**R Programming Lab Mini Project**

**IMDB Movie Rating Prediction**

**Introduction:**

**How can we tell the greatness of a movie before it is released in cinema?This question arises as there is no universal way to claim the goodness of movies. Many people rely on critics to gauge the quality of a film, while others use their instincts. But it takes the time to obtain a reasonable amount of critics review after a movie is released. And human instinct sometimes is unreliable.Given that thousands of movies were produced each year, is there a better way for us to tell the greatness of movie without relying on critics or our own instincts?**

**To answer this question, we tried to predict using KNN, CART, Random Forest.**

It is so hard to find the rating of movie before it is released in the theatres. We are hence predicting Ratings based on different factors such as Actor,Director Facebook likes, Critic Reviews, Budget and Gross money spent on movie, IMDB Score.

**Dataset used:**

Dataset Origin: <https://www.kaggle.com/carolzhangdc/imdb-5000-movie-dataset>

As per the analysis report made on the data set we found that there were 28 columns and 5043 rows.It has many variables like - movie\_title" "color" "num\_critic\_for\_reviews" "movie\_facebook\_likes" "duration" "director\_name" "director\_facebook\_likes" "actor\_3\_name" "actor\_3\_facebook\_likes" "actor\_2\_name" "actor\_2\_facebook\_likes" "actor\_1\_name" "actor\_1\_facebook\_likes" "gross" "genres" "num\_voted\_users" "cast\_total\_facebook\_likes" "facenumber\_in\_poster" "plot\_keywords" "movie\_imdb\_link" "num\_user\_for\_reviews" "language" "country" "content\_rating" "budget" "title\_year" "imdb\_score" "aspect\_ratio".

**Step 1: Importing the dataset**

**Code:**

rm(list=ls())

library(ggplot2)

library(GGally)

library(dplyr)

library(tree)

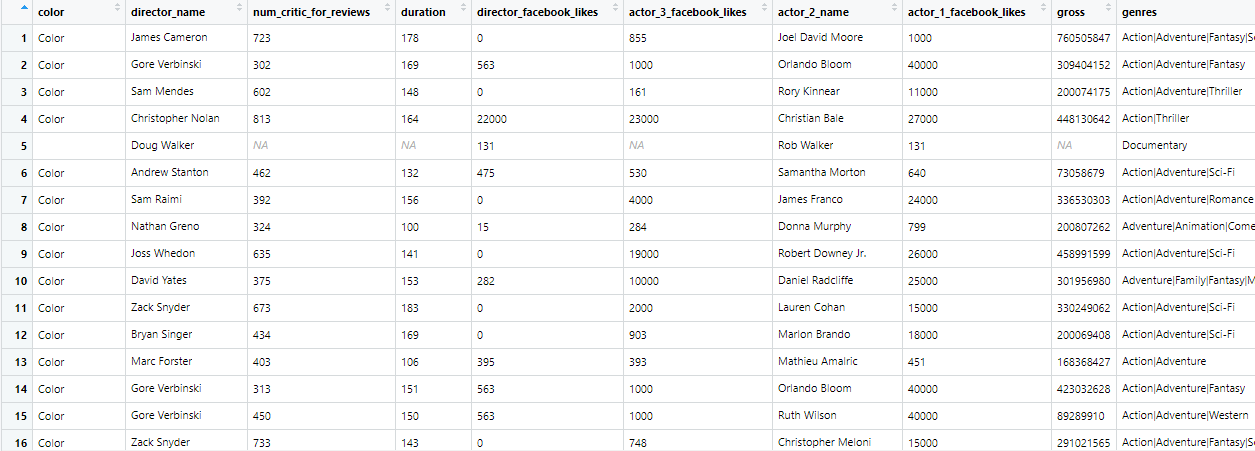
library(rpart)

library(rpart.plot)

#Our Dataset

movies <- read.csv(file="C:/Users/Shweta/Desktop/examples/movie\_metadata.csv",header = T,stringsAsFactors = F)

View(movies)



**Step 2: Cleaning the data**

From the chunk of data, only the required data was added into the data frame.



* Made a sub-set of required columns.
* Deleted the rows – With NA.
* Modified the columns:

For example: ratings 1-7 as – Low , ratings 7-10 as – High.

* Deleted the rows with irrelevant values like Actor Name, Budget.
* Converted the given time into years.
* Normalized whole dataset of required columns before applying KNN, CART and Random Forest.
* Shuffled the datasets for better performance of model.
* Converted all possible values to numeric and categorical forms for comparison.

**Code:**

**#We look for missing data, if there are, then in what proportion?**

**sum(is.na(movies))**

**colSums(is.na(movies))**

**mean(is.na(movies))**

**columns <- c()**

**for(i in 1:dim(movies)[2])**

**{**

**if(is.numeric(movies[,i])|| is.integer(movies[,i]))**

**{**

**columns[i]=T**

**}**

**else**

**{**

**columns[i]=F**

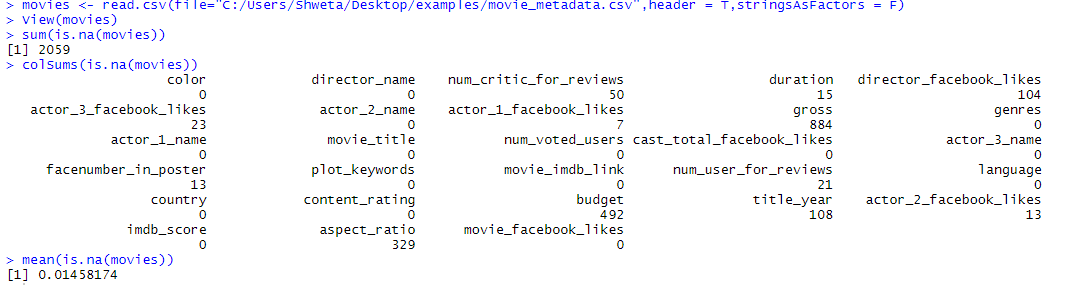
**}**

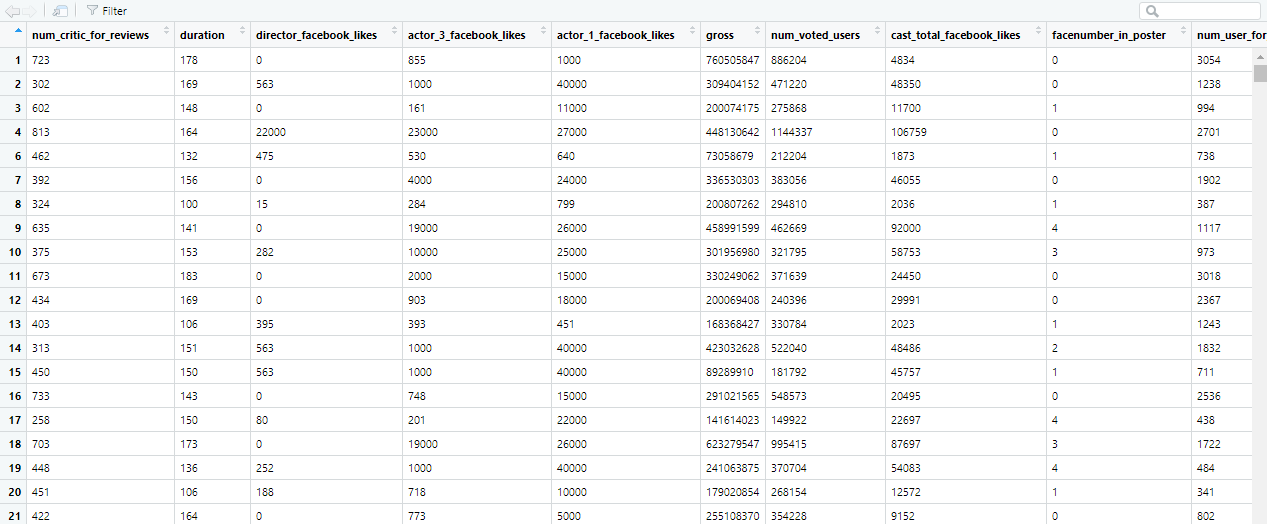
**}**

**movie\_data <- na.omit(movies[,columns])**

**View(movie\_data)**

**Output:**





**Step 3: Data Normalization and Visualization**

After visiting different movie rating web-sites like IMDB, Rotten Tomatoes we came across different variables like “number of users voted”, ”Occasions of movie release”, “Critic review’s” etc.But it is difficult to judge the exact formula as these sites don’t reveal them. We have considered the deciding variables for our predicting model from the common factors from these different websites and we also have created a co-relation matrix between response and independent variables.

