20171121_Batch 33_CSE 7302c_CUTe Report

Group-4

Problem Statement:

AMZ is an e-commerce giant. The organization manages its own warehouse at different locations and ships the products to its customers. Often, the products delivered are returned by the customers for several reasons. Can you predict which of the products are likely to be returned? This would enable them to manage their inventory more efficiently.

Data:

1. The total number of records is 34293 with 23 independent attributes and the 24rd column is the target which needs to be predicted.

2. The variables are masked and we get little information from their names.

3. Missing values are represented as NA and in one column missing value is represented by -99

Solution:

We have to predict whether a particular product is returned or not given some factors.

So, this is a **classification problem**.

Steps in model building:

1. Data Preprocessing.
2. Splitting the data into train and test.
3. Building the models.
4. Evaluating the models.
5. Performance metrics.
6. Choosing the best model.

1. Data Preprocessing:

Reading the dataset

```
#data <- read.csv(file = "DataSets/Data_20171121.csv",header = T)
```

Looking at the structure and summary of data

```
#str(data)
#summary(data)
```

Data preprocessing

Removing the id attribute as it is not useful for modle building.

```
#data <- data[,-1]
```

S7 is a constant column so removing it.

```
#str(data)
```

Checking whether the columns are constant or not.

```
#atrr <- c("RF1","RF2","RF3","RF4","RF5","PI")
#atrData <- subset(data,select=atrr)
#c <- apply(atrData,MARGIN = 2,FUN = sum)
#c
```

Removing PI,RF5,RF2,RF5 as they contain almost same value

```
#data <- subset(data,select=-c(S7,RF5,RF2,RF4,PI))
#head(data)
```

Checking for missing values

```
#sum(is.na(data))
```

Handling missing values

Writing the function for replacing -99 with NA's in two col's

```
#Atr <- c("P6","P12")
#for(i in Atr){
        for(j in 1:nrow(data)){
                if(data[j,i]==-99){
                        data[j,i]=NA
                 }
        }
}
```

```
#str(data)
#summary(data)
#sum(is.na(data))
```

Checking for the rows which contain more than 20% of missing values

```
#library(DMwR)
#manyNAs(data,0.2)
```

There are no rows containing 20% of missing values.

Converting categorical attributes using as.factor method

```
#catAtr <- c("RF1","RF3","Target")
#for(i in catAtr){
#	data[,i] <- as.factor(as.character(data[,i]))
#}
#str(data)
```

Imputation of missing values

```
#set.seed(123)
#data <- knnImputation(data,k=10)
#sum(is.na(data))
#str(data)
#summary(data)
```

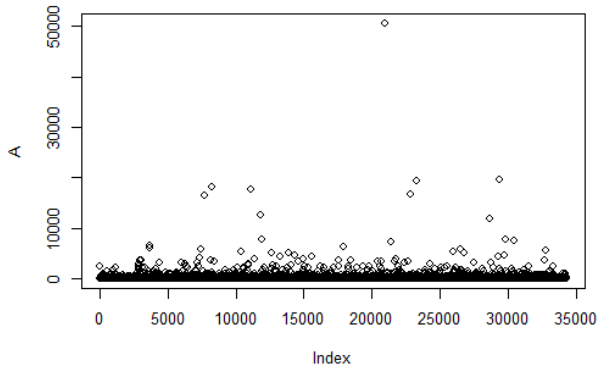
Data insights:

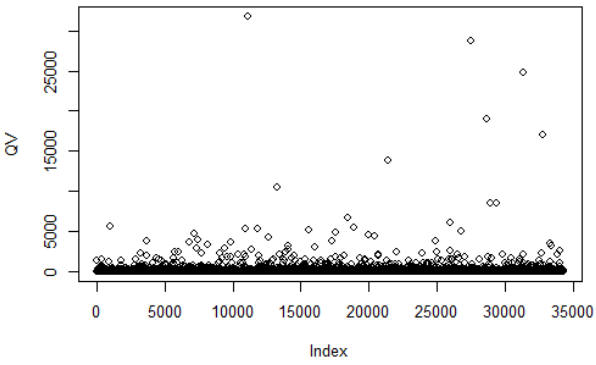
Checking for outliers

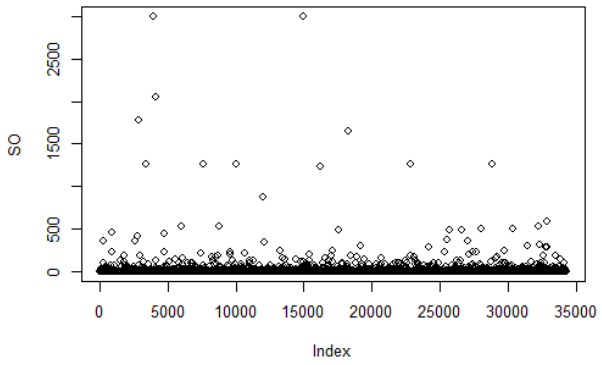
```
#vars <- c("A","QV","SO","F3","F6","F9","S1","S3","S6","S9")
#rem <- setdiff(colnames(data),vars)
```

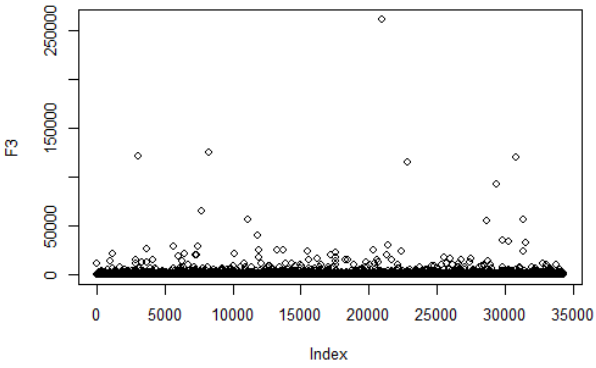
Plotting the data attribute wise

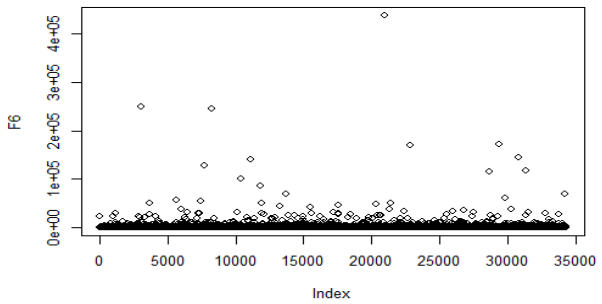
```
#par(mfrow=c(2,2))
#for(i in vars){
#	plot(data[,i],ylab = i)
#}
```

