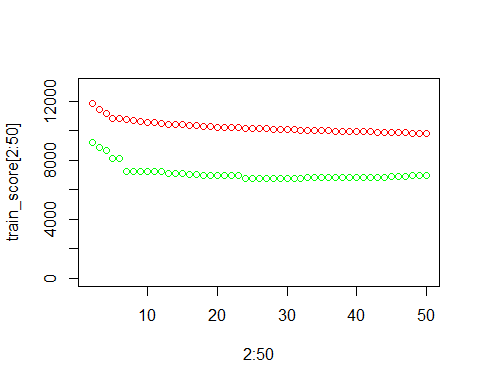
setwd("C:/Masters/TDDE01-Machine Learning/Lab/Lab2")  
library(tidyr)  
library(tree)  
library(rpart)  
bank\_data <- read.csv("bank-full.csv")  
column\_names <- colnames(bank\_data)   
column\_names <- unlist(strsplit(column\_names,"[.]")) #fetching column names from bank data  
  
new\_bank\_data <- separate(bank\_data,1, column\_names, sep =";") #data formatting  
  
# removing column "balance" from data, as not part of input variable  
new\_bank\_data <- new\_bank\_data[,-6]   
  
  
new\_bank\_data$age <- as.numeric(new\_bank\_data$age)  
new\_bank\_data$job <- as.factor(new\_bank\_data$job)  
new\_bank\_data$marital <- as.factor(new\_bank\_data$marital)  
new\_bank\_data$education <- as.factor(new\_bank\_data$education)  
new\_bank\_data$default <- as.factor(new\_bank\_data$default)  
new\_bank\_data$housing <- as.factor(new\_bank\_data$housing)  
new\_bank\_data$loan <- as.factor(new\_bank\_data$loan)  
new\_bank\_data$contact <- as.factor(new\_bank\_data$contact)  
new\_bank\_data$month <- as.factor(new\_bank\_data$month)  
new\_bank\_data$day <- as.factor(new\_bank\_data$day)  
new\_bank\_data$duration <- as.numeric(new\_bank\_data$duration)  
new\_bank\_data$campaign <- as.numeric(new\_bank\_data$campaign)  
new\_bank\_data$pdays <- as.numeric(new\_bank\_data$pdays)  
new\_bank\_data$previous <- as.numeric(new\_bank\_data$previous)  
new\_bank\_data$poutcome <- as.factor(new\_bank\_data$poutcome)  
new\_bank\_data$y <- ifelse(new\_bank\_data$y =="yes", 1,0)  
new\_bank\_data$y <- as.factor(new\_bank\_data$y)  
  
#remove input variable "duration"  
processed\_bank\_data <- new\_bank\_data[, !names(new\_bank\_data) %in% c("duration")]  
  
#####Task1#####  
#dividing the data into training/validation/test  
n <- dim(processed\_bank\_data)[1]  
set.seed(12345)  
id <- sample(1:n, floor(n\*0.4))  
training <- processed\_bank\_data[id,]  
  
id1 <- setdiff(1:n, id)  
set.seed(12345)  
id2 <- sample(id1, floor(n\*0.3))  
validation <- processed\_bank\_data[id2,]  
  
id3 <- setdiff(id1,id2)  
test <- processed\_bank\_data[id3,]  
  
  
####Task2#####  
  
#Fit  
#A: Decision Tree With Default Settings  
default\_tree <- tree(y~., data = training)  
  
#Prediction with training and validation data(default tree).  
  
prediction\_deafult\_train = predict(default\_tree, training, type = "class")  
prediction\_deafult\_valid = predict(default\_tree, validation, type = "class")  
  
#B: Decision Tree with smallest allowed node size equal to 7000  
smallest\_tree <- tree(y~., data = training, control = tree.control(nobs = nrow(training), minsize = 7000))  
  
#Prediction with training and validation data(Smallest Allowed Node tree).  
  
prediction\_smallest\_train = predict(smallest\_tree, training, type = "class")  
prediction\_smallest\_valid = predict(smallest\_tree, validation, type = "class")  
  
#C: Decision Trees minimum deviance to 0.0005.  
mindev\_tree <- tree(y~., data = training, control = tree.control(nobs = nrow(training), mindev = 0.0005))  
  
#Prediction with training and validation data(Minimum deviance tree).  
  
prediction\_mindev\_train = predict(mindev\_tree, training, type = "class")  
prediction\_mindev\_valid = predict(mindev\_tree, validation, type = "class")  
  