**Code:**

require(corrplot)

# Correlation between numeric values

correlation\_graph <- cor(movie\_data[sapply(movie\_data, is.numeric)])

corrplot(correlation\_graph, method="shade")

#How are the variables correlated to the response variable?

correlation <- c()

for(i in 1:dim(movie\_data)[2])

{

correlation[i] <- cor(movie\_data[,i],movie\_data[,'imdb\_score'])

}

correlation

#**Scatter plot to Visualize correlation**

ggplot(movie\_data, aes(x=num\_voted\_users, y=imdb\_score)) + geom\_point(colour="grey60") +

stat\_smooth(method=lm, se=FALSE, colour="black")+ggtitle(paste('R:',correlation[7]))

ggplot(movie\_data, aes(x=duration, y=imdb\_score)) + geom\_point(colour="grey60") +

stat\_smooth(method=lm, se=FALSE, colour="black")+ggtitle(paste('R:',correlation[2]))

ggplot(movie\_data, aes(x=gross, y=imdb\_score)) + geom\_point(colour="grey60") +

stat\_smooth(method=lm, se=FALSE, colour="black")+ggtitle(paste('R:',correlation[6]))

ggplot(movie\_data, aes(x=num\_critic\_for\_reviews, y=imdb\_score)) + geom\_point(colour="grey60") +

stat\_smooth(method=lm, se=FALSE, colour="black")+ggtitle(paste('R:',correlation[1]))

ggplot(movie\_data, aes(x=num\_user\_for\_reviews, y=imdb\_score)) + geom\_point(colour="grey60") +

stat\_smooth(method=lm, se=FALSE, colour="black")+ggtitle(paste('R:',correlation[10]))

#As per the correlation , selecting the respective importance variables

summary(knn\_movie\_data)

data <- knn\_movie\_data[,c(1,2,3,6,7,8,10,12,16)]

View(data)

#Normalizing the data

normalize <- function(x) {

+ return((x-min(x))/ (max(x) - min(x)))

}

data <- as.data.frame(lapply(data[,c(1,2,3,4,5,6,7,8,9)], normalize))

summary(data)

View(data)

#binding the columns

data\_1 <- cbind(data,knn\_movie\_data$imdb\_score)

colnames(data\_1)[10] <- "imdb\_score"

View(data\_1)

#data\_new <- data\_1

set.seed(9850)

g <- runif(nrow(data\_1))

data\_new <- data\_1[order(g),]

str(data\_new)

View(data\_new)

#Actual Data

idx\_test\_1 <- data\_new[seq(1, nrow(data\_new), by = 5),]

idx\_test\_1

nrow(idx\_test\_1)

idx\_train\_1 <- data\_new[-seq(1, nrow(data\_new), by = 5),]

idx\_train\_1

nrow(idx\_train\_1)

test\_1 <- idx\_test\_1[,-10]

training\_1 <- idx\_train\_1[,-10]

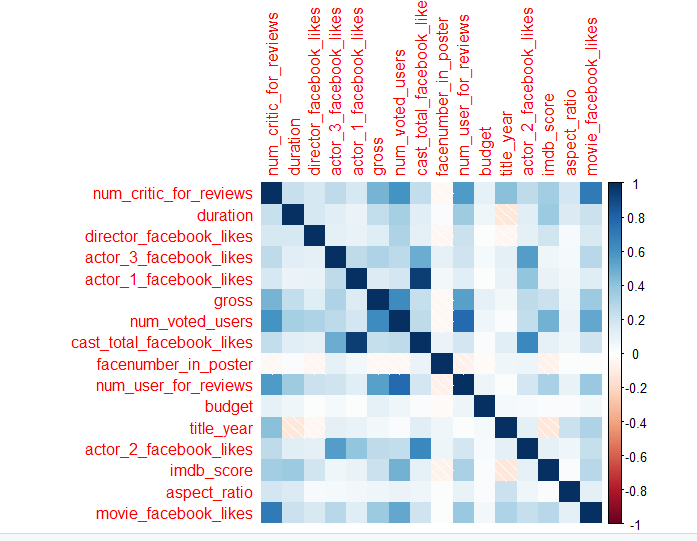
#Target Data

training\_target <- idx\_train\_1[,10]

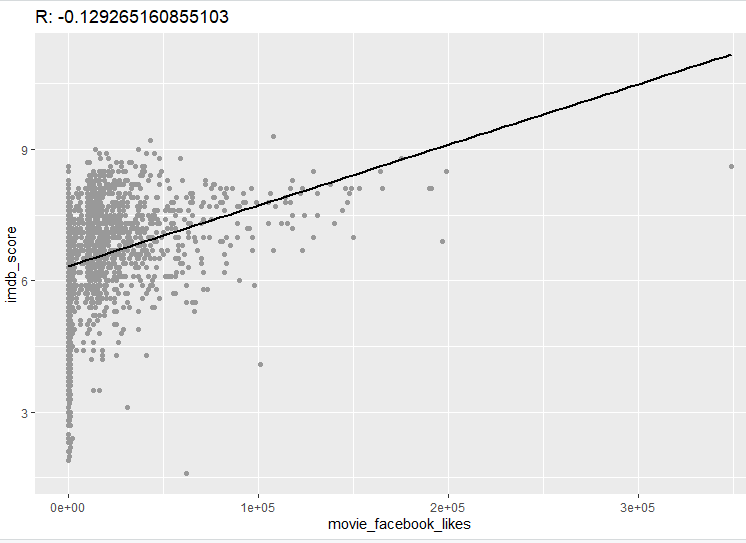
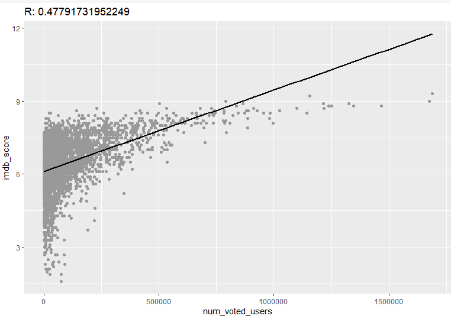
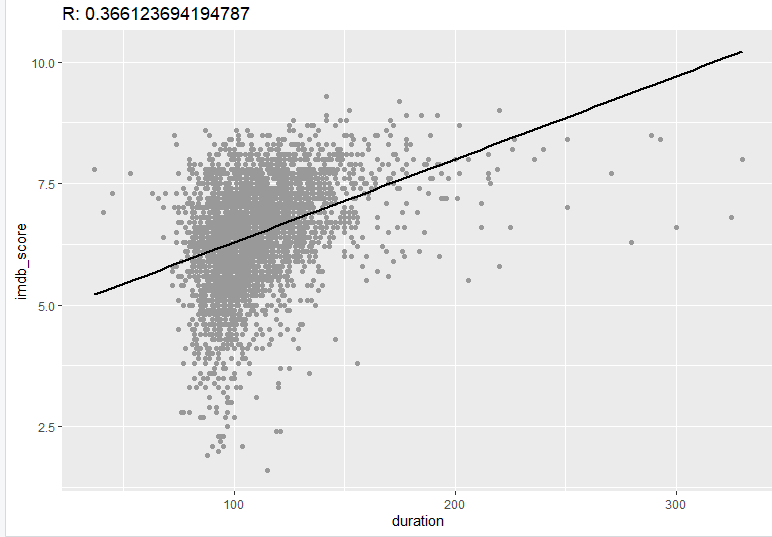
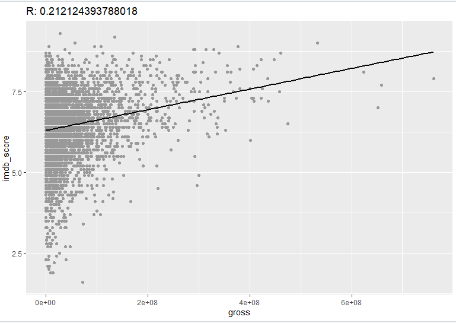
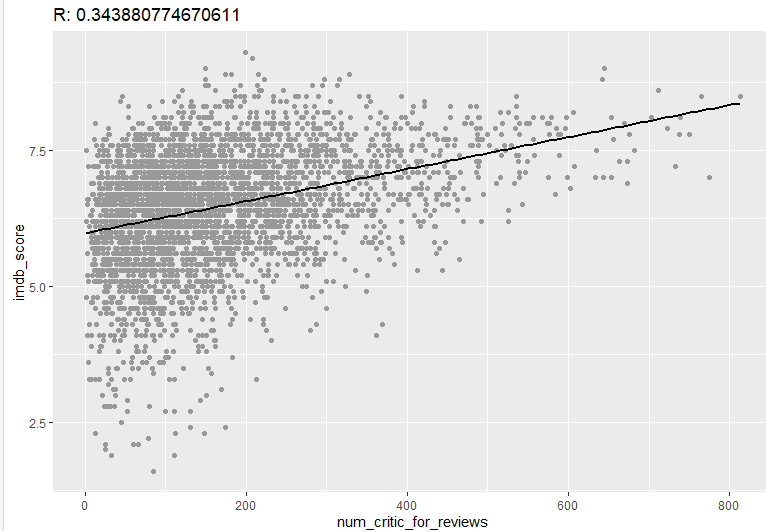
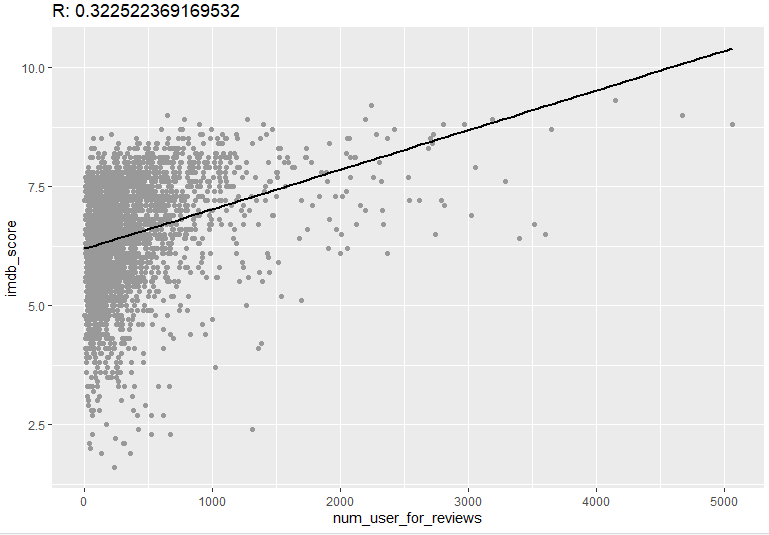
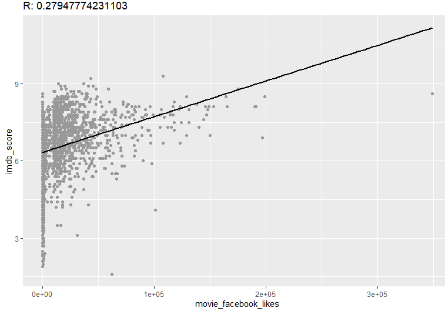
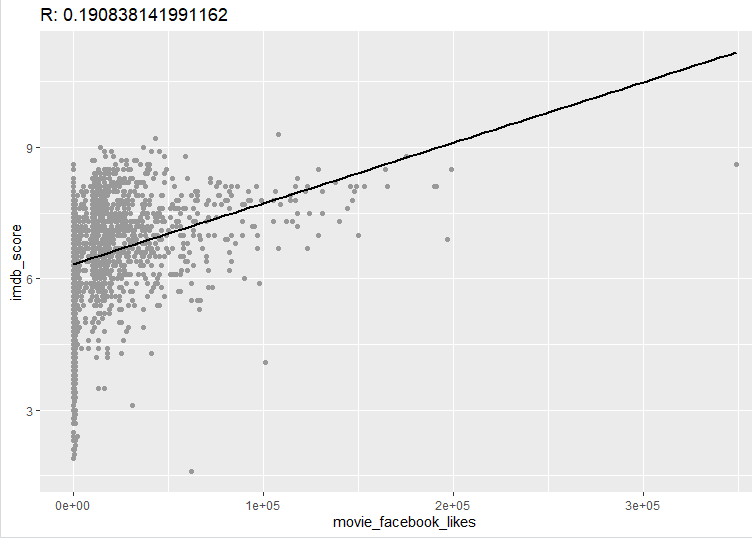
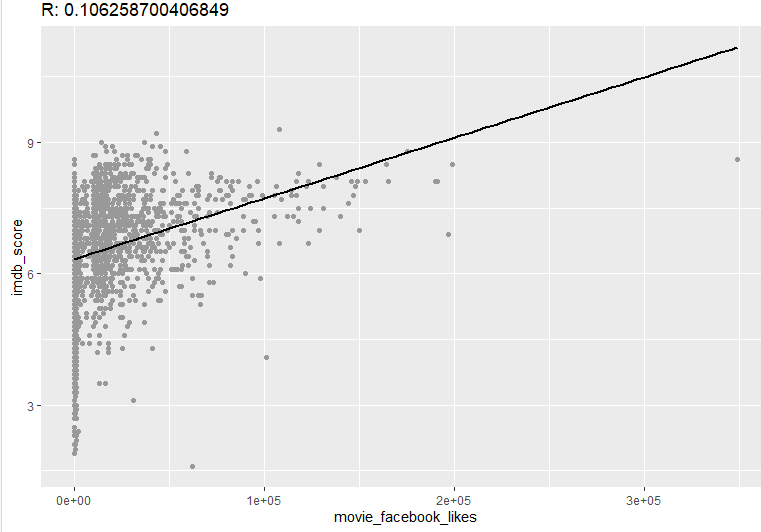
test\_target <- idx\_test\_1[,10]

