Lab2 Report

Group B33

2023/12/05

Group members: Priyansh Gupta, Afshan Hashemi Hosseinabad, Ragini Gunda, Yuting Huang

Statement of contribution: Assignment 1 is mainly done by Ragini Gunda, Priyansh Gupta was mainly responsible for assignment 2 and, assignment 3 is mostly done by Afshan Hashemi Hosseinabad. Results from all assignments are discussed afterwards between all of us and the group report is formulated based on the discussion. The report is reviewed by Yuting Huang based on the discussion.

Assignment 1: Explicit regularization

1a. Fit the linear regression to the training data and estimate the training and test errors.
Task 1:

```
fileName <- read.csv("tecator.csv")
#Dividing the data(50/50)
noOfrows <- nrow(fileName)
set.seed(12345)
id <- sample(1:noOfrows,floor(noOfrows*0.5))
training <- fileName[id, ]
testing <- fileName[-id, ]
#install.packages("dplyr")

library(dplyr)
training <- training %>% select(2:102)
testing <- testing %>% select(2:102)

#Fit the linear regression model
linear_model <- lm(Fat~.,training)
summary(linear_model)
predictionsTrain <- predict(linear_model, training)
predictionTest <- predict(linear_model,testing)

#MSE of Training and testing
MSE_training <- mean((training$Fat-predictionsTrain)^2)
MSE_testing <- mean((testing$Fat-predictionTest)^2)
cat("The MSE for training is:",MSE_training,"\n")

```

```
> cat("The MSE for training is:",MSE_training,"\n")
The MSE for training is: 0.005709117
```

```
cat("The MSE for testing is:",MSE_testing,"\n")

```

```
> cat("The MSE for testing is:",MSE_testing,"\n")
The MSE for testing is: 722.4294
```

Q. Comment on the quality of fit and prediction and therefore on the quality of model

The training data exhibits a notably low Mean Squared Error (MSE) of 0.0057, indicating that the model performs well on the training set. However, the test data tells a different story, with a considerably higher MSE of 722.429. This substantial difference between training and test MSE suggests that the model's performance does not generalize effectively to new, unseen data. In summary, the overall model performance is less than satisfactory.

1b. 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.
β should be optimized,the formula is below:

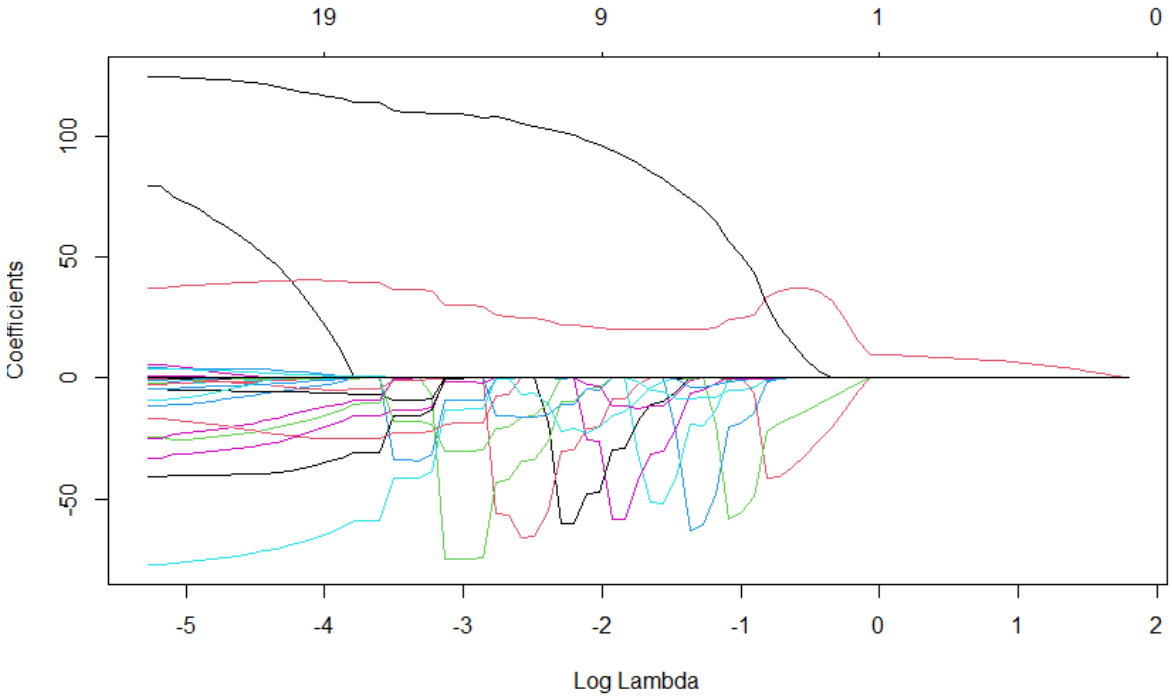
$$argmin[\frac{1}{N}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|]$$

In this formula, λ a penalty factor and it is bigger than 0. N is the number of the observations. β is the parameters.

```
#cost function

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

```

1c. Fit the LASSO regression model to the training data. Present a plot illustrating how the regression coefficients depend on the log of penalty factor (log$\lambda\lambda$) and interpret this plot.

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

#Task 3
#alpha=1 is the Lasso penalty in glmnet function
#install.packages("glmnet")
library(glmnet)
lassoModel <- glmnet(x = as.matrix(covariates),y=response,alpha=1,family="gaussian")
plot(lassoModel, xvar="lambda")

```

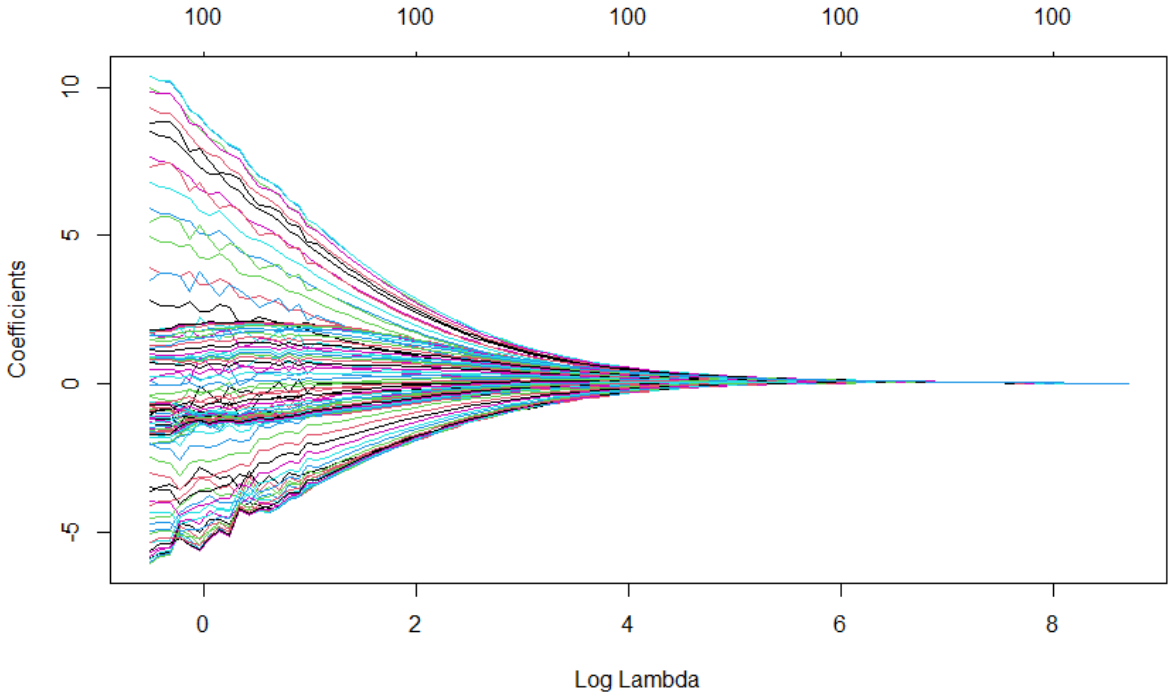


Q. What value of the penalty factor can be chosen if we want to select a model with only three features?

As the logarithm of λ increases, the coefficients of the parameters gradually approach zero, leading to a reduction in the non-zero coefficients. Based on the figure, it is observed that when the logarithm of λ is equal to -0.2, the number of features is reduced to 3. Therefore, a suitable choice for λ is determined to be 0.8187308.

1d. Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4. Conclusions?