# function to calculate misclassifications rate  
missclass\_error <- function(X,Y){  
 len <- length(X)  
 return(1 - sum(diag(table(X,Y)))/len)  
}  
  
#Calculating misclassification on training data  
  
miss\_default <- missclass\_error(training$y, prediction\_deafult\_train)  
miss\_small <- missclass\_error(training$y, prediction\_smallest\_train)  
miss\_mindev <- missclass\_error(training$y, prediction\_mindev\_train)  
  
# calculate misclassification on validation data  
  
miss\_default\_1 <- missclass\_error(validation$y, prediction\_deafult\_valid)  
miss\_small\_1 <- missclass\_error(validation$y, prediction\_smallest\_valid)  
miss\_mindev\_1 <- missclass\_error(validation$y, prediction\_mindev\_valid)  
  
missclass\_rate <- data.frame("Default\_Tree" = c(miss\_default, miss\_default\_1),  
 "Smallest\_Node" = c(miss\_small,miss\_small\_1),   
 "Minimum\_Deviance" = c(miss\_mindev,miss\_mindev\_1))  
rownames(missclass\_rate) <- c("Training","Validation")  
  
missclass\_rate

## Default\_Tree Smallest\_Node Minimum\_Deviance  
## Training 0.1048441 0.1048441 0.09544349  
## Validation 0.1092679 0.1092679 0.11398658

####Task3####  
  
train\_score <- rep(0,50)  
valid\_score <- rep(0,50)  
  
for(i in 2:50)  
{  
 pruned\_training <- prune.tree(mindev\_tree, best = i)  
 predict\_pTree <- predict(pruned\_training, validation, type = "tree")  
 train\_score[i] <- deviance(pruned\_training)  
 valid\_score[i] <- deviance(predict\_pTree)  
}  
  
plot(2:50, train\_score[2:50], type = "b", col="red", ylim = c(0, 13000))  
points(2:50, valid\_score[2:50], type = "b", col="green")

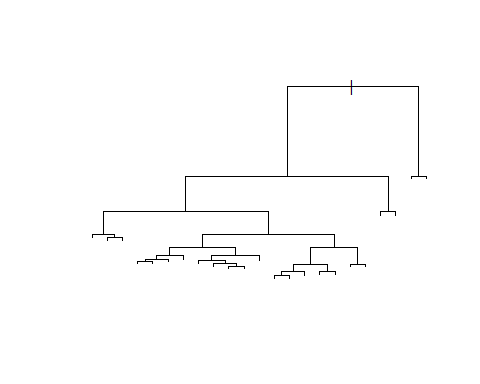


The above graph shows the dependence of deviances for the training and the validation data on the number of leaves for the model of 0.0005 minimum deviance. The lesser the terminal nodes(simple model), then the model has high bias and low variance which leads to underfitting and the greater the terminal nodes(more complex model), then the model has low bias and high variance which leads to overfitting

#optimal amount of leaves  
  
optimal\_leaf <- which.min(valid\_score[-1])  
  
optimal\_tree <- prune.tree(mindev\_tree, best = optimal\_leaf)  
summary(optimal\_tree)

##   
## Classification tree:  
## snip.tree(tree = mindev\_tree, 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

plot(optimal\_tree)



The optimal number of leaves is 23. The following features seem to be most important for decision making in the tree. “poutcome” “month” “contact” “day” “age” “pdays” “job” “campaign” “housing”.From the decision tree, the first three features (the features poutcome, month, contact) are the most decisive and the most fo the data points are classified to the flatter side.

####Task4####  
  
# test data prediction with optimal tree  
  
prediction\_test <- predict(optimal\_tree, test, type = "class" )  
  
#confusion matrix   
test\_confusionMatrix <- table(true=test$y,prediction\_test)  
test\_confusionMatrix

## prediction\_test  
## true 0 1  
## 0 11782 197  
## 1 1251 334

#Test Accuracy  
test\_cm\_diag = sum(diag(test\_confusionMatrix))  
test\_cm\_sum = sum(test\_confusionMatrix)  
  
test\_accuracy = test\_cm\_diag/test\_cm\_sum  
  
test\_accuracy

## [1] 0.8932468

The accuracy of this model is 89.32%

#F1 Score  
  
test\_recall <- test\_confusionMatrix[2,2]/sum(test\_confusionMatrix[2,])  
test\_precision <- test\_confusionMatrix[2,2] / sum(test\_confusionMatrix[,2])  
  