**Output:**

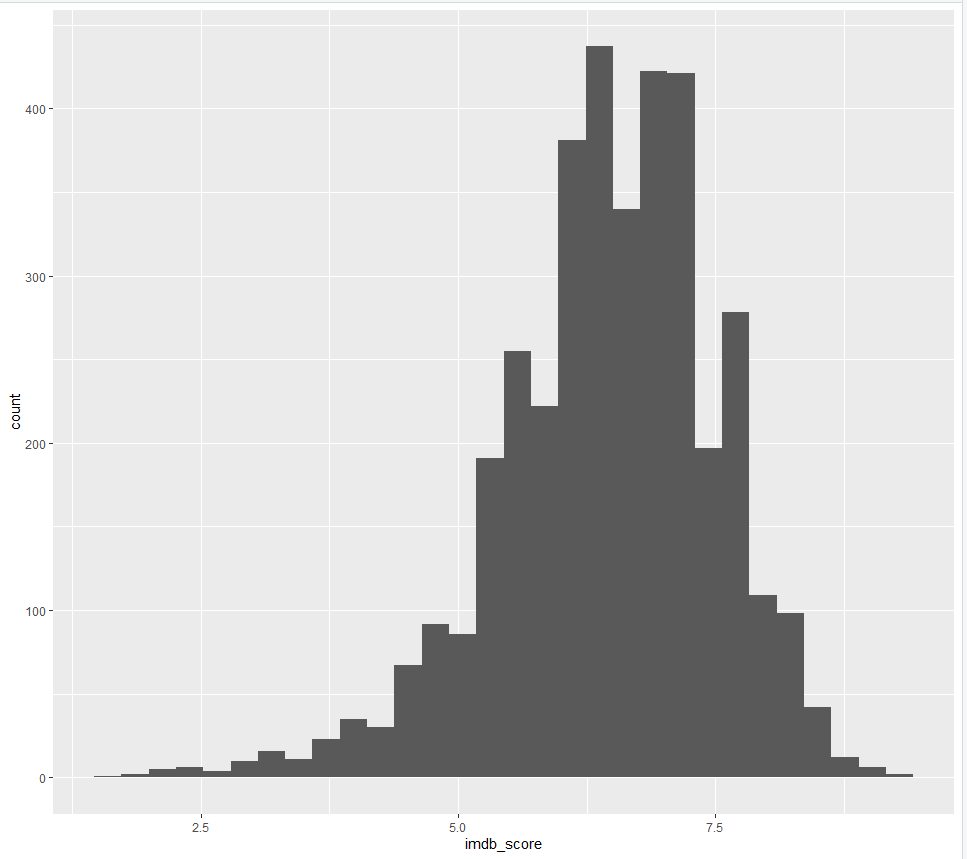




**Correlation Scatter Graphs:**



**Scatter Plot:**



**Step 4: Model Selection and Prediction**

**KNN(K-Nearest Neighbors)**

The Algorithm:  
k-Nearest Neighbor is an example of instance-based learning, in which the training data set is stored, so that a classification for a new unclassified record may be found simply by comparing it to the most similar records in the training set.

1)First we tried KNN on all the numeric factors we have in our dataset. We then analysed the accuracy of our model, and took out the unwanted variable which were affecting the prediction accuracy.

2)We divided our dataset in 20% - Test and 80% - Train data and we applied the KNN algorithm over the training dataset and tested the model on our test Dataset.

3)Afterwords using the confusion matrix we compared the actual values with predicted values.

