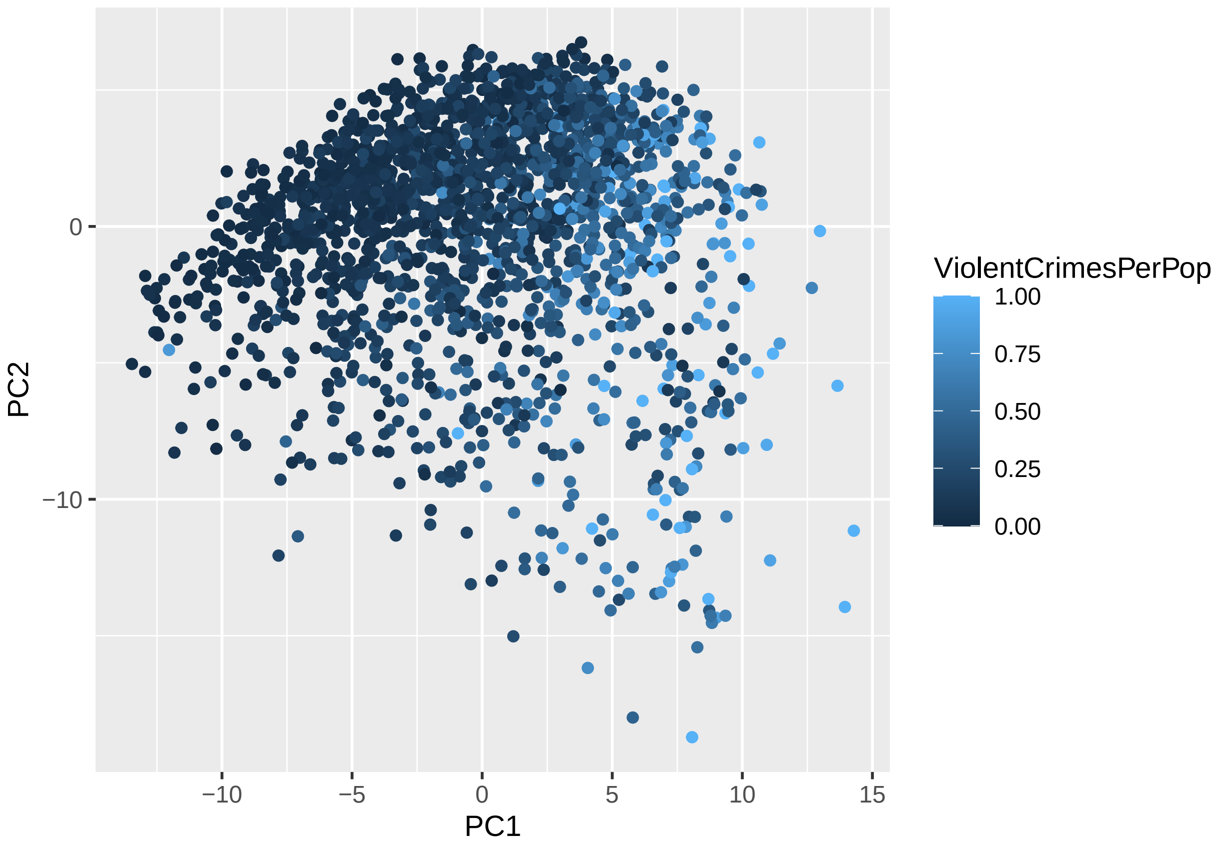
pctWInvInc: percentage of households with investment / rent income in 1989

Lower percentages of households with investment/rent income in 1989 in a community may or may not be linked to higher rates of violent crime. It cannot be said with certainty that households with low investment/rent income contribute to violent crimes.

PctPopUnderPov: percentage of people under the poverty level

Poverty can produce violent crimes because it is an easy way to get large quantities of goods.

```
##      medFamInc      medIncome    PctKids2Par     pctWInvInc PctPopUnderPov
##      0.1833080      0.1819830      0.1755423      0.1748683      0.1737978
```



From the plot above we can say that less violent crimes(darker color) in the top left area and more violent crimes (lighter color) in the bottom right side area. We can conclude that higher scores are assigned to PC1 and lesser scores are assigned to PC2 for violent crimes per population.