```
#Task 4
#alpha=0 is the ridge penalty in glmnet function

ridgeModel <- glmnet(x= as.matrix(covariates), y=response, alpha=0, family="gaussian")
plot(ridgeModel,xvar = "lambda")

```

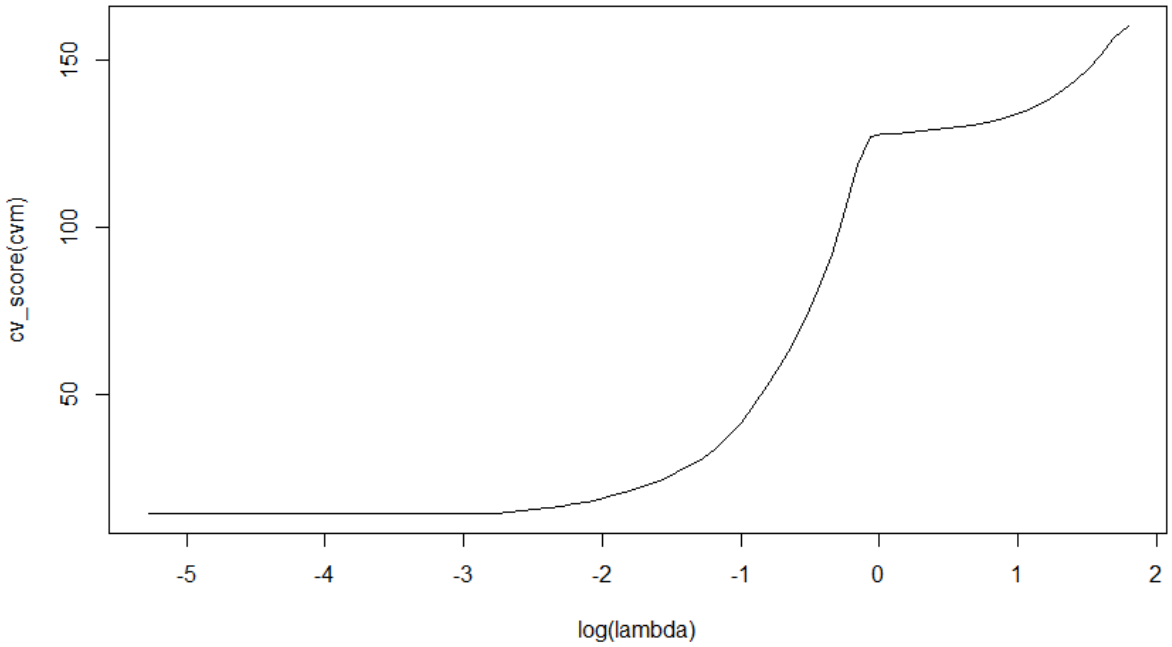


In the Lasso regression plot, the coefficients of features are observed to gradually become exactly equal to 0 as the regularization strength increases (likely controlled by the penalty parameter, λ). Lasso regularization has a feature selection property, effectively driving some coefficients to zero, leading to sparsity in the model.

In the Ridge regression plot, the coefficients of features are noted to be gradually converging to 0 as the regularization strength increases. However, unlike Lasso, the coefficients in Ridge regression do not become exactly equal to 0. Ridge regularization tends to shrink coefficients towards zero but doesn't result in complete sparsity.

1e. Use cross-validation with the default number of folds to compute the optimal LASSO model. Present a plot showing the dependence of the CV score on log$\lambda\lambda$ and comment on how the CV score changes with log$\lambda\lambda$. Report the optimal $\lambda\lambda$ and how many variables were chosen in this model.

```
model_cv<-cv.glmnet(as.matrix(covariates),
                    response,
                    family = "gaussian",
                    alpha = 1)
plot(x=log(model_cv$lambda),
     y=model_cv$cvm,
     type='l',
     xlab="log(lambda)",
     ylab ="cv_score(cvm)")

```




```
1. bestLambda <- model_cv$lambda.min
2. cat("The optimal lambda value is:",bestLambda, "\n")
3.
```

```
> bestLambda <- model_cv$lambda.min
> cat("The optimal lambda value is:",bestLambda, "\n")
The optimal lambda value is: 0.05744535
```