**Code:**

**#Applying KNN using all the variables**

**#Actual Data**

**idx\_test\_1\_temp <- data\_new\_temp[seq(1, nrow(data\_new\_temp), by = 5),]**

**idx\_test\_1\_temp**

**nrow(idx\_test\_1\_temp)**

**idx\_train\_1\_temp <- data\_new\_temp[-seq(1, nrow(data\_new\_temp), by = 5),]**

**idx\_train\_1\_temp**

**nrow(idx\_train\_1\_temp)**

**test\_1\_temp <- idx\_test\_1\_temp[,-16]**

**training\_1\_temp <- idx\_train\_1\_temp[,-16]**

**#Target Data**

**training\_target\_temp <- idx\_train\_1\_temp[,16]**

**test\_target\_temp <- idx\_test\_1\_temp[,16]**

**library(class)**

**require(class)**

**#Use knn with k=15 and classify the test dataset**

**#Measuring the performance with k=5**

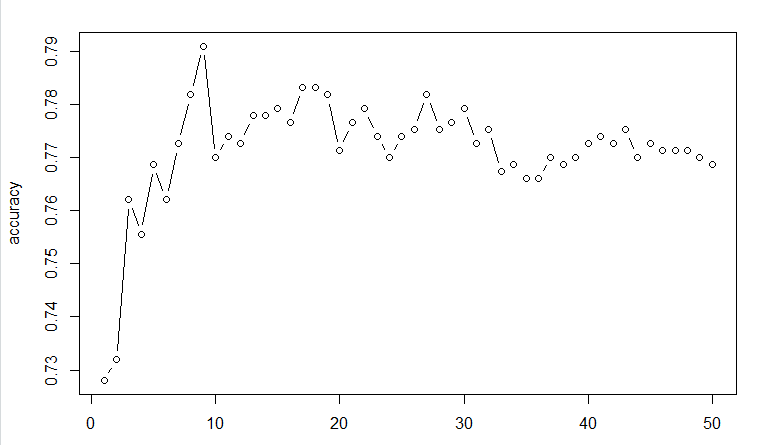
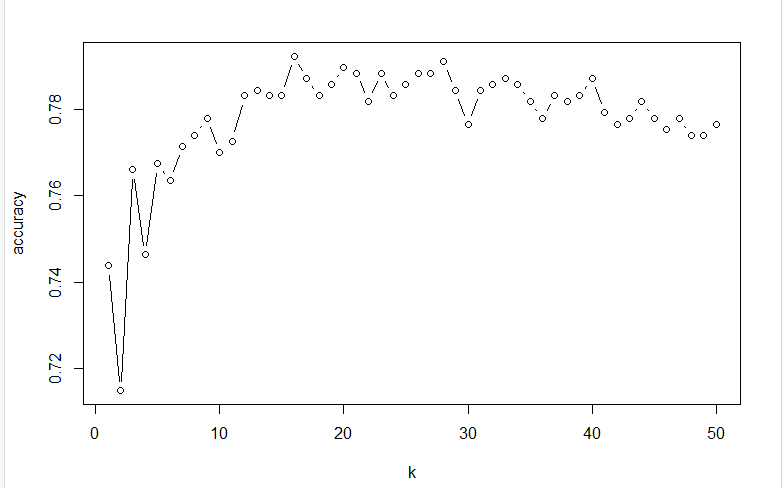
**predict\_temp <- knn(training\_1\_temp, test\_1\_temp, training\_target\_temp, k=5)**

**tab\_temp <- table(Actual=test\_target\_temp,Predict=predict\_temp)**

**#Accuracy**

**mean(predict\_temp == test\_target\_temp)**

**Output:**

We started with different values of K for prediction starting with K=5 and increasing thereafter. With K=5 we got an **accuracy of 76.47%** and misclassification rate of 23.52%

To increase the prediction accuracy rate we tried different values and found highest accuracy for K=16 as 78.71% and misclassification rate of 21.28%, which is shown in above graph.

**Classification and Regression Trees(CART):**

**CART :**

Decision trees produced by CART are strictly binary, containing exactly two branches for each decision node. The CART algorithm grows the tree by conducting for each decision node, an exhaustive search of all available variables and all possible splitting values, selecting the optimal split according to the following criteria.



For Implementing CART in RStudio we are using **RPART** function.

We divided our dataset in 20% - Test and 80% - Train data and we applied the CART algorithm over the training dataset and tested the model on our test Dataset.

**Code:**

data\_tree <- knn\_movie\_data[,c(1,2,3,6,7,8,10,12,16)]

data\_1\_tree <- cbind(data\_tree,knn\_movie\_data$imdb\_score)

colnames(data\_1\_tree)[10] <- "imdb\_score"

View(data\_1\_tree)

#data\_new <- data\_1

set.seed(9850)

g <- runif(nrow(data\_1\_tree))

data\_new\_tree <- data\_1\_tree[order(g),]

str(data\_new\_tree)

View(data\_new\_tree)

head(data\_new\_tree)

nrow(data\_new\_tree)

data\_tree <- knn\_movie\_data[,c(1,2,3,6,7,8,10,12,16)]

data\_1\_tree <- cbind(data\_tree,knn\_movie\_data$imdb\_score

colnames(data\_1\_tree)[10] <- "imdb\_score"

View(data\_1\_tree)

#data\_new <- data\_1

set.seed(9850)

g <- runif(nrow(data\_1\_tree))

data\_new\_tree <- data\_1\_tree[order(g),]

str(data\_new\_tree)

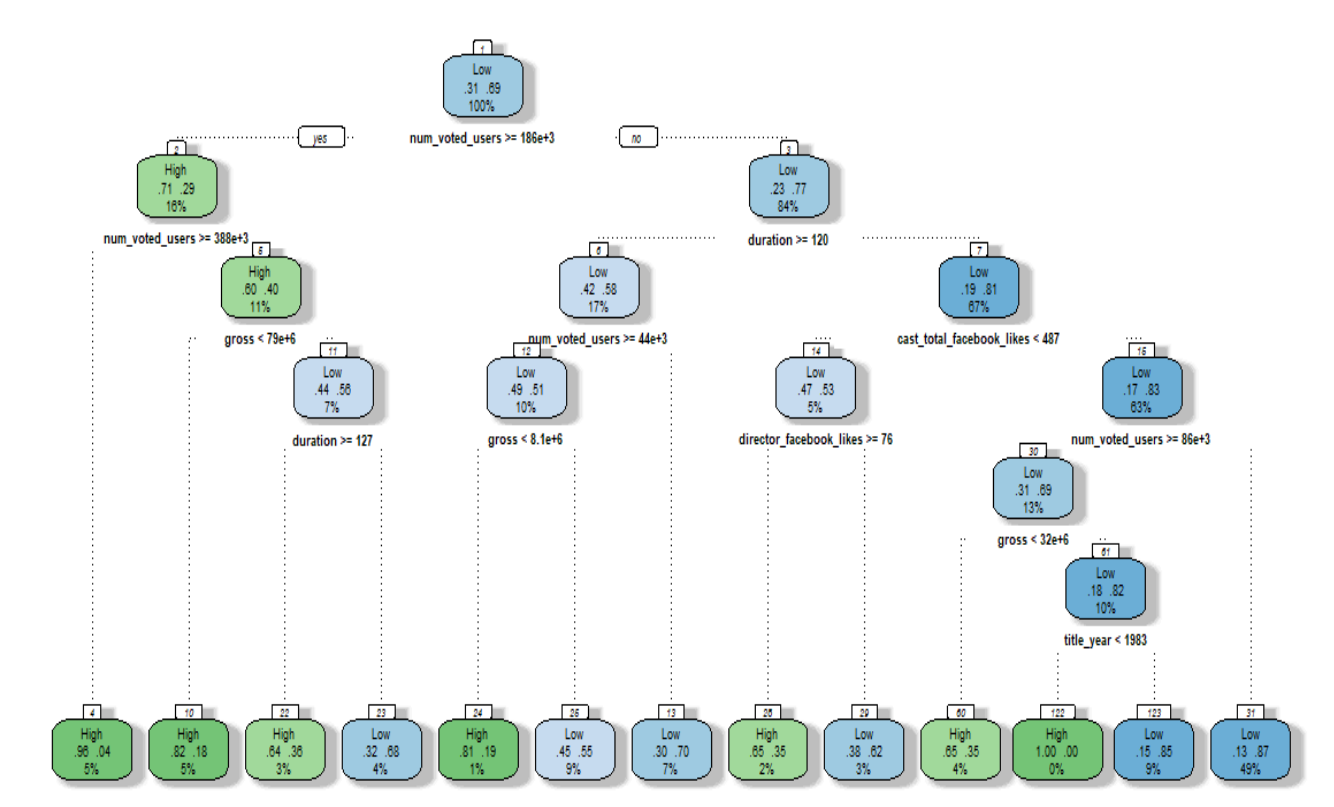
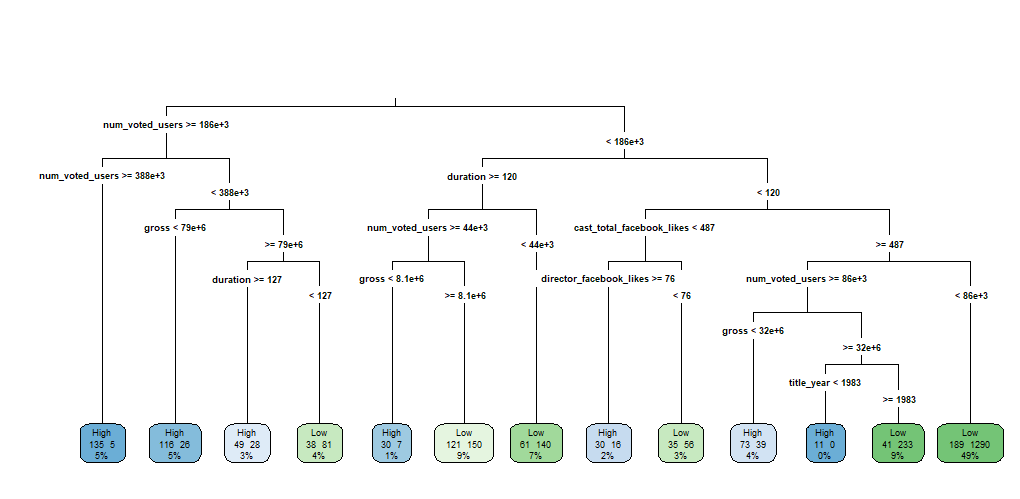
View(data\_new\_tree)