F1\_score <- 2\*test\_precision\*test\_recall / (test\_precision+test\_recall)  
F1\_score

## [1] 0.31569

**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.**

Since the data has uneven class distribution, F1 score shoul be used 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

####Task5####  
  
loss\_matrix <- matrix(c(0,5,1,0), nrow = 2)  
  
lost\_tree\_model <- rpart(y~., data = training, method = "class", parms = list(loss= loss\_matrix))  
  
loss\_predict\_tree <- predict(lost\_tree\_model, test, type = "class")  
  
#Confusion Matrix of Tree with Loss Matrix  
loss\_confusionMatrix <- table(true=test$y, predicted=loss\_predict\_tree)  
  
loss\_confusionMatrix

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

# Accuracy  
loss\_cm\_diag = sum(diag(loss\_confusionMatrix))  
loss\_cm\_sum = sum(loss\_confusionMatrix)  
  
loss\_accuracy = loss\_cm\_diag/loss\_cm\_sum  
loss\_accuracy

## [1] 0.8595547

The accuracy of this model is 85.95%

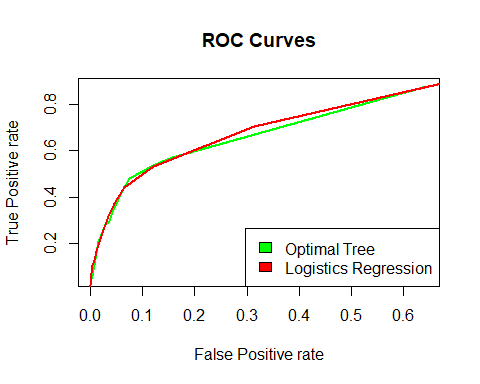
#F1 score  
  
loss\_recall <- loss\_confusionMatrix[2,2]/sum(loss\_confusionMatrix[2,])  
loss\_precision <- loss\_confusionMatrix[2,2] / sum(loss\_confusionMatrix[,2])  
  
loss\_F1\_score <- 2\*loss\_precision\*loss\_recall / (loss\_precision+loss\_recall)  
loss\_F1\_score

## [1] 0.4401998

**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?**

In comparison with the results with step 4, the accuracy of the model has decreased to 85.95% and F1 score is increased to 0.440 from 0.315. Here the loss matrix is used to to penalize for each misclassifications.We have penalized false positives five times more than the false negatives which leads to the loss of making incorrect prediction.

####Task6####  
  
optimal\_final\_tree <- prune.tree(mindev\_tree, best = 23)  
optimal\_tree\_predection <- predict(optimal\_final\_tree, test, type = "vector")  
  
#logistics regression  
logistics <- glm(y~., data = training, family = "binomial")  
logistics\_prediction <- predict(logistics, test, type = "response")  
  
#computing TPR and FPR with different thresholds  
  
tree\_TPR <- c()  
tree\_FPR <- c()  
logistics\_TPR <- c()  
logistics\_FPR <- c()  
  
k <- 1  
  
for(i in seq(0.05, 0.95, 0.05)){  
 t\_tree <- ifelse(optimal\_tree\_predection[,2]>i,1,0)  
 t\_logistics <- ifelse(logistics\_prediction>i, 1,0)  
 t1 <- table(test$y, t\_tree)  
 if(dim(t1)[2]>1){  
 tree\_TPR[k] <- t1[2,2] / sum(t1[2,])  
 tree\_FPR[k] <- t1[1,2] / sum(t1[1,])  
 }  
   
 t2 <- table(test$y, t\_logistics)  
 if(dim(t2)[2]>1){  
 logistics\_TPR[k] <- t2[2,2] / sum(t2[2,])  
 logistics\_FPR[k] <- t2[1,2] / sum(t2[1,])  
 }  
 k=k+1  
}  
  
#plotting ROC curves  
  
plot(tree\_FPR, tree\_TPR, type = "l", col= "green",  
 lwd=2,xlab = "False Positive rate",ylab = "True Positive rate", main = "ROC Curves")  
lines(logistics\_FPR, logistics\_TPR, type = "l", col= "red", lwd=2)  
legend("bottomright", c("Optimal Tree", "Logistics Regression"), fill= c("green","red"))



**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**

From the above graph, can observe that the area under the curve is more for logistic regression model than optimal tree model. We can come to a conclusion that use of logistic regression model is the best choice here. Precision-recall curve could be a better option here since the data has unbalanced classes.