```
1. coefficientNumber <- lassoModel$df
2. cat("Variables that were chosen in this model is:",length(coefficientNumber))
3.
```

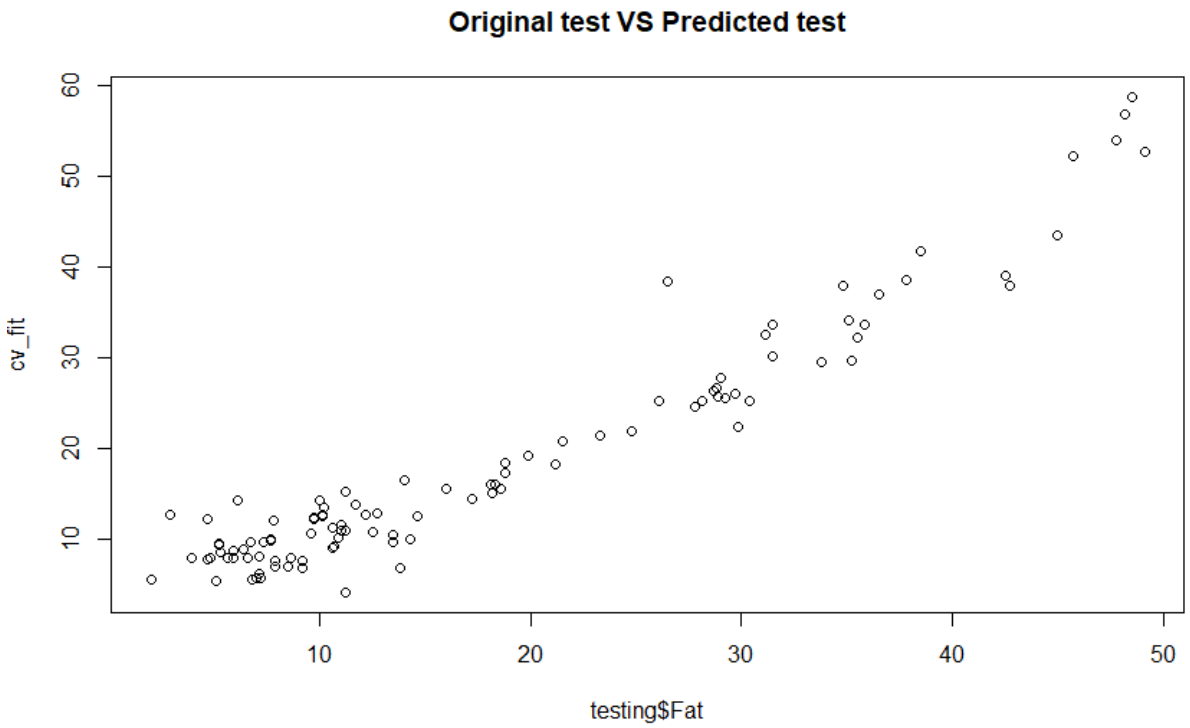
```
> coefficientNumber <- lassoModel$df
> cat("Variables that were chosen in this model is:",length(coefficientNumber))
Variables that were chosen in this model is: 77
```

To identify the optimal Lasso model, a cross-validation technique is employed. The graphical representation illustrates the algorithm's progression in selecting a significant subset of features, with all 77 variables being chosen. As the value of lambda increases, the Mean Squared Error (MSE) exhibits a rising trend. Initially, it remains stable and then starts increasing within the range of log lambda. Hence, optimal lambda shows no statistic difference with log lambda=-4.

```
1. predict_testing_new <- predict(lassoModel, as.matrix(testing[1:100]))
2. MSE_testing_new <- mean((predict_testing_new - testing[,101])^2)
3. cat("With the optimal lambda , the test MSE is",MSE_testing_new)
4.
```

```
> predict_testing_new <- predict(lassoModel, as.matrix(testing[1:100]))
> MSE_testing_new <- mean((predict_testing_new - testing[,101])^2)
> cat("With the optimal lambda , the test MSE is",MSE_testing_new)
With the optimal lambda , the test MSE is 52.1125
```

```
1. x_test_covariates <- testing[,1:100]
2. cv_fit <- predict(model_cv,as.matrix(x_test_covariates))
3. plot(x=testing$Fat,y=cv_fit, main="Original test VS Predicted test")
```



A scatter plot is employed to visually represent the linear association between the dependent (response) variable and the independent (predictor) variables. Upon examination of the scatter plot, it is evident that as the test values increase, there is a corresponding gradual rise in the predicted test values. The points on the plot clearly demonstrate a best-fitting trend, indicating that the model predictions align well with the observed data. Hence, the model's predictive performance is deemed effective.

Assignment 2: Decision tree and logistic regression for bank marketing.

2a. Fit decision tree on training data with default settings

```
#a. Decision tree with default settings
defaulttree <- tree(y~ ., data = trainingData)
```

2b. Fit decision tree on training data with smallest allowed node size equal to 7000.

```
#b.Decision tree with smallest allowed node size equal to 7000.
tree_min_node_size <- tree(y~., data = trainingData, minsize = 7000)
```

2c. Fit decision tree on training data with minimum deviance to 0.0005.

```
#c. Decision tree with
tree_with_deviance <- tree(y~., data = trainingData, mindev = 0.0005)
```

Misclassification rates for training and validation data

```
#Prediction on training data
trainPredictDefault<- predict(defaulttree, trainingData, type = "class")
trainPredictSmallest <-predict(tree_min_node_size, trainingData, type = "class")
trainPredictMindev <-predict(tree_with_deviance, trainingData, type = "class")

# predicting on validation data
validPredictDefault <- predict(defaulttree, validationData , type = "class")
validPredictSmallest <- predict(tree_min_node_size, validationData, type = "class")
validPredictMindev <- predict(tree_with_deviance, validationData, type = "class")

# calculate the misclassification rate
length(validationData$y)

misclassified <- function(Y,Y1){
  n<-length(Y)
  return(1-(sum(diag(table(Y,Y1))))/n))
}

mscTrainDefault <- misclassified(trainingData$y, trainPredictDefault)
mscTrainSmallest <- misclassified(trainingData$y, trainPredictSmallest)
mscTrainMindev <- misclassified(trainingData$y, trainPredictMindev)
mscValidDefault <- misclassified(validationData$y, validPredictDefault)
mscValidSmallest <- misclassified(validationData$y, validPredictSmallest)
mscValidMindev <- misclassified(validationData$y, validPredictMindev)

MSE_Matrix<-matrix(c(mscTrainDefault,mscTrainSmallest,mscTrainMindev,
                     mscValidDefault,mscValidSmallest,mscValidMindev),
                   nrow =3, ncol = 2)
colnames(MSE_Matrix) <-c("Training_data",'Validation_data')
rownames(MSE_Matrix) <-c("Default_Tree","Tree_min_node","Tree_with_deviance")
print(MSE_Matrix)
```

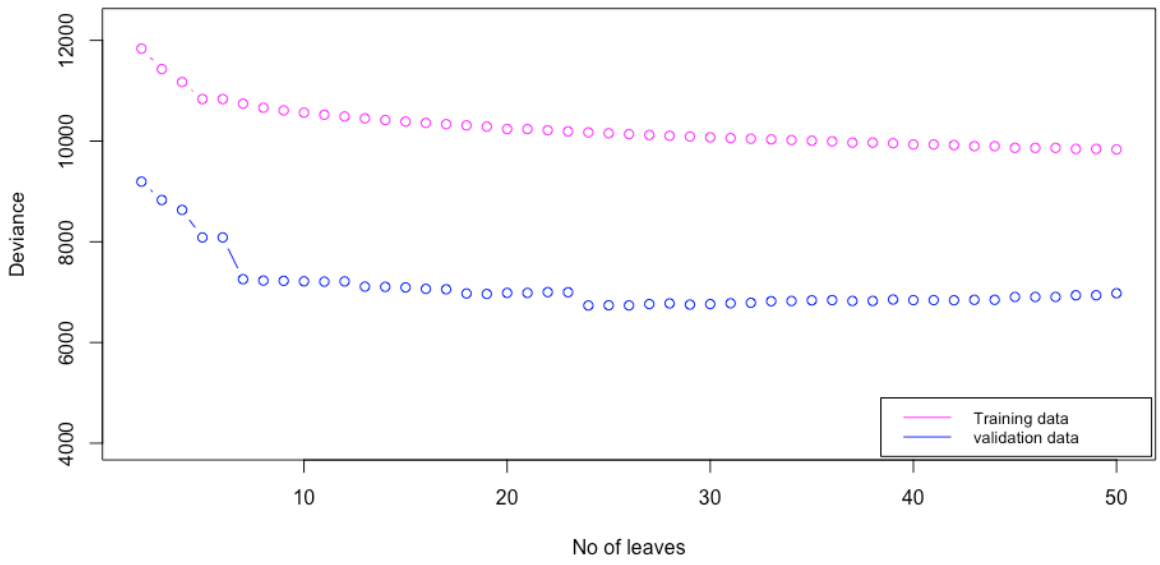
Output is as follow:

```
> print(MSE_Matrix)
                  Training_data Validation_data
Default_Tree          0.10484406       0.1092679
Tree_min_node         0.10484406       0.1092679
Tree_with_deviance    0.09560938       0.1138391
```

The above misclassification rates show us that there is no much difference between the misclassification rates of training and validation data sets. We can clearly see that the tree with minimum deviance as 0.0005 classify better than the tree with default settings and smallest node size as 7000.

3**. 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 trade-off. 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).**

```
tree_with_deviance <- tree(y~., data = trainingData, mindev = 0.0005)
trainDeviance <- seq(1,50)
validDeviance <-seq(1,50)

for(i in 2:50){
  prunedTree <- prune.tree(tree_with_deviance, best=i)
  pred_valid <- predict(prunedTree,validationData, "tree")
  trainDeviance[i] = deviance(prunedTree)
  validDeviance[i] = deviance(pred_valid)
}

plot(2:50, trainDeviance[2:50], type = "b", col="magenta",ylim=c(4000,12300),
     xlab = "No of leaves", ylab = "Deviance")
points(2:50, validDeviance[2:50], type = "b", col = "blue")
legend(38.4, 4900, legend=c("Training data", "validation data"),
       col=c("magenta", "blue"),lty=1:1, cex=0.8)
```



The graph shows the dependence of deviance for the training and validation data sets on the number of leaves for the decision tree with minimum deviance as 0.0005. If the model has a smaller number of leaves, then tree show high bias and less variance which termed as underfitting. Whereas when the tree has high number of leaves, it shows low bias and high variance which is overfitting the tree.

Optimal amount of leaves and variables seems to be most important for decision making.

```
optimal_no_leaves <- which.min(validDeviance[-1])
optimal_no_leaves
optimalTree<- prune.tree(tree_with_deviance, best=optimal_no_leaves)
summary(optimalTree)
```

Output is as follow:

```
> optimal_no_leaves <- which.min(validDeviance[-1])
> optimal_no_leaves
[1] 23
> optimalTree<- prune.tree(tree_with_deviance, best=optimal_no_leaves)
> summary(optimalTree)

Classification tree:
snip.tree(tree = tree_with_deviance, 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"      "campaign"
[9] "housing"
Number of terminal nodes:  23
Residual mean deviance:  0.5644 = 10190 / 18060
Misclassification error rate: 0.1027 = 1857 / 18084
```

This means tree with 23 leaves give the minimum deviance when evaluated with validation data.

The most important information provided by tree and variables most important for decision making are “poutcome”, “month”, “contact”, “day”, “age”, “pdays”, “job”, “campaign” and “housing”.

4. **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(optimalTree, testingData,"class")
#confusion metrix
confusion<- table(testingData$y,pred_test)
confusion
```

Confusion matrix for the above prediction is as follow

```
> confusion
   pred_test
        0     1
  0 11782   197
  1  1251   334
```

Accuracy and F1 score is calculated as follow:

```
#accuracy
1-misclassified(testingData$y,pred_test)
#F1
recall<-confusion[2,2]/sum(confusion[,2])
precision<-confusion[2,2]/sum(confusion[2,])
F1Score <-2*precision*recall/(precision+recall)
F1Score
```

Output is :

```
> #accuracy
> 1-misclassified(testingData$y,pred_test)
[1] 0.8932468
> F1Score
[1] 0.31569
```

Accuracy of the tree with test data is 0.8932468. While F1 score with test data is 0.31569. Since the data has uneven distribution of class, F1 score with test data should be consider to determine the predictive power of the tree.as F1 score is 0.315, it clearly states that tree has high number of false positive. So this tree will not be a good for prediction.

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.

```
1. library(rpart)
2. L <- matrix(c(0,5,1,0), nrow=2)
3. decisionTree<- rpart(y~., data=trainingData, method="class", parms=list(loss=L))
4. pred_decisionTree <- predict(decisionTree, testingData, type="class")
5. confusion_Loss <-table(testingData$y,pred_decisionTree)
6. confusion_Loss
```

Confusion matric for the above decision tree is:

```
> confusion_Loss
   pred_decisionTree
       0     1
  0 10910  1069
  1   836   749
```

Accuracy and F1 score of calculated as follow:

```
1. #accuracy
2. 1-misclassified(testingData$y, pred_decisionTree)
3. #F1 score
4. recall_decisiontree<- confusion_Loss[2,2]/sum(confusion_Loss[,2])
5. precision_decisiontree <- confusion_Loss[2,2]/sum(confusion_Loss[2,])
6. F1_decisiontree<- 2*precision_decisiontree*recall_decisiontree/
7.   (precision_decisiontree+recall_decisiontree)
8. F1_decisiontree
```

Output is:

```
> #accuracy
> 1-misclassified(testingData$y, pred_decisionTree)
[1] 0.8595547
> F1_decisiontree
[1] 0.4401998
```

In comparison of results from step 4, accuracy of this tree is has come down to 85.95% while the F1 score is increased to 0.44 from 0.315. In this tree, a loos matrix is used to penalize each misclassification. According to loss matrix false positive is penalize five times than the false negative, which changes the way of splitting such that loss of making incorrect prediction 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 = nowhere 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?