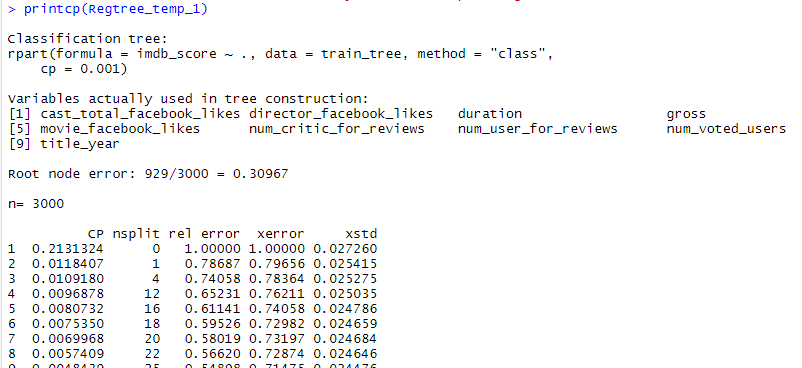
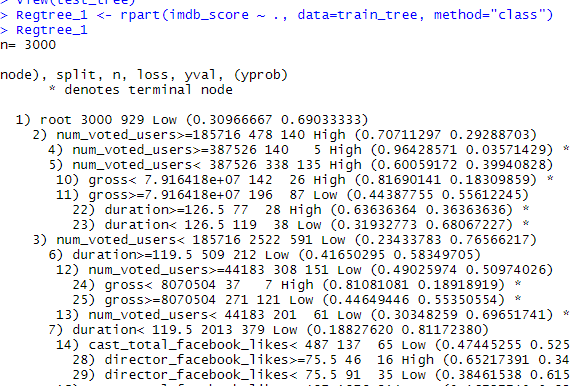
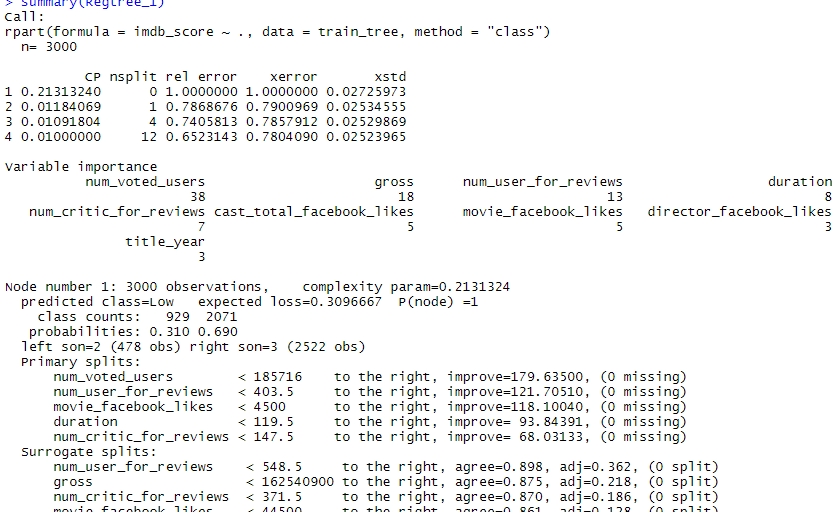
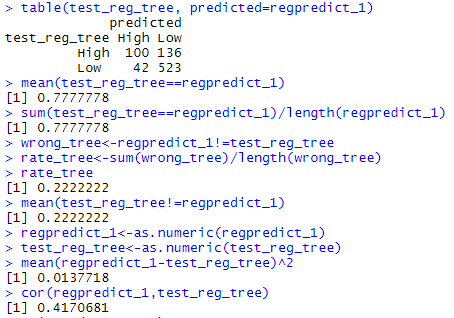
head(data\_new\_tree)

nrow(data\_new\_tree)

**Output:**

Below is the decision tree after applying the model on our training dataset.