Sub question 3

Below we can see the Test and Training error, by looking at these values we can comment that quality of model is pretty good.

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

## Training Mean squared error is : 0.2828799

## Test Mean squared error is : 0.4141649
```

Sub question 4

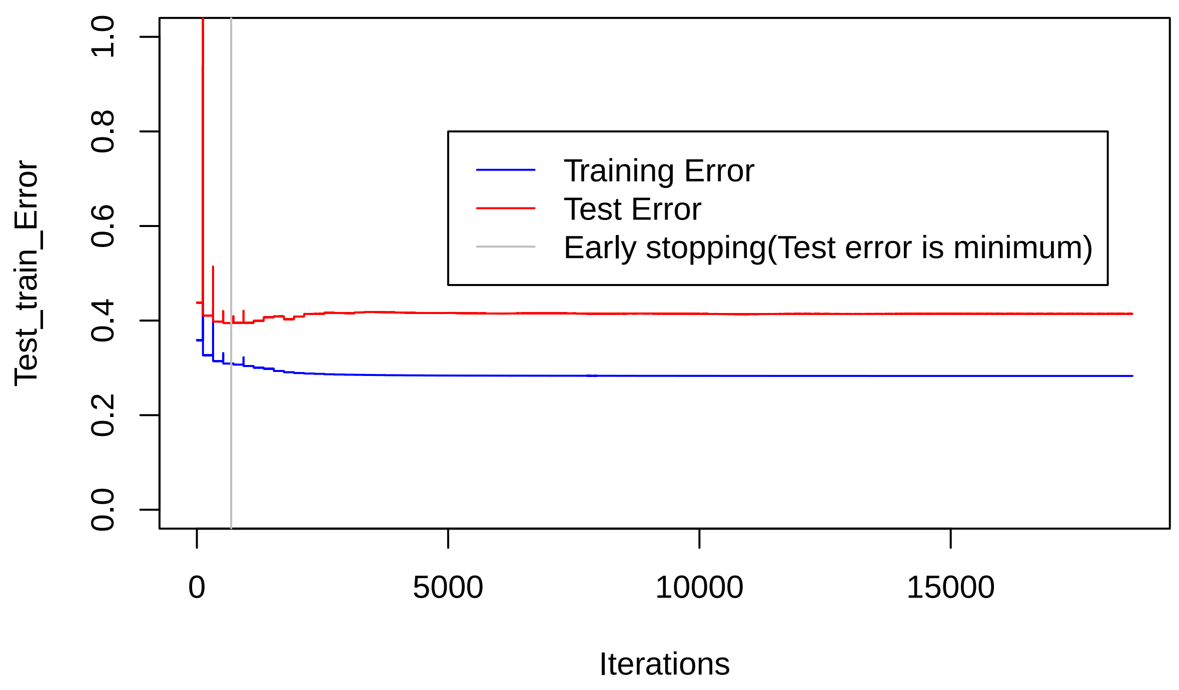
Early stopping can be done at iteration 2182 where test error becomes minimum. Training and test error of optimal model at iteration 2182 can be seen below, by comparing to the test error at step 3 above, we can conclude that model quality is improved when we optimize the parameters using BFGS method(optim() function without gradient specified),the test error(mean squared error) drops from 0.4141649 to 0.3946325. However, MSE of training data increases slightly compared to step 3.