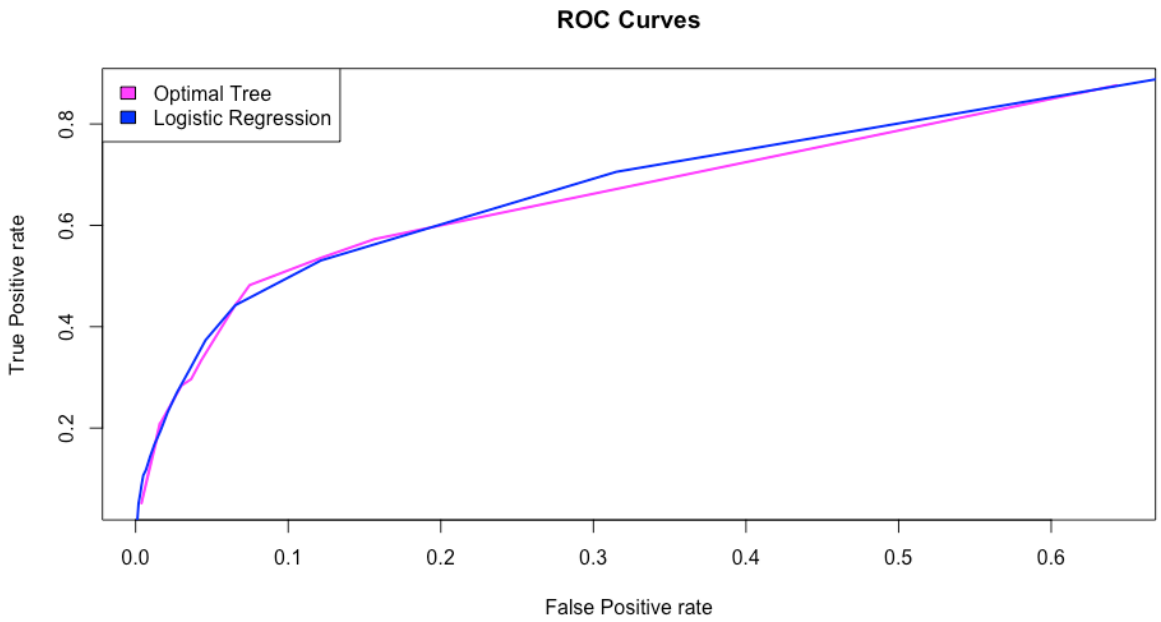
```
1. #Pred with optimal tree
2. pred_Optimal_test <-predict(optimalTree, testingData, type="vector")
3. #logistic regression
4. lRegression <- glm(y~.,trainingData, family = binomial)
5. lr_pred <- predict(lRegression, testingData,type="response")
6. #TPR adn FPR values for different values of treshold (pi)
7. TPR_tree<-c()
8. FPR_tree<-c()
9. TPR_LR<-c()
```

```
FPR_LR<-c()

j<-1
for (i in seq(0.05,0.95,0.05)){
  temporary_tree <- ifelse(pred_Optimal_test[,2]>i,1,0)
  temporary_LR <- ifelse(lr_pred>i,1,0)
  t1<-table(testingData$y,temporary_tree)
  if(dim(t1)[2]>1){
    TPR_tree[j] <- t1[2,2]/sum(t1[2,])
    FPR_tree[j] <- t1[1,2]/sum(t1[1,])
  }
  t2<-table(testingData$y, temporary_LR)
  if(dim(t2)[2]>1){
    TPR_LR[j] <- t2[2,2]/sum(t2[2,])
    FPR_LR[j] <- t2[1,2]/sum(t2[1,])
  }
  j=j+1
}
```

Plotting the ROC curve

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

```

The ROC curve is as follow:



As we can see that the data under both the curve is almost same in both the curves. However, the area under the curve of logistic regression is slightly high than the optimal tree model. Thus, logistic regression will be preferred over the tree model. As per my understanding as the data has unbalanced classes, Precision – Recall curve will be more suitable for this discussion.

Assignment 3: Principal components and implicit regularization

Q1.

After scaling the data we can compute the variance by computing the eigen values of the covariance matrix. Then we divide each variance by the sum of all variances to calculate the percentage. From runing the below code we can get that we need 35 components to reach 95% accuracy.

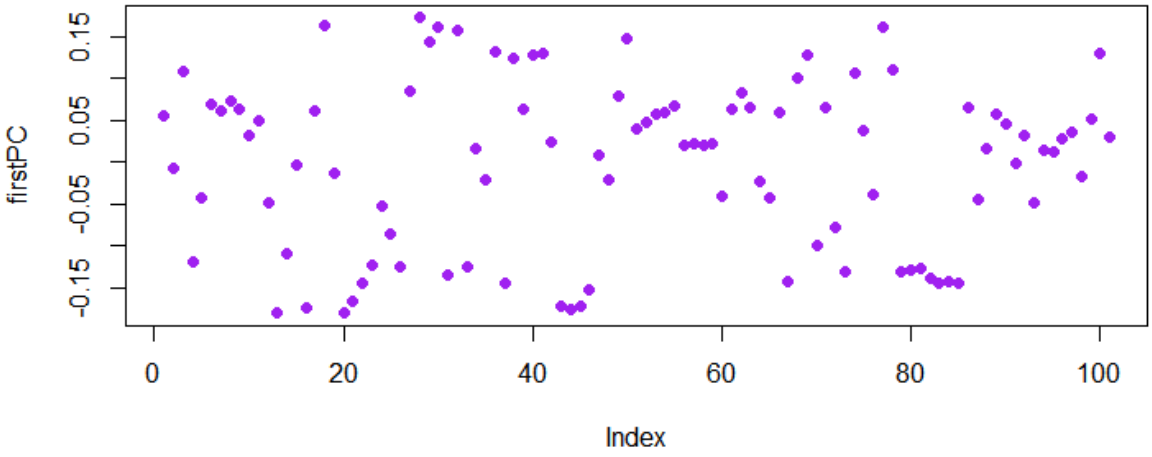
```
1. cumsum_percentage=cumsum(pca_variation$values / sum(pca_variation$values)*100)
2. which(cumsum_percentage >= 95)[1]
```

By running the code below, we can find that the first two principle components explain 25.44859 and 16.88074 percent of the data.