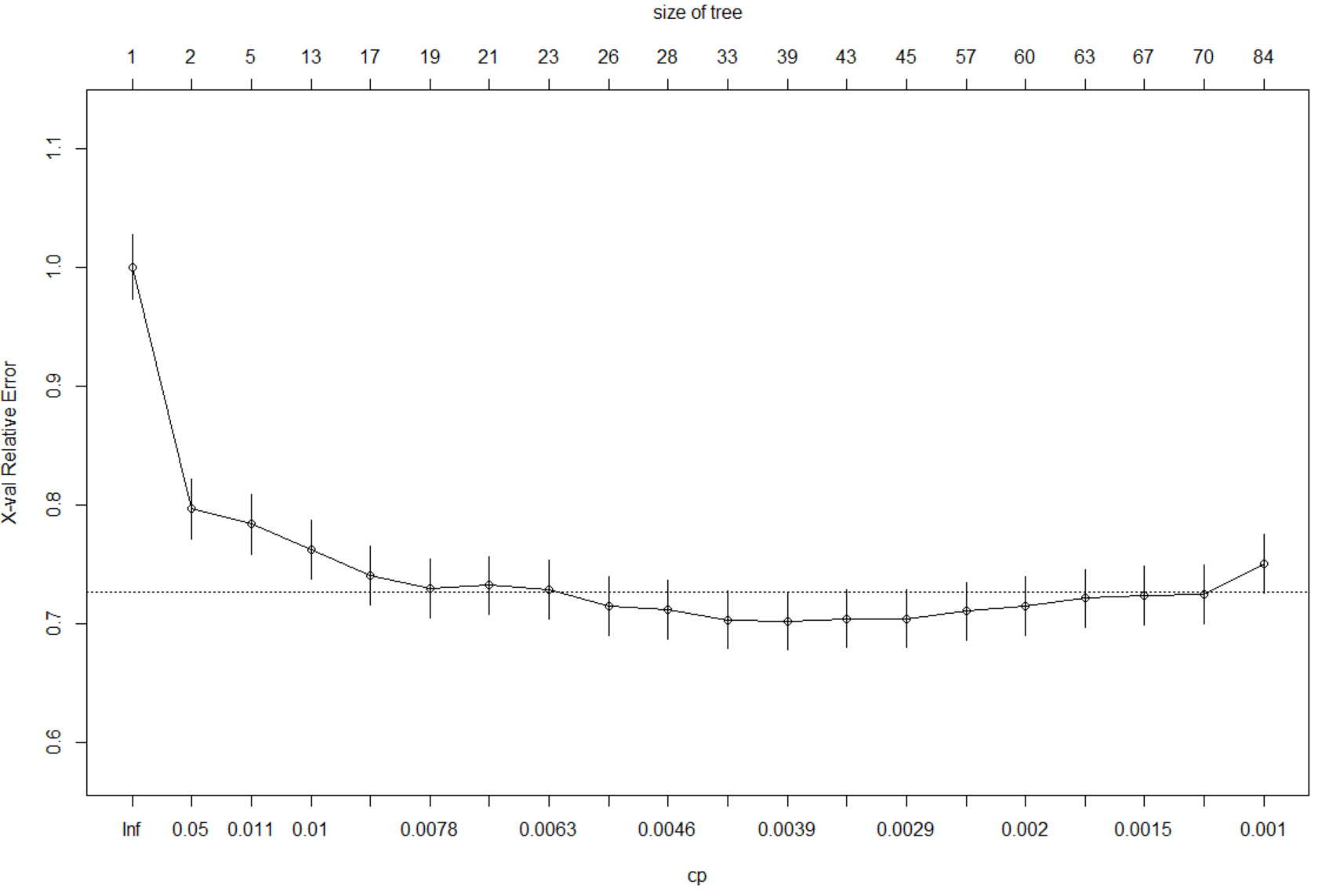


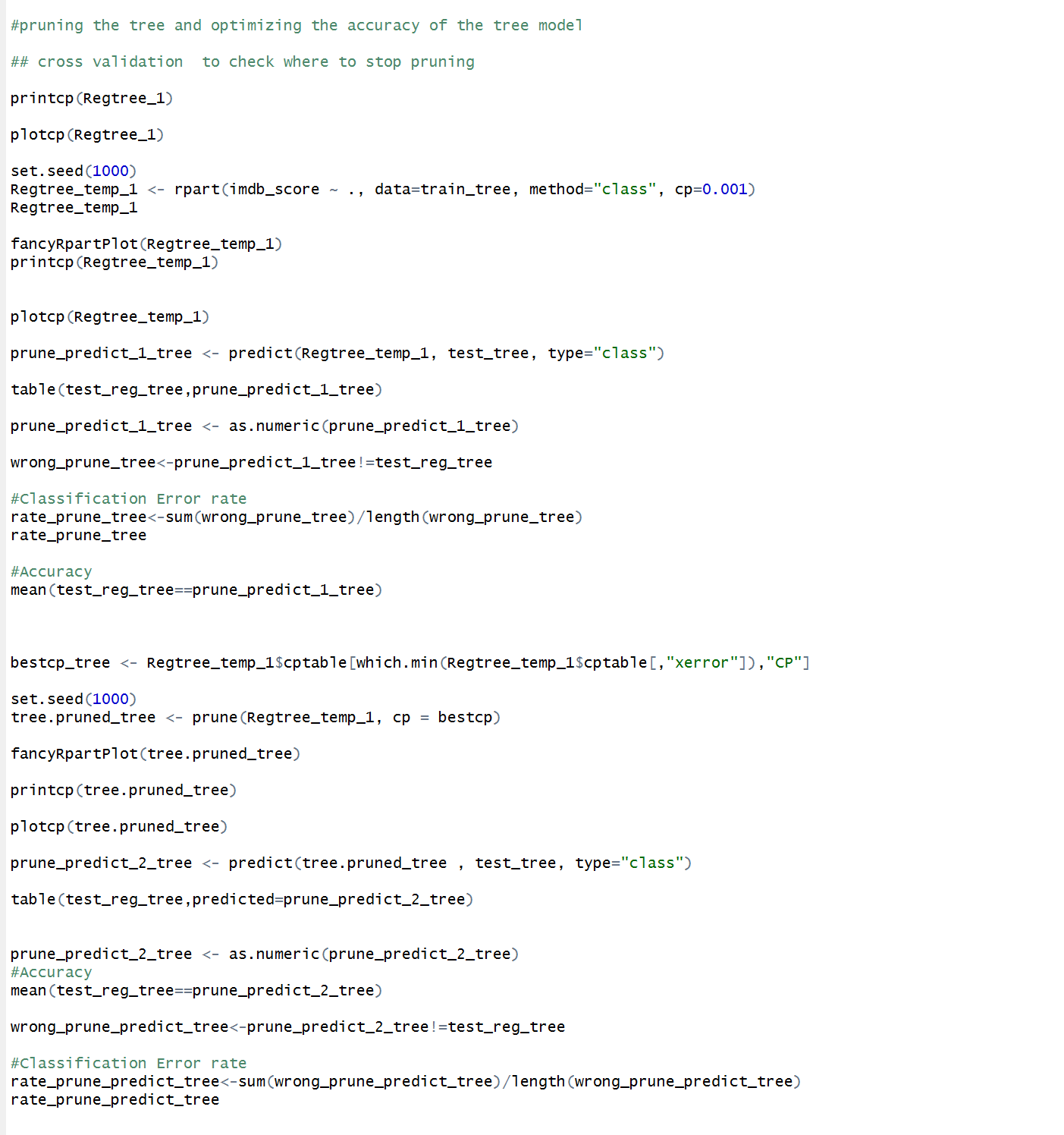
After testing the model on our test dataset, we compared it with target variable and got **accuracy of 77.77%** and error rate of 22.22%.

Co-relation between the predicted values and actual values is around 42%.

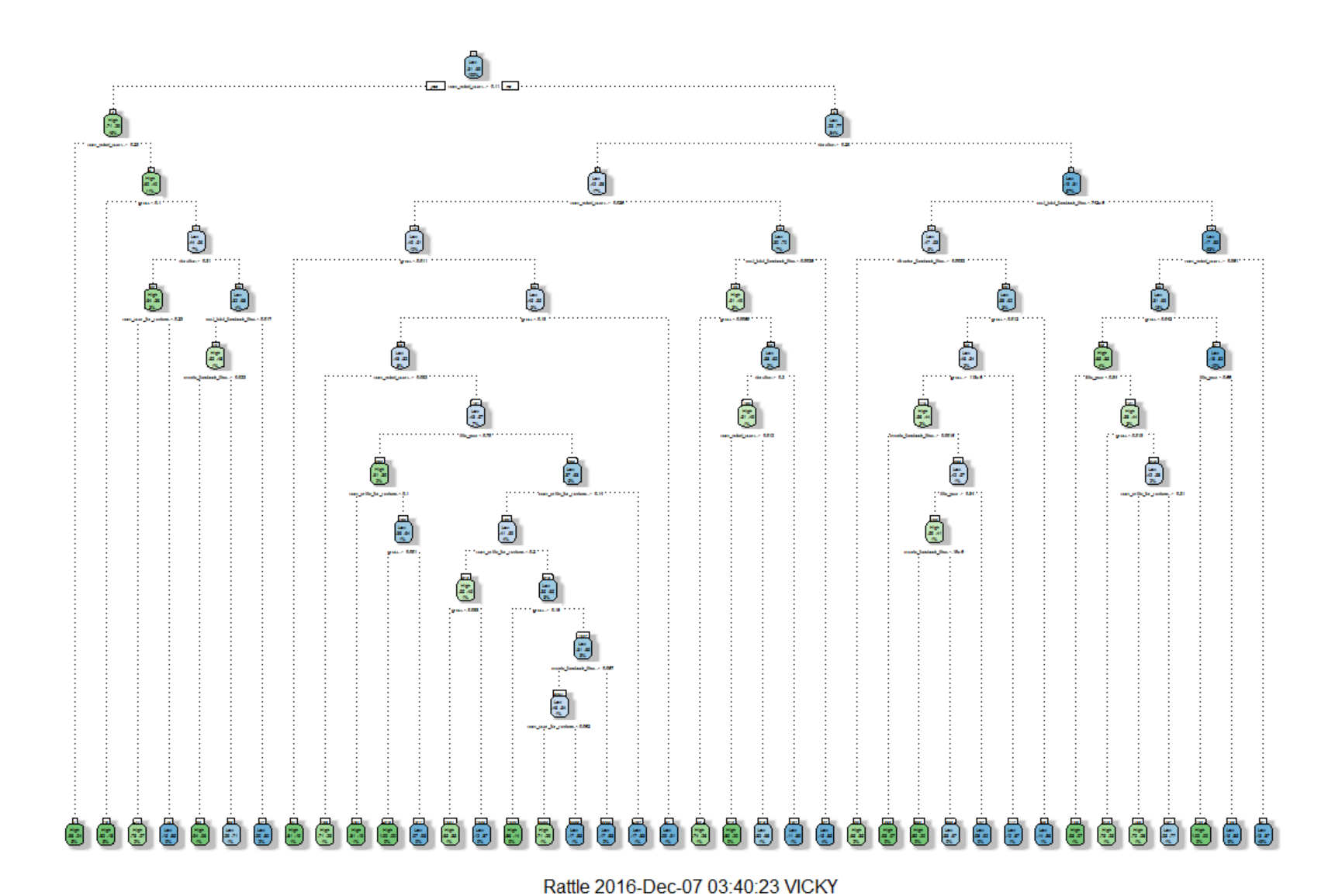
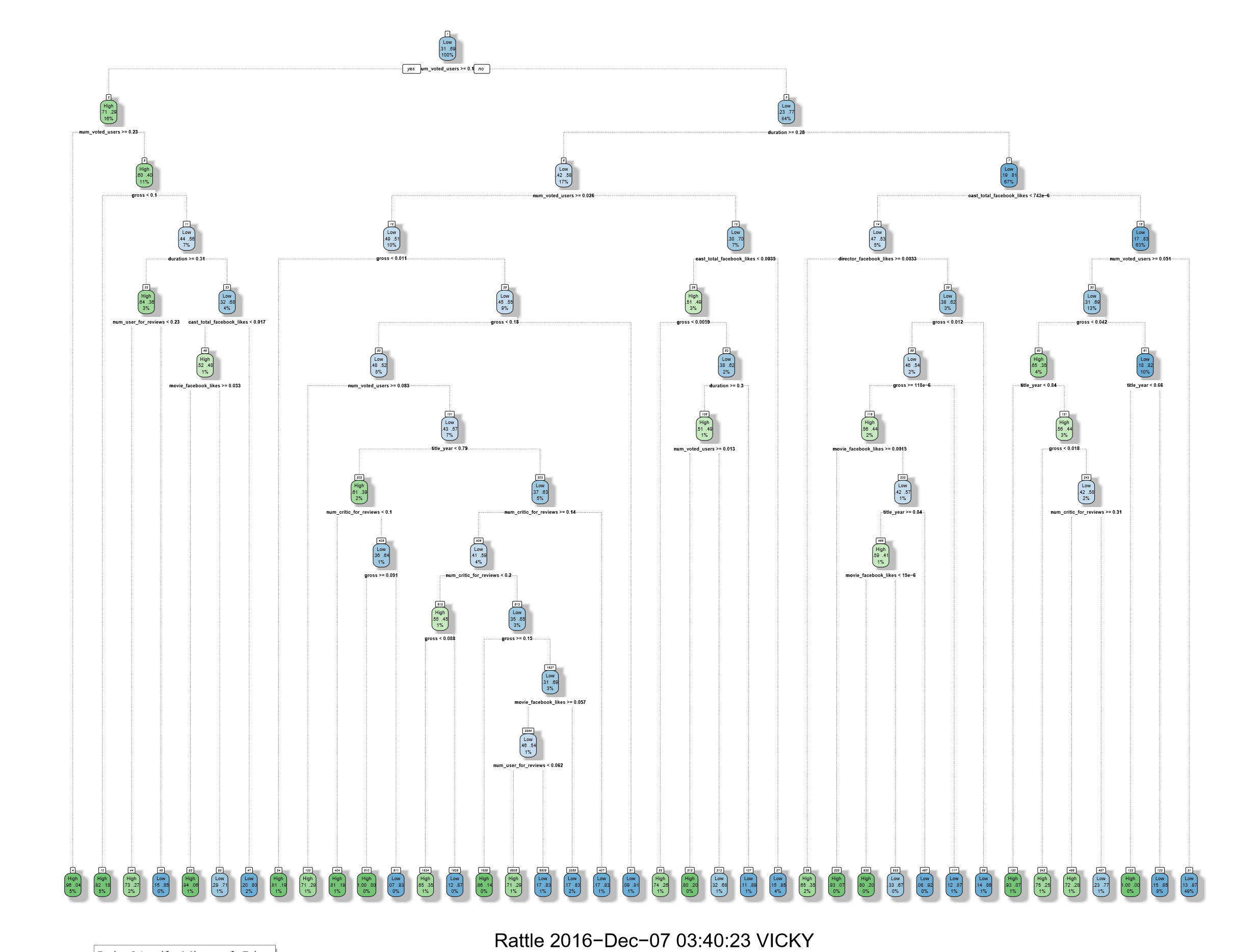
Pruned the tree to increase the optimality.