```
## Iteration for early stopping 2182
```

```
## Test error of optimal model at 2182 iteration is 0.3946325
```

```
## Training error of optimal model at 2182 iteration is 0.3090996
```

To the plot above we have added the abline to the point of iteration where test error is minimum i.e 0.3946325.



Appendix

```
library(dplyr)
library(tree)
library(ggplot2)
library(pROC)
library(glmnet)
knitr::opts_chunk$set(echo = TRUE)
######### Code for Explicit Regularization #########
#Read the data
data <- read.csv('tecator.csv', header = TRUE)
data <- data %>% select(Fat, Channel1:Channel100)

#Splitting the data to test and train
set.seed(12345)
```

```r
n <- nrow(data)#215
train_index <- sample(1:n,
                      size = floor(0.5*n)) #Assuming 50% for train
train <- data[train_index,]
test <- data[-train_index,]

#Linear Regression model: Fat~Channels
mod <- lm(formula = Fat~., data = train)
lin_train_predict <- predict(mod, newdata = train)
lin_test_predict <- predict(mod, newdata = test)
cat('Model Summary:')
summary(mod)
train_mse <- mean((train$Fat - lin_train_predict)^2)
test_mse <- mean((test$Fat - lin_test_predict)^2)
cat('Training Error:', train_mse)
cat('Test Error:', test_mse)
cat('Number of features used in the model:', length(colnames(train))-1)
cat('Training samples used:', nrow(train))
cat('Variance in Fat:', var(train$Fat))
x_train <- as.matrix(train %>% select(-Fat))
y_train <- as.matrix(train %>% select(Fat))
lasso_mod <- glmnet(x = x_train, y = y_train,
                    family = 'gaussian', alpha = 1)
plot(lasso_mod, xvar = 'lambda')
lambda_degfree_df <- data.frame(lambda = lasso_mod$lambda,
                                deg_free = lasso_mod$df)
lambda_degfree_df <- lambda_degfree_df %>% arrange(desc(deg_free))
val <- lambda_degfree_df[which(lambda_degfree_df$deg_free==4),] %>%
  head(1) %>% pull(lambda)
plot<- plot(lasso_mod, xvar = 'lambda')
abline(v = c(log(val)))
cat('Penalty value:', val,'/log value:',log(val))
ridge_mod <- glmnet(x = x_train, y = y_train,
                    family = 'gaussian', alpha = 0)
plot(ridge_mod, xvar = 'lambda')
cv_mod <- cv.glmnet(x_train, y_train, family = 'gaussian',
                    alpha = 1)#1 for Lasso and 0 for Ridge
plot(cv_mod)
cat('Optimal lambda(in log):', log(cv_mod$lambda.min))
cat('Number of features used:', cv_mod$nzero[which(cv_mod$lambda == cv_mod$lambda.min)])
#Optimal model using the optimal lambda
x_test <- as.matrix(test %>% select(-Fat))
y_test <- as.matrix(test %>% select(Fat))
opt_lasso_mod <- glmnet(x = x_train, y = y_train,
                    family = 'gaussian', alpha = 1,
                    lambda = cv_mod$lambda.min)
opt_lasso_pred <- predict(opt_lasso_mod, 'response', newx = x_test)

scatter_df <- as.data.frame(matrix(nrow=0, ncol = 3))
colnames(scatter_df) <- c('x', 'Type', 'Fat')
#Include Actuals, Linear Reg prediction & optimal prediction
scatter_df <- rbind(scatter_df, data.frame(
  x = test$Fat,
  Type = rep('Actual Value', length(test$Fat)),
  Fat = test$Fat
), data.frame(
  x = test$Fat,
  Type = rep('Linear Reg Model', length(test$Fat)),
  Fat = lin_test_predict
), data.frame(
  x = test$Fat,
  Type = rep('Optimal Lasso Model', length(test$Fat)),
  Fat = as.vector(opt_lasso_pred)
))
ggplot(scatter_df, aes(x = x, y = Fat)) +
  geom_line(aes(color = Type, linetype = Type))

######### Code for Decision Trees and Logistic Regression #########

# Reading the bank data set

# setwd('C:/Users/Varun/Desktop/LiU - STIMA/Machine Learning/Labs/Lab 2')
bank_dataset <- read.csv2('bank-full.csv', header = TRUE)

# Removing the Duration column

bank_dataset_new <- bank_dataset[,-12]