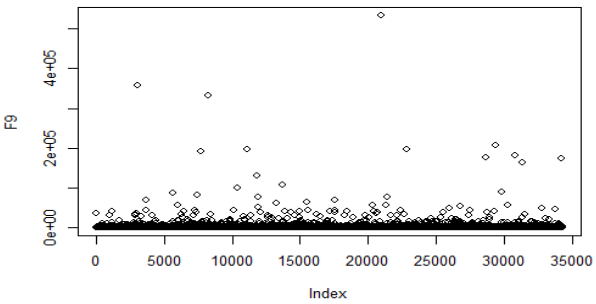

A
0
10000
30000
50000
0
5000
10000
15000
20000
25000
30000
35000
Index

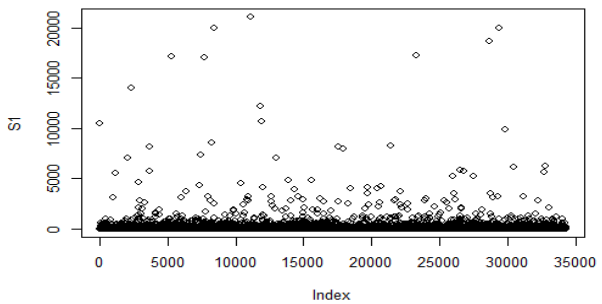

QV
0
5000
15000
25000
0
5000
10000
15000
20000
25000
30000
35000
Index

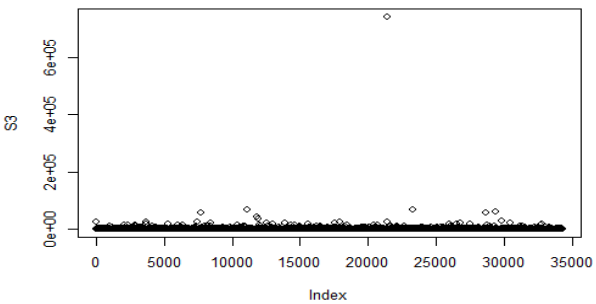

SO
0
500
1500
2500
0
5000
10000
15000
20000
25000
30000
35000
Index

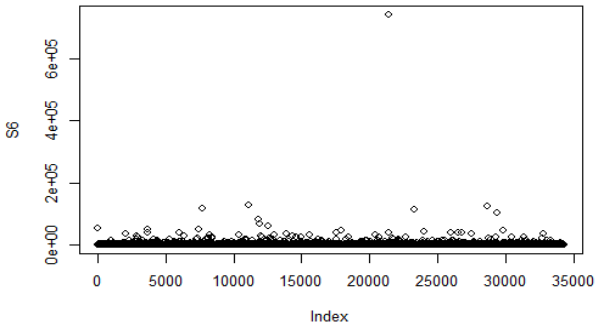

F3
0
50000
150000
250000
0
5000
10000
15000
20000
25000
30000
35000
Index

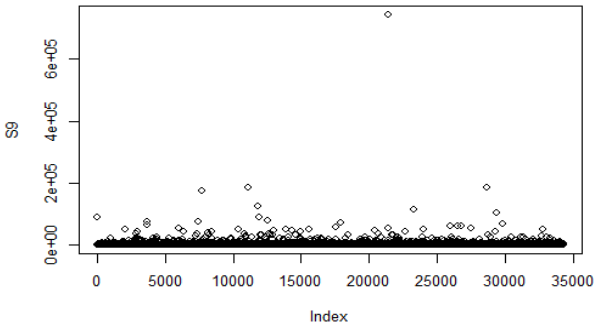

F6
0e+00
1e+05
2e+05
3e+05
4e+05
0
5000
10000
15000
20000
25000
30000
35000
Index


F9
0e+00
2e+05
4e+05
0
5000
10000
15000
20000
25000
30000
35000
Index


S1
0
5000
10000
15000
20000
0
5000
10000
15000
20000
25000
30000
35000
Index


S3
0e+00
2e+05
4e+05
6e+05
0
5000
10000
15000
20000
25000
30000
35000
Index





Removing the influential points

```
#outs = list()

#outs$A <- which(data$A > 10000)

#outs$QV <- which(data$QV > 10000)

#outs$SO <- which(data$SO > 1500)

#outs$F3 <- which(data$F3 > 100000)

#outs$F6 <- which(data$F6 > 100000)

#outs$F9 <- which(data$F9 > 200000)

#outs$S1 <- which(data$S1 > 100000)

#outs$S3 <- which(data$S3 > 200000)

#outs$S6 <- which(data$S6 > 200000)

#outs$S9 <- which(data$S9 > 200000)
```

```
#outs$IV <- which(data$IV > 100000)

#nrow <- unique(unlist(outs))

#data <- data[-nrow,]

#summary(data)
```

Applying data transformation as the data is positively skewed

Feature Engineering

```
#mydata <- subset(data,select=vars)

#summary(mydata)
```

Cubeth root is applied to numerical cols as the max val is in order of 10^3 times of min val

```
#trans <- function(x){

        x^(1/3)

#}

#mydata <- apply(mydata,2,trans)

#summary(mydata)

#data_trans <- data.frame(mydata,data[,rem])

#str(data_trans)
```

2. Splitting the train and test set:

Preparing the train and test set using create data partition function in caret package.

```
#library(caret)
#set.seed(1234)
```

```
#trainRows <- createDataPartition(data_trans$Target,p = 0.7,list = F)
#trainData <- data_trans[trainRows,]
#testData <- data_trans[-trainRows,]
```

3. Model Building:

1. Model-1(Logistic Regression):

Model1 using logistic regression

```
#log_model1 <- glm(Target~.,data = data_trans,family = binomial)
#summary(log_model1)
```

AIC 29852 - for cubeth root transformation(Feature Engineering)

#AIC 32334 - for sqrt transformation(Feature engineering)

Evaluating the error metrics

```
#prob_train <- predict(log_model1,trainData,type = "response")
#prob_test <- predict(log_model1,testData,type = "response")
```