Applied rpart with least cp, resulted in accuracy reduction and increase in error rate.





**Optimal Decision Tree:**

**Random Forest:**

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.For Implementing Random Forest in RStudio we are using randomForest function.

We divided our dataset in 20% - Test and 80% - Train data and we applied the Random Forest algorithm over the training dataset and tested the model on our test Dataset.

**Code:**

set.seed(500)

install.packages("randomForest")

library(randomForest)

set.seed(500)

rf\_first <- randomForest(imdb\_score~.,data=train\_tree)

plot(rf\_first)

pred\_rf\_first <- predict(rf\_first, test\_tree)

table(test\_reg\_tree,predicted=pred\_rf\_first)

pred\_rf\_first<-as.numeric(pred\_rf\_first

#Mean Square error

mean(pred\_rf\_first- test\_reg\_tree)^2

#Accuracy

mean(pred\_rf\_first==test\_reg\_tree)

#With parameter ntree=500

rf <- randomForest(imdb\_score~.,data=train\_tree ,ntree=500,mtry=floor(dim(data\_new)[2]/3))

plot(rf)

pred\_rf <- predict(rf, test\_tree)

table(test\_reg\_tree,predicted=pred\_rf)

pred\_rf<-as.numeric(pred\_rf)

#Mean Square error

mean(pred\_rf- test\_reg\_tree)^2

#Accuracy

mean(pred\_rf==test\_reg\_tree)

wrong\_rf\_algo<-pred\_rf!=test\_reg\_tree

#Classification Error rate

rate\_rf\_algo<-sum(wrong\_rf\_algo)/length(wrong\_rf\_algo)

rate\_rf\_algo

array\_ntree<- c(100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500)

mse <- c()

j<-1

for(i in array\_ntree)

{ set.seed(500)

rf <- randomForest(imdb\_score~.,data=train\_tree,ntree=i)

pred\_rf <- predict(rf,test\_tree)

pred\_rf <- as.numeric(pred\_rf)

mse[j]<-mean((pred\_rf-test\_reg\_tree)^2)

j=j+1

}

#Accuracy

mean(pred\_rf\_tune==test\_reg\_tree)

wrong\_rf<-pred\_rf\_tune!=test\_reg\_tree

rate\_rf<-sum(wrong\_rf)/length(wrong\_rf)

rate\_rf

plot(rf\_tune)

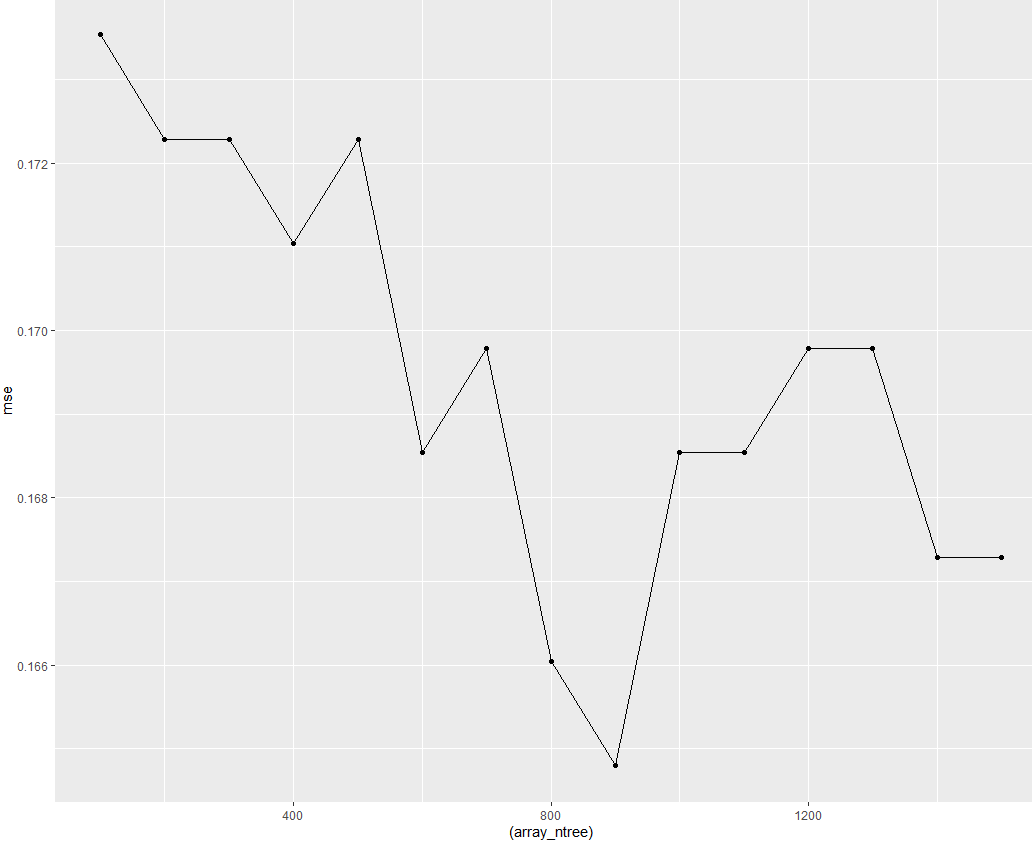
varImpPlot(rf)

**Output:**

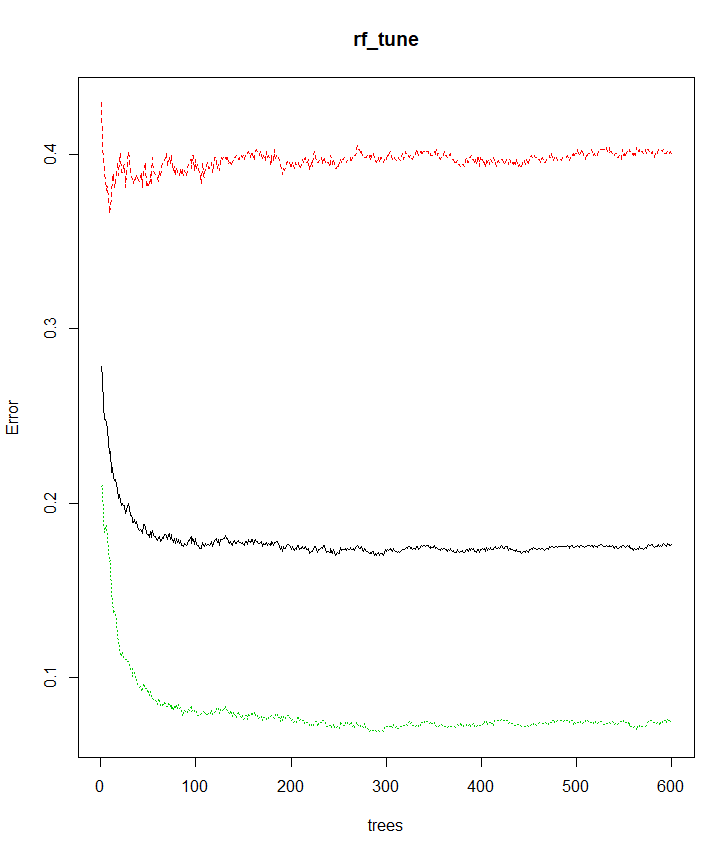
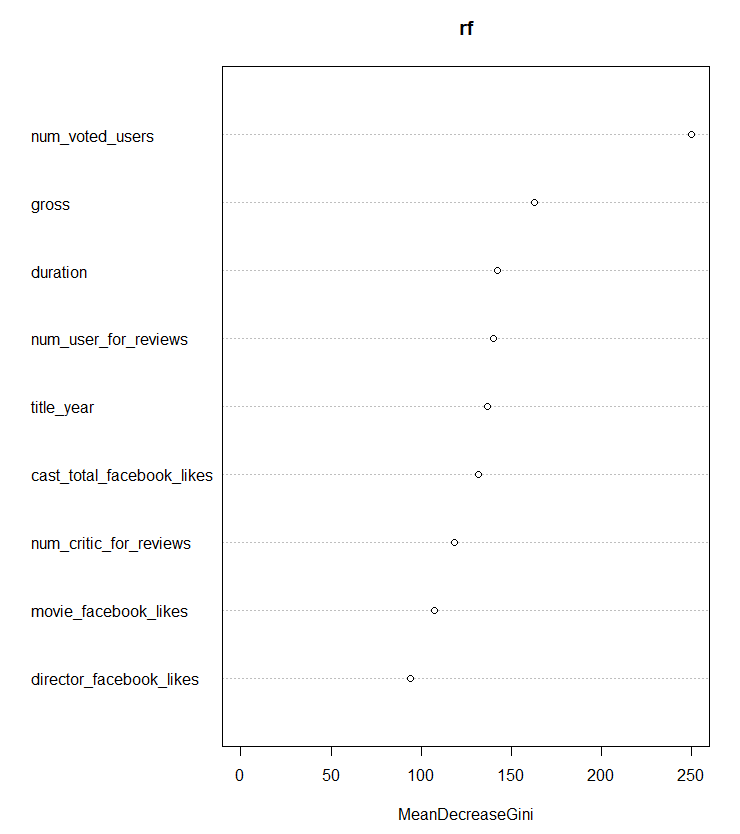
After testing the model on our test dataset, we compared it with target variable and got accuracy of 82.52% and error rate of 17.47%.