# Converting strings/characters to factors

bank_dataset_new <- bank_dataset_new%>%
  mutate_if(is.character, as.factor)

# Hold-out - partitioning into train, valid and test.

n <- dim(bank_dataset_new)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.4))

bank_train <- bank_dataset_new[id,]

id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n*0.3))

bank_valid <- bank_dataset_new[id2,]

id3 <- setdiff(id1, id2)

bank_test <- bank_dataset_new[id3,]
# Default setting decision tree:

default_fit <- tree(y~., data = bank_train)
plot(default_fit, type = 'uniform')

# Decision tree with smallest allowed node size set to 7000:
```

```r
tree_fit_2 <- tree(y~.,
                   data = bank_train,
                   control = tree.control(nobs = nrow(bank_train),
                                          minsize = 7000),
                   split = c('deviance', 'gini'))

plot(tree_fit_2, type = 'uniform')
# Decision tree with minimum deviance set to 0.0005:

tree_fit_3 <- tree(y~.,
                   data = bank_train,
                   control = tree.control(nobs = nrow(bank_train),
                                          mindev = 0.0005),
                   split = c('deviance', 'gini'))
plot(tree_fit_3, type = 'uniform')

# Function to calculate mis-classification  error
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

# Data frame that stores details about mis-classification errors

misclass_df <- data.frame(matrix(ncol = 3, nrow = 0))
names(misclass_df) <- c('Model', 'DataSet', 'MisclassRate')

# Function that calculates mis-classification error based on model and data set.
calc_misclass_error <- function(model, data_set){

  prob_tree_calc <- predict(object = model,
                            newdata = data_set)
  bestI_calc <- apply(prob_tree_calc, MARGIN = 1, which.max)
  pred_tree_calc <- levels(data_set$y)[bestI_calc]
  data_set_misclass_err <- missclass(data_set$y, pred_tree_calc)

  misclass_df <<- rbind(misclass_df,
                        data.frame('Model' = as.character(substitute(model)),
                                   'DataSet' = as.character(substitute(data_set)),
                                   'MisclassRate' = data_set_misclass_err))
  return(misclass_df)
}
# Running the function on different models and data sets:

calc_misclass_error(model = default_fit, data_set = bank_train)
calc_misclass_error(model = default_fit, data_set = bank_valid)
calc_misclass_error(model = tree_fit_2, data_set = bank_train)
calc_misclass_error(model = tree_fit_2, data_set = bank_valid)
calc_misclass_error(model = tree_fit_3, data_set = bank_train)
calc_misclass_error(model = tree_fit_3, data_set = bank_valid)
misclass_df
# Using the model from sub question 2c to find optimal depth:
```

```r
opt_res <- cv.tree(object = tree_fit_3)

trainScore <- rep(0,52)
validScore <- rep(0,52)
prune_df <- data.frame(matrix(ncol = 3, nrow = 0))
colnames(prune_df) <- c('Leaves', 'Training Deviance', 'Validation Deviance')
for (i in 2:52) {
  opt_tree_pruned <- prune.tree(tree_fit_3, best = i)
  pred_valid <- predict(opt_tree_pruned,
                        newdata = bank_valid,
                        type = 'tree')
  trainScore[i] <- deviance(opt_tree_pruned)
  validScore[i] <- deviance(pred_valid)
  prune_df <- rbind(prune_df, data.frame('Leaves' = i,
                                         'Training Deviance' = trainScore[i],
                                         'Validation Deviance' = validScore[i]))
}

# Plot of dependency of training and validation deviances on number of leaves:

plot(2:52, prune_df[,2], type = 'b', col = 'red', ylim = c(8000, 12000),
     main = 'Dependency of Deviance vs. Leaves',xlab = 'Number of Leaves',
     ylab = 'Deviance')
points(2:52, prune_df[,3], type = 'b', col = 'blue')
legend(x = 30, y = 11500, legend = c('Training Deviance', 'Validation Deviance'),
       col = c('red', 'blue'), lty = rep(1,2))
# Determining the optimum number of leaves based on model deviances on validation data:

optimal_leaves <- prune_df[which.min(prune_df$Validation.Deviance),1]
optimal_leaves
# Fitting the decision tree from 2c and pruning the tree to the optimal leaves
# found in the steps above:

opt_tree_new <- prune.tree(tree = tree_fit_3,
                           best = optimal_leaves)
plot(opt_tree_new, type = 'uniform')
summary(opt_tree_new)