```
1. #first PCA
2. pca_variation$values[1]
3. #second PCA
4. pca_variation$values[2]
```

Q2.

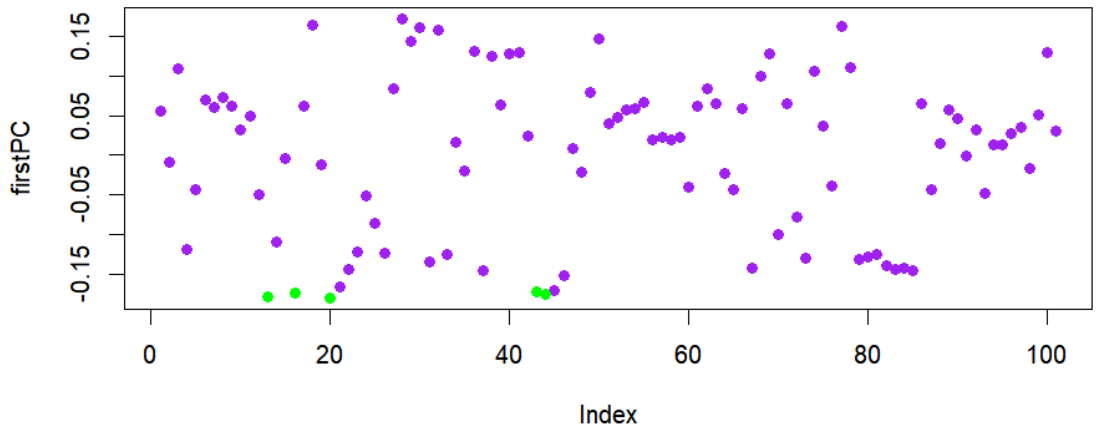
The trace plot of the first principal component is like:



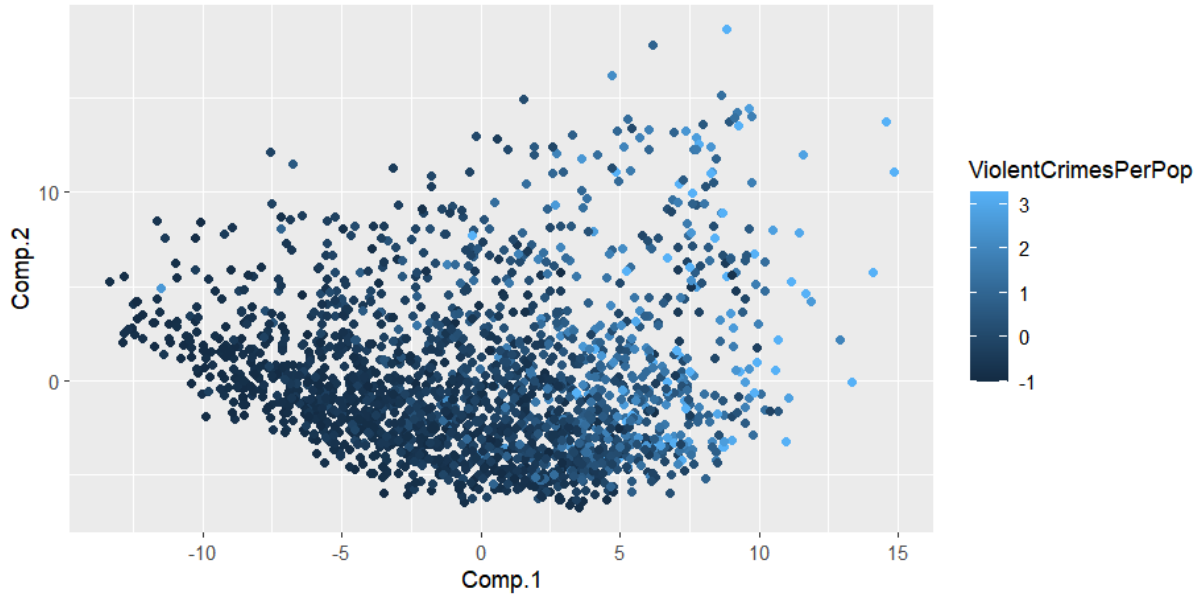
The features that have the most impact on first pc are:

- **MedIncome:** median houshold incom
- **PctWInvInc:** percentage of households with investment
- **MedFamInc:** median family income (differs from household income for non-family households)
- **PctFam2Par:** percentage of families (with kids) that are headed by two parents
- **PctKids2Par:** percentage of kids in family housing with two parents

Most of these features are economical so money is the reason behind many of the crimes. As we see in the plot below all these features are negative so crime will increase as the income decrease or one of the parents become the only provider.



In the diagram below we can see that the cp1 is more representative of ViolentCrimesPerPop since as the value of the cp1 increases ViolentCrimesPerPop increases as well. But this isn't the case with cp2.

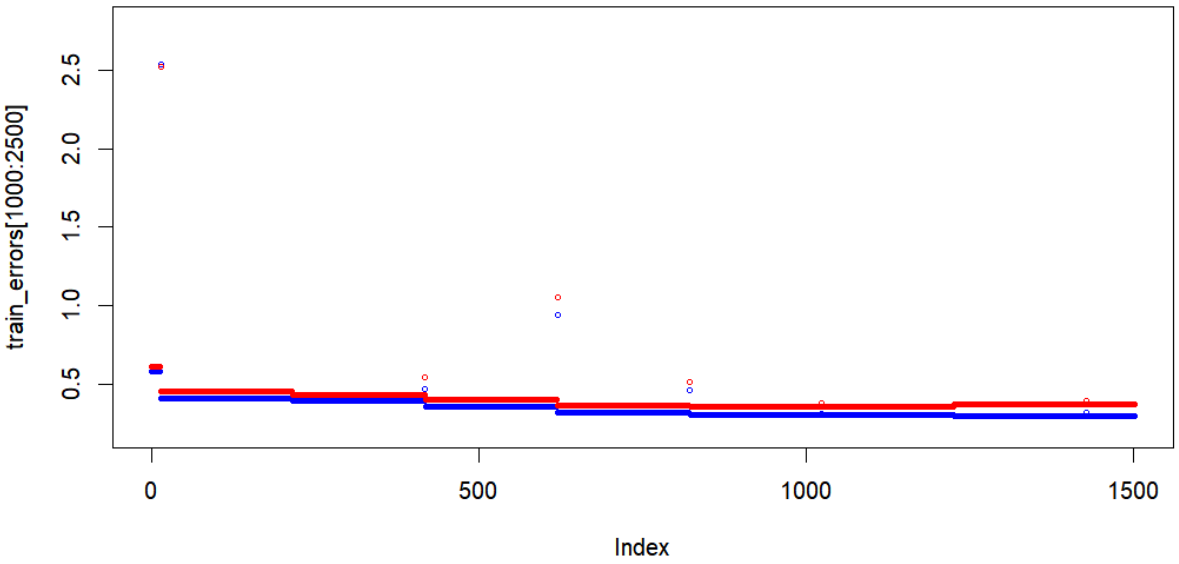


Q3.

MSE for the training data set is 0.27 and for the testing data set is 0.38. So the training error is less than testing error but there is no huge gap, so the model is not underfitting or over fitting and acceptable.

Q4.

We use MSE to calculate the cost in our function. For comparing the two models we should compare the errors in the best(optimal) model with the errors we got from the previous question. The training error for this model is 0.275 and the testing error is 0.382 which is pretty much the same as the errors we calculated in the previous question and that is good sign.



Iteration values are as followed:

```
initial  value 0.998997
iter  10 value 0.308219
iter  20 value 0.280144
iter  30 value 0.276877
iter  40 value 0.276017
iter  50 value 0.275680
iter  60 value 0.275499
iter  70 value 0.275372
iter  80 value 0.275296
iter  90 value 0.275243
iter 100 value 0.275221
final  value 0.275221
```

So it might be a good idea to stop the iterations after 50 since we have almost reached the optimal value.

Appendix:

Code for assignment 1:

```
fileName <- read.csv("tecator.csv")
#Dividing the data(50/50)
noOfrows <- nrow(fileName)
set.seed(12345)
id <- sample(1:noOfrows,floor(noOfrows*0.5))
training <- fileName[id, ]
testing <- fileName[-id, ]
#install.packages("dplyr")
library(dplyr)
training <- training %>% select(2:102)
testing <- testing %>% select(2:102)
#Fit the linear regression model
linear_model <- lm(Fat~.,training)
summary(linear_model)
predictionsTrain <- predict(linear_model, training)
predictionTest <- predict(linear_model,testing)
#MSE of Training and testing
MSE_training <- mean((training$Fat-predictionsTrain)^2)
MSE_testing <- mean((testing$Fat-predictionTest)^2)
cat("The MSE for training is:",MSE_training,"\n")
cat("The MSE for testing is:",MSE_testing,"\n")

#Task 2
#cost function

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

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

#Task 3
#alpha=1 is the Lasso penalty in glmnet function
#install.packages("glmnet")
library(glmnet)
lassoModel <- glmnet(x = as.matrix(covariates),y=response,alpha=1,family="gaussian")
plot(lassoModel, xvar="lambda")


#Task 4
#alpha=0 is the ridge penalty in glmnet function

ridgeModel <- glmnet(x= as.matrix(covariates), y=response, alpha=0, family="gaussian")
plot(ridgeModel,xvar = "lambda")

#Task 5
model_cv<-cv.glmnet(as.matrix(covariates),
                    response,
                    family = "gaussian",
                    alpha = 1)
plot(x=log(model_cv$lambda),
     y=model_cv$cvm,
     type='l',
     xlab="log(lambda)",
     ylab ="cv_score(cvm)")
```

```
bestLambda <- model_cv$lambda.min
cat("The optimal lambda value is:",bestLambda, "\n")
coefficientNumber <- lassoModel$df
cat("Variables that were chosen in this model is:",length(coefficientNumber))
predict_testing_new <- predict(lassoModel, as.matrix(testing[1:100]))
MSE_testing_new <- mean((predict_testing_new - testing[,101])^2)
cat("With the optimal lambda , the test MSE is",MSE_testing_new)
#df <- data.frame(value_test=c(testing[,101]),y=c(predict_testing_new))
#install.packages("ggplot2")
#library(ggplot2)
#ggplot(df,aes(x=value_test,y=y))+geom_point()


x_test_covariates <- testing[,1:100]
cv_fit <- predict(model_cv,as.matrix(x_test_covariates))
plot(x=testing$Fat,y=cv_fit, main="Original test VS Predicted test")


```

Code for assignment 2:

```
#install.packages("tree")
library(tree)
library(tidyr)
library(tidyverse)
#import data, clean it properly and divide into 40/30/30.
bankdata <- read.csv("bank-full.csv")
col_Names <- colnames(bankdata)
col_Names <- unlist(str_split(col_Names, "[.]"))
col_Names
bankdata1 <- separate(bankdata,1,col_Names,sep = ";")
bankdata1 <- bankdata1[,-6] # removing "balance" as it is not part of input variables.

bankdata1$age <-as.numeric(bankdata1$age)
bankdata1$job <- as.factor(bankdata1$job)
bankdata1$marital <- as.factor(bankdata1$marital)
bankdata1$education <- as.factor(bankdata1$education)
bankdata1$default <- as.factor(bankdata1$default)
bankdata1$housing <- as.factor(bankdata1$housing)
bankdata1$loan <- as.factor(bankdata1$loan)
bankdata1$contact <- as.factor(bankdata1$contact)
bankdata1$month <- as.factor(bankdata1$month)
bankdata1$day <- as.factor(bankdata1$day)
bankdata1$duration <- as.numeric(bankdata1$duration)
bankdata1$campaign <- as.numeric(bankdata1$campaign)
bankdata1$pdays <- as.numeric(bankdata1$pdays)
bankdata1$previous <- as.numeric(bankdata1$previous)
bankdata1$poutcome <- as.factor(bankdata1$poutcome)
bankdata1$y <- ifelse(bankdata1$y=='yes',1,0) # 1 for yes 0 for no.
bankdata1$y <- as.factor(bankdata1$y)
# drop duration column as its not in use
data_cleaned <- bankdata1[, -which(names(bankdata1) %in% c("duration"))]
tail(data_cleaned)
head(data_cleaned)

n <- dim(data_cleaned)[1]
set.seed(12345)
index <- sample(1:n, floor(n*0.4))
trainingData = data_cleaned[index,]
index1 <- setdiff(1:n, index)
set.seed(12345)
index2 <- sample(index1, floor(n*0.3))
validationData = data_cleaned[index2,]
```

```
index3<-setdiff(index1,index2)
testingData = data_cleaned[index3,]

#define 3 different trees
library(tree)
#a. Decision tree with default settings
defaulttree <- tree(y~ ., data = trainingData)

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

#c. Decision tree with
tree_with_deviance <- tree(y~., data = trainingData, mindev = 0.0005)

#Prediction on training data
trainPredictDefault<- predict(defaulttree, trainingData, type = "class")
trainPredictSmallest <-predict(tree_min_node_size, trainingData, type = "class")
trainPredictMindev <-predict(tree_with_deviance, trainingData, type = "class")

# predicting on validation data
validPredictDefault <- predict(defaulttree, validationData , type = "class")
validPredictSmallest <- predict(tree_min_node_size, validationData, type = "class")
validPredictMindev <- predict(tree_with_deviance, validationData, type = "class")

# calculate the misclassification rate
length(validationData$y)

misclassified <- function(Y,Y1){
  n<-length(Y)
  return(1-(sum(diag(table(Y,Y1))))/n))
}

mscTrainDefault <- misclassified(trainingData$y, trainPredictDefault)
mscTrainSmallest <- misclassified(trainingData$y, trainPredictSmallest)
mscTrainMindev <- misclassified(trainingData$y, trainPredictMindev)
mscValidDefault <- misclassified(validationData$y, validPredictDefault)
mscValidSmallest <- misclassified(validationData$y, validPredictSmallest)
mscValidMindev <- misclassified(validationData$y, validPredictMindev)

MSE_Matrix<-matrix(c(mscTrainDefault,mscTrainSmallest,mscTrainMindev,
                     mscValidDefault,mscValidSmallest,mscValidMindev),
                   nrow =3, ncol = 2)
colnames(MSE_Matrix) <-c("Training_data",'Validation_data')
rownames(MSE_Matrix) <-c("Default_Tree","Tree_min_node","Tree_with_deviance")
print(MSE_Matrix)