Co-relation between the predicted values and actual values is around 56.08%

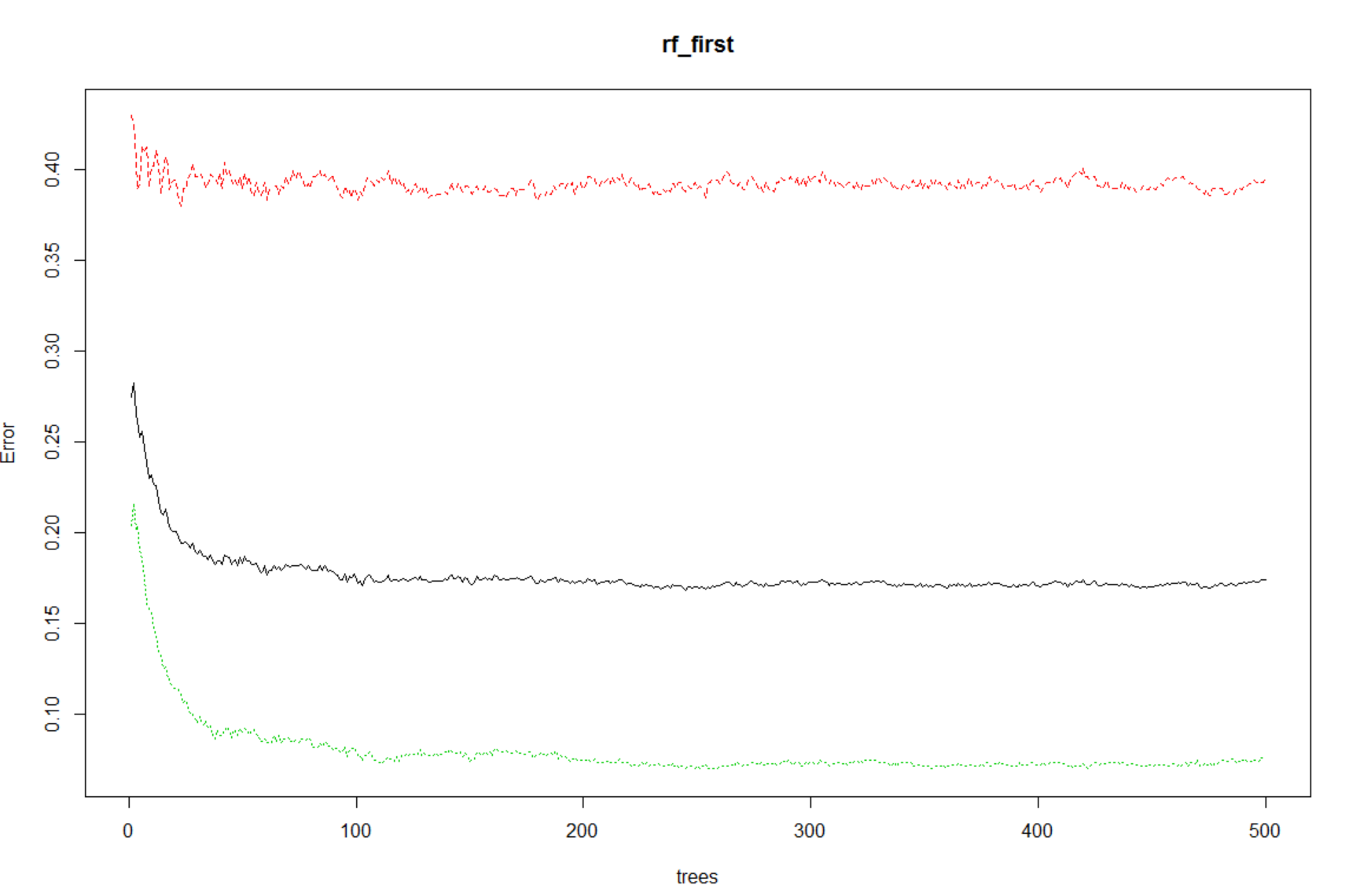
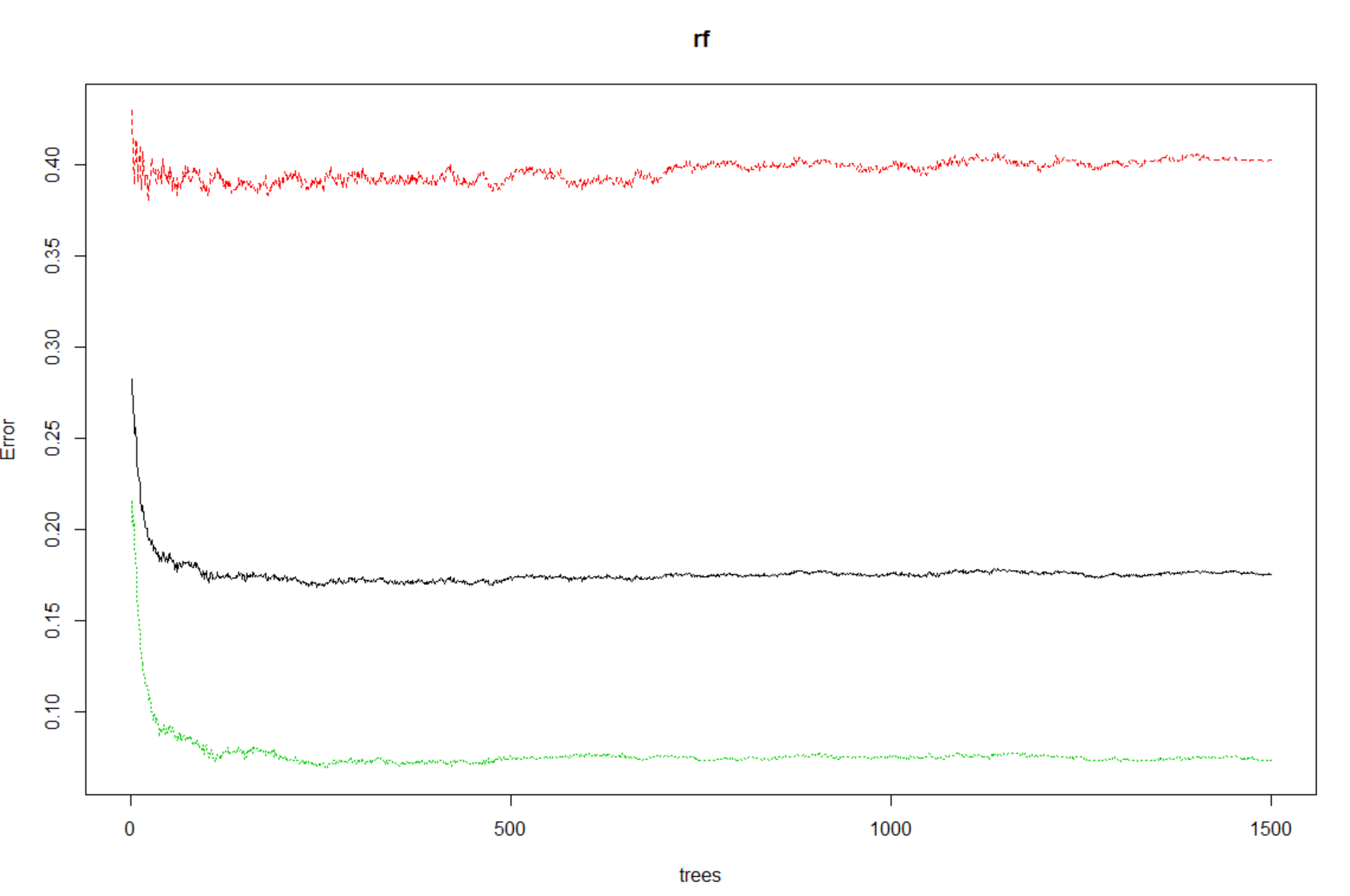
Tuning the n-tree parameter.



After Changing the entry parameter to its optimal value in randomForest function it gained an **accuracy of 82.64%** and error rate of 17.35%.

**Random Forest Plot:**

Without the ntree parameter

With ntree parameter = 500

**Conclusion:**

Finally we conclude by deducing that Random Forest can predict the highest accuracy on this dataset on the optimal variables considered by highest co-relation values. We can use this predictive model on any new dataset having the same variables for optimal solution.