# Predictions on validation data using the tree model from above:

opt_pred_valid <- predict(opt_tree_new,
                          newdata = bank_valid,
                          type = 'class')
calc_misclass_error(model = opt_tree_new,
                    data_set = bank_valid)

# Predictions on test data using the tree model from above:

opt_pred_test <- predict(object = opt_tree_new,
                         newdata = bank_test,
                         type = 'class')

calc_misclass_error(model = opt_tree_new,
                    data_set = bank_test)

misclass_df[7,]
misclass_df[8,]
table(bank_test$y, opt_pred_test)
# Model Metrics data frame to store data:

model_metric_df <- data.frame(matrix(ncol = 6, nrow = 0))
colnames(model_metric_df) <- c('TPR',
                               'FPR',
                               'Precision',
                               'Recall',
                               'Accuracy',
                               'F1Score')
# Function to calculate model metrics:

calc_model_metric <- function(act_vals,pred_vals){
  confusion_matrix <- table(act_vals,pred_vals)
  TN <- confusion_matrix[1,1]
  TP <- confusion_matrix[2,2]
  FP <- confusion_matrix[1,2]
  FN <- confusion_matrix[2,1]
  N <- TN + FP
  P <- TP + FN
  TPR <- TP/P
  FPR <- FP/N
  recall <- TP/(TP+FN)
  precision <- TP/(TP+FP)
  accuracy <- (TP+TN)/(P+N)
  f1_score <- (2*precision*recall)/(recall+precision)
  model_metric_df <<- rbind(model_metric_df, data.frame('TPR' = TPR,
                                                        'FPR' = FPR,
                                                        'Precision' = precision,
                                                        'Recall' = recall,
                                                        'Accuracy' = accuracy,
                                                        'F1Score' = f1_score))
  return(model_metric_df)

}
# Calculation of model performance when fit on test data:

calc_model_metric(bank_test$y, opt_pred_test)

# Predicting by setting type = vector to get probability of classifications:

opt_pred_test_loss <- predict(object = opt_tree_new,
                         newdata = bank_test,
                         type = 'vector')
new_test_ds <- cbind(bank_test, as.data.frame(opt_pred_test_loss))

true_labels <- new_test_ds[,16]
new_pred <- c()
```

```r
# penalizing the predictions by comparing to the true labels:

for (i in 1:length(true_labels)) {
  if (((new_test_ds$no[i]/new_test_ds$yes[i])>5)) {
    new_pred <- rbind(new_pred, 'no')
  }else{
    new_pred <- rbind(new_pred, 'yes')
  }
}
table(bank_test$y, new_pred)
missclass(bank_test$y, new_pred)
calc_model_metric(bank_test$y, new_pred)[2,]

opt_tree_new_pi <- prune.tree(tree = tree_fit_3,
                              best = optimal_leaves,
                              method = 'misclass')
# pred_test_tree <- predict(opt_tree_new_pi,
#                            newdata=bank_test,
#                            # type='vector')


log_model_pi <- glm(formula = y~., family = binomial, data = bank_train)
# pred_test_logmodel <- predict(log_model_pi,newdata=bank_test,type = 'response')

model_eval_df <- as.data.frame(matrix(ncol = 5, nrow = 0))
colnames(model_eval_df) <- c('Pi_Vals','TPR', 'FPR', 'Precision', 'Recall')