#studying the tree upto 50 leaves in 2c.
tree_with_deviance <- tree(y~., data = trainingData, mindev = 0.0005)
trainDeviance <- seq(1,50)
validDeviance <-seq(1,50)

for(i in 2:50){
  prunedTree <- prune.tree(tree_with_deviance, best=i)
  pred_valid <- predict(prunedTree,validationData, "tree")
  trainDeviance[i] = deviance(prunedTree)
  validDeviance[i] = deviance(pred_valid)
}

plot(2:50, trainDeviance[2:50], type = "b", col="magenta",ylim=c(4000,12300),
     xlab = "No of leaves", ylab = "Deviance")
points(2:50, validDeviance[2:50], type = "b", col = "blue")
legend(38.4, 4900, legend=c("Training data", "validation data"),
       col=c("magenta", "blue"),lty=1:1, cex=0.8)
```

```
106.
107. optimal_no_leaves <- which.min(validDeviance[-1])
108. optimal_no_leaves
109. optimalTree<- prune.tree(tree_with_deviance, best=optimal_no_leaves)
110. summary(optimalTree)
111. #Prediction with test data
112. pred_test <- predict(optimalTree, testingData,"class")
113. #confusion metrix
114. confusion<- table(testingData$y,pred_test)
115. confusion
116. #accuracy
117. 1-misclassified(testingData$y,pred_test)
118. #F1
119. recall<-confusion[2,2]/sum(confusion[,2])
120. precision<-confusion[2,2]/sum(confusion[2,])
121. F1Score <-2*precision*recall/(precision+recall)
122. F1Score
123. library(rpart)
124. L <- matrix(c(0,5,1,0), nrow=2)
125. decisionTree<- rpart(y~., data=trainingData, method="class", parms=list(loss=L))
126. pred_decisionTree <- predict(decisionTree, testingData, type="class")
127. confusion_Loss <-table(testingData$y,pred_decisionTree)
128. confusion_Loss
129. #accuracy
130. 1-misclassified(testingData$y, pred_decisionTree)
131. #F1 score
132. recall_decisiontree<- confusion_Loss[2,2]/sum(confusion_Loss[,2])
133. precision_decisiontree <- confusion_Loss[2,2]/sum(confusion_Loss[2,])
134. F1_decisiontree<- 2*precision_decisiontree*recall_decisiontree/
135.   (precision_decisiontree+recall_decisiontree)
136. F1_decisiontree
137. #Pred with optimal tree
138. pred_Optimal_test <-predict(optimalTree, testingData, type="vector")
139. #logistic regression
140. lRegression <- glm(y~.,trainingData, family = binomial)
141. lr_pred <- predict(lRegression, testingData,type="response")
142. #TPR adn FPR values for different values of treshold (pi)
143. TPR_tree<-c()
144. FPR_tree<-c()
145. TPR_LR<-c()
146. FPR_LR<-c()
147.
148. j<-1
149. for (i in seq(0.05,0.95,0.05)){
150.   temporary_tree <- ifelse(pred_Optimal_test[,2]>i,1,0)
151.   temporary_LR <- ifelse(lr_pred>i,1,0)
152.   t1<-table(testingData$y,temporary_tree)
153.   if(dim(t1)[2]>1){
154.     TPR_tree[j] <- t1[2,2]/sum(t1[2,])
155.     FPR_tree[j] <- t1[1,2]/sum(t1[1,])
156.   }
157.   t2<-table(testingData$y, temporary_LR)
158.   if(dim(t2)[2]>1){
159.     TPR_LR[j] <- t2[2,2]/sum(t2[2,])
160.     FPR_LR[j] <- t2[1,2]/sum(t2[1,])
161.   }
162.   j=j+1
163. }
164. plot(FPR_tree,TPR_tree, type="l", col="magenta",lwd=2,
165.      xlab ="False Positive rate", ylab="True Positive rate", main="ROC Curves")
166. lines(FPR_LR, TPR_LR, type = "l", col = "blue", lwd=2)
167. legend("topleft", c("Optimal Tree", "Logistic Regression"),
168.        fill = c("magenta", "blue"))
```

Code for assignment 3:

```
library("ggplot2")

data=read.csv("communities.csv",header = TRUE)
############## Question1 ##############

#normalizing variebles
scaled_data=as.data.frame(cbind(scale(data[-data$ViolentCrimesPerPop]),
                        data$ViolentCrimesPerPop))

#applying PCA
pca=prcomp(scaled_data,scale. = FALSE)
pca_variation= eigen(cov(scaled_data))
print(cov(scaled_data))

cumsum_percentage=cumsum(pca_variation$values / sum(pca_variation$values)*100)

which(cumsum_percentage >= 95)[1]
#first PCA
pca_variation$values[1]
#second PCA
pca_variation$values[2]
############## Question2 ##############
pca2=princomp(scaled_data)
firstPC=pca2$loadings[,1]
plot(firstPC,col="purple",pch=19)

top_feature=which(abs(firstPC) %in% head(sort(abs(firstPC),
                                  decreasing = TRUE),5))
points(top_feature,pca2$loadings[top_feature,1],col="green",pch=19)

dtp=data.frame("ViolentCrimesPerPop"=scaled_data$ViolentCrimesPerPop,
               pca2$scores[,1:2])
ggplot(data = dtp) +
  geom_point(aes(x = Comp.1, y = Comp.2, col = ViolentCrimesPerPop))

############## Question3 ##############

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

LR=lm(ViolentCrimesPerPop~.,data=train)
train_pred=predict(LR,type="response")
test_pred=predict(LR,test,type="response")

train_error=mean((train$ViolentCrimesPerPop-train_pred)^2)
test_error=mean((test_pred-test$ViolentCrimesPerPop)^2)

############## Question4 ##############
# Divide data

index=which(colnames(train)=="ViolentCrimesPerPop")
train_x=as.matrix(train[,-index])
test_x= as.matrix(test[,-index])
train_y=train$ViolentCrimesPerPop
test_y=test$ViolentCrimesPerPop

cost_function=function(theta){
```

```
62.
63.   mse_train=mean((train_y - train_x %*% theta)^2)
64.   mse_test=mean((test_y - test_x %*% theta)^2)
65.
66.   train_errors<<-c(train_errors,mse_train)
67.   test_errors<<-c(test_errors,mse_test)
68.
69.   return(mse_train)
70. }
71.
72. test_errors=c()
73. train_errors=c()
74.
75. theta=rep(0, 100)
76. opt=optim(par = theta, fn = cost_function, method = "BFGS",
77.           control=list(trace=TRUE))
78. plot(train_errors[1000:2500],col="blue",ylim = c(0.2,2.8), cex=0.5)
79. points(test_errors[1000:2500],col="red", cex=0.5)
80.
81. train_error_cusfunc=train_errors[which.min(train_errors)]
82. test_error_cusfunc=test_errors[which.min(train_errors)]
83.
84. train_error_cusfunc
85. test_error_cusfunc
86.
87.
```