Plotting the roc curve and calculating the auc

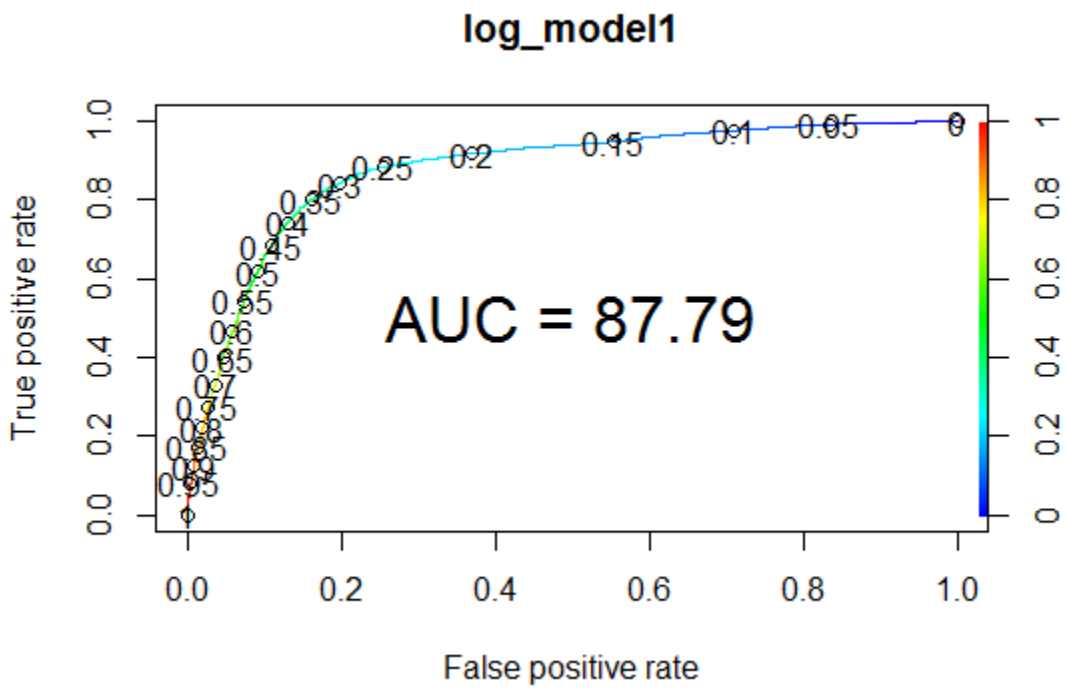
```
#library(ROCR)
#pred <- prediction(prob_train, trainData$Target)
#perf <- performance(pred, measure="tpr", x.measure="fpr")
```

Plot the ROC curve using the extracted performance measures (TPR and FPR)

```
#dev.off()
#plot(perf, main = "log_model1", col=rainbow(10), colorize=T,
#print.cutoffs.at=seq(0,1,0.05))
#perf_auc <- performance(pred, measure="auc")
#auc <- perf_auc@y.values[[1]]
#print(auc)
```

AUC - 0.877

```
#lbl <- paste0("AUC = ",round(auc*100,digits = 2))
#text(x = 0.5,y = 0.5,labels = lbl ,cex = 2)
```



Here fpr is imp so choosing the cutoff at 0.4 (Because fpr = 1-specificity)

(Specificity is imp as we have to reduce the false positives. i.e, we predicted that product is returned but actually it is not returning .We may think that the product is being returned and lower the sales of that product which may cause loss to the business.)

```
#pred_class <- ifelse(prob_train>0.4,1,0)
#pred_test <- ifelse(prob_test>0.4,1,0)
```

Training metrics

```
#confusionMatrix(trainData$Target,pred_class,positive = "1")
```

 # Accuracy - 82.8, Kappa - 0.61
 # Specificity : 0.8732 , Sensitivity : 0.7371

Test metrics

```
#confusionMatrix(testData$Target,pred_test,positive = "1")
```

#Accuracy - 82.06, Kappa - 0.595
#Specificity : 0.8698 , Sensitivity : 0.7233

Model Tuning :

Implementing StepAIC to select best features and for checking multicollinearity.

Using stepAIC to improve the model

```
#library(MASS)
#model2 <- stepAIC(log_model1)
```

No features were dropped.