calc_pi_func <- function(model_pi, pi_vals){
  new_pred_pi <- c()
  if (isTRUE(attr(x = model_pi, which = 'class') == 'tree')) {
    opt_pred_test_pi <- as.data.frame(predict(object = model_pi,
                                              newdata = bank_test))
    for (i in 1:nrow(opt_pred_test_pi)) {
      if ((opt_pred_test_pi$yes[i]>pi_vals)) {
        new_pred_pi <- rbind(new_pred_pi, 'yes')
      }else{
        new_pred_pi <- rbind(new_pred_pi, 'no')
      }
    }
    conf_matrix_pi <- table(bank_test$y, new_pred_pi)
    TN_pi <- conf_matrix_pi[1,1]
    TP_pi <- conf_matrix_pi[2,2]
    FP_pi <- conf_matrix_pi[1,2]
    FN_pi <- conf_matrix_pi[2,1]
    N_pi <- TN_pi + FP_pi
    P_pi <- TP_pi + FN_pi
    TPR_pi <- TP_pi/P_pi
    FPR_pi <- FP_pi/N_pi
    recall <- TP_pi/(TP_pi+FN_pi)
    precision <- TP_pi/(TP_pi+FP_pi)
    model_eval_df <<- rbind(model_eval_df, data.frame('Pi_Vals' = pi_vals,
                                                      'TPR' = TPR_pi,
                                                       'FPR' = FPR_pi,
                                                       'Precision' = precision,
                                                       'Recall' = recall))

  }else{
    logistic_prob_test_pi <- predict(object = model_pi, bank_test, type = 'response')
    logistic_pred_test_pi <- ifelse(logistic_prob_test_pi > pi_vals, 'yes', 'no')
    conf_matrix_pi <- table(bank_test$y, logistic_pred_test_pi)
    TN_pi <- conf_matrix_pi[1,1]
    TP_pi <- conf_matrix_pi[2,2]
    FP_pi <- conf_matrix_pi[1,2]
    FN_pi <- conf_matrix_pi[2,1]
    N_pi <- TN_pi + FP_pi
    P_pi <- TP_pi + FN_pi
    TPR_pi <- TP_pi/P_pi
    FPR_pi <- FP_pi/N_pi
    recall <- TP_pi/(TP_pi+FN_pi)
    precision <- TP_pi/(TP_pi+FP_pi)
    model_eval_df <<- rbind(model_eval_df, data.frame('Pi_Vals' = pi_vals,
                                                       'TPR' = TPR_pi,
                                                       'FPR' = FPR_pi,
                                                       'Precision' = precision,
                                                       'Recall' = recall))
  }
  return(model_eval_df)
}

pi_val <- seq(0.05, 0.95, 0.05)

for (i in 1:length(pi_val)) {
  calc_pi <- calc_pi_func(model_pi = opt_tree_new_pi,pi_vals = pi_val[i])
}

for (i in 1:length(pi_val)) {
  calc_pi <- calc_pi_func(model_pi = log_model_pi,pi_vals = pi_val[i])
}
# log_model_roc <- roc(bank_test[,16],pred_test_logmodel)
# tree_model_roc <- roc(bank_test[,16],pred_test_tree[,2])

plot(x = model_eval_df$FPR[1:19],
     y = model_eval_df$TPR[1:19],
     type = 'l', col = 'red',
     main = 'ROC Curve for Optimal Decision Tree Model',
     xlab = 'FPR', ylab = 'TPR')
plot(x = model_eval_df$Recall[1:19],
     y = model_eval_df$Precision[1:19],
     type = 'l', col = 'blue',
     main = 'Precision-Recall Curve for Optimal Decision Tree Model',
     xlab = 'Recall', ylab = 'Precision')

plot(x = model_eval_df$FPR[20:38],
     y = model_eval_df$TPR[20:38],
     type = 'l', col = 'blue',
     main = 'ROC Curve for Logistic Regression Model',
     xlab = 'FPR', ylab = 'TPR')

plot(x = model_eval_df$Recall[20:38],
     y = model_eval_df$Precision[20:38],
     type = 'l', col = 'blue',
     main = 'Precision-Recall Curve for Logistic Regression Model',
     xlab = 'Recall', ylab = 'Precision')




########################Assignment 3###################

#reading data
com <- read.csv('communities.csv', header = TRUE)

#scaling data
com.s<-scale(com[,-ncol(com)])

#calculating eigen
e<-eigen(cov(com.s))

#proportion variation
prop_var<-e$values/100

plot((prop_var*100),main='proportion of variance explained by each PC',xlab='PCA',ylab='variance')

#cumulative proportion
cumulative_proportion<-cumsum(e$values)

# 95% variance
cat('Components needed for 95% variance is :',sum(cumulative_proportion<95))

#proportion variance of first two
first<-prop_var[1]
cat('Proportion of variation explained by first component is :',first)

second<-cat('Proportion of variation explained by second components is :',prop_var[2])

data<-as.matrix(com.s) %*% e$vectors
colnames(data)<-colnames(com.s)


# princomp
x<-princomp(com.s)
l<-x$sdev^2

#proportion variation
prop_variation<-(l/sum(l))*100
res<-x$loadings

plot(res[,1],type='l',ylab='Principle component 1', xlab='Features',main='Traceplot of first principle 
```

```r
# contribution for 1st PC

sort(abs(res[,1]),decreasing = TRUE)[1:5]
my_df<-as.data.frame(x$scores)
my_df$ViolentCrimesPerPop<-com[,ncol(com)]

#PC score plot
library(ggplot2)
ggplot(data=my_df,aes(x=my_df[,1],y=my_df[,2]))+
  geom_point(aes(colour = ViolentCrimesPerPop,fill=ViolentCrimesPerPop))+xlab('PC1')+ylab('PC2')

#splitting the data
set.seed(12345)
n <- nrow(com)
train_index <- sample(1:n, size=floor(n*0.6))
train <- com[train_index,]
test <- com[-train_index,]

# scaling
params <- caret::preProcess(train)
test <- predict(params, test)
train <- predict(params, train)

# model creation
model<-lm(ViolentCrimesPerPop~.,train)
test_pred<-predict(model,test)
train_pred<-predict(model,train)

# mean square error
library(rminer)
mse_test<-mmetric(test[,ncol(test)],test_pred,'MSE')
mse_train<-mmetric(train[,ncol(train)],train_pred,'MSE')

cat('Training Mean squared error is :',mse_train)
cat('Test Mean squared error is :',mse_test)


# optimizing parameters
test_error<-list()
train_error<-list()
i<-0

cost_func_train<-function(x){
  theta<-matrix(x[1:100],ncol = 1,nrow=100)
  X_train<-model.matrix(ViolentCrimesPerPop~.-1,train)
  X_test<-model.matrix(ViolentCrimesPerPop~.-1,test)
  mse_train<-(sum(((X_train%*%theta)-train[,ncol(train)])^2))/nrow(train)
  mse_test<-(sum(((X_test%*%theta)-test[,ncol(train)])^2))/nrow(test)
  .GlobalEnv$i<- .GlobalEnv$i+1
  .GlobalEnv$train_error[[i]]<-mse_train
  .GlobalEnv$test_error[[i]]<-mse_test
  return(mse_train)
```

```
}

res<-optim(c(rep(0,100)), fn=cost_func_train,  method="BFGS")


# Iteration for early stopping

cat('Iteration for early stopping',which.min(unlist(test_error)))

#2182 iteration early stopping
cat('Test error of optimal model at 2182 iteration is', test_error[[which.min(unlist(test_error))]])
cat('Training error of optimal model at 2182 iteration is', train_error[[which.min(unlist(test_error))]


plot(as.numeric(train_error[1500:length(train_error)]), type="l", col="blue", ylim=c(0,1),
     ylab="Test_train_Error",xlab='Iterations')
points(as.numeric(test_error[1500:length(test_error)]),type='l', col="red")
abline(v=which.min(test_error[1500:length(test_error)]),col='gray')
legend(x = 5000, y = 0.8,
       legend = c('Training Error', 'Test Error','Early stopping(Test error is minimum)'),
       col = c('blue', 'Red','gray'), lty = c(1,1))
```