Checking VIF:

Use vif to find any multi-collinearity

```
#library(car)
#model_vif = vif(model2)
#model_vif
```

Building the model with removing attributes with high vif

```
#log_model3 <- glm(Target ~ IV + A + TR + QV + SO + RF1 + RF3,trainData,family =
#binomial)
#summary(log_model3)
```

#AIC 26509

Evaluating the error metrics

```
#prob_train <- predict(log_model3,trainData,type = "response")
#prob_test <- predict(log_model3,testData,type = "response")
```

Plotting ROC Curve and finding AUC

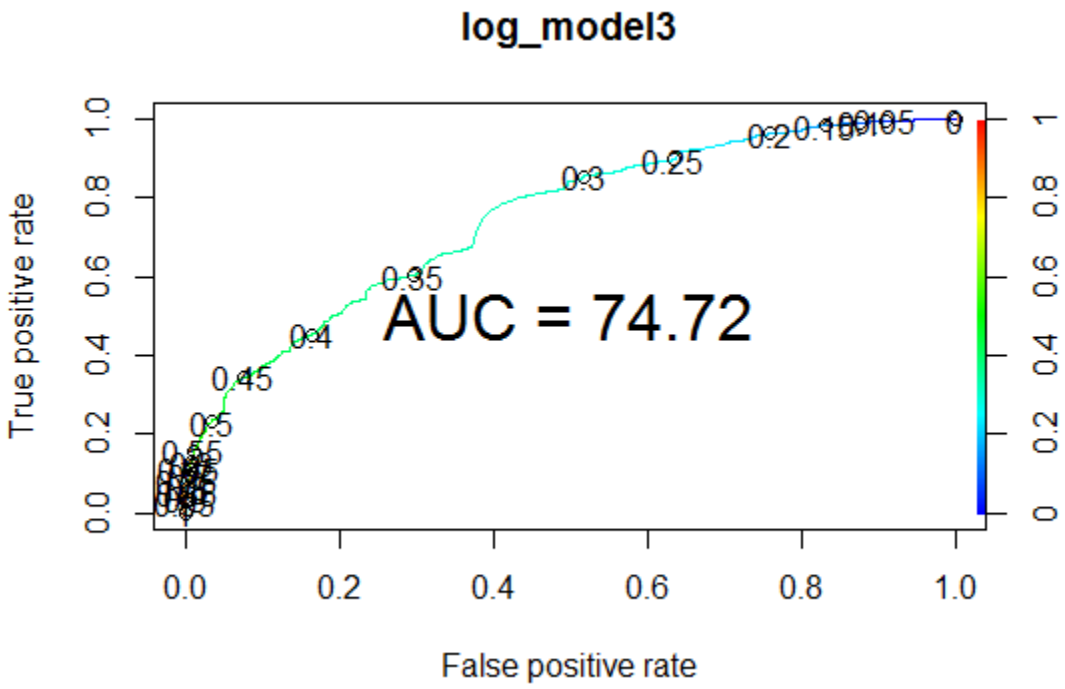
```
#library(ROCR)
#pred <- prediction(prob_train, trainData$Target)
#perf <- performance(pred, measure="tpr", x.measure="fpr")
```

Plot the ROC curve using the extracted performance measures (TPR and FPR)

```
#plot(perf, main = "log_model3",col=rainbow(10), colorize=T,
#print.cutoffs.at=seq(0,1,0.05))
#perf_auc <- performance(pred, measure="auc")
#auc <- perf_auc@y.values[[1]]
#print(auc)
```

 # AUC - 74.72

```
#lbl <- paste0("AUC = ",round(auc*100,digits = 2))
#text(x = 0.5,y = 0.5,labels = lbl ,cex = 2)
```



Here fpr is imp so choosing the cutoff at 0.4

```
#pred_class <- ifelse(prob_train>0.4,1,0)
#pred_test <- ifelse(prob_test>0.4,1,0)
```

Training metrics

```
#confusionMatrix(trainData$Target,pred_class,positive = "1")
```

 # Accuracy - 71.02, kappa - 0.306
 # Sensitivity : 0.5766 , Specificity : 0.7569

Test metrics

```
#confusionMatrix(testData$Target,pred_test,positive = "1")
```

 # Accuracy - 70.89, kappa - 0.301
 # Sensitivity : 0.5745 , Specificity : 0.7553

Not better than model1.

2. Model-2(Logistic Regression with Lasso Regularization):

Building model using glmnet library (lasso regularization)

```
#library(glmnet)
#str(data)
```

As the lasso regularization takes only numeric values,
Converting the cat attributes into dummies using model.matrix

```
#xfactors <- model.matrix(data_trans$Target ~ data_trans$RF1 +
#data_trans$RF3)[,-1]
#xfactors
#dim(xfactors)
#num_atr <- data_trans[,c(1:15)]
#str(num_atr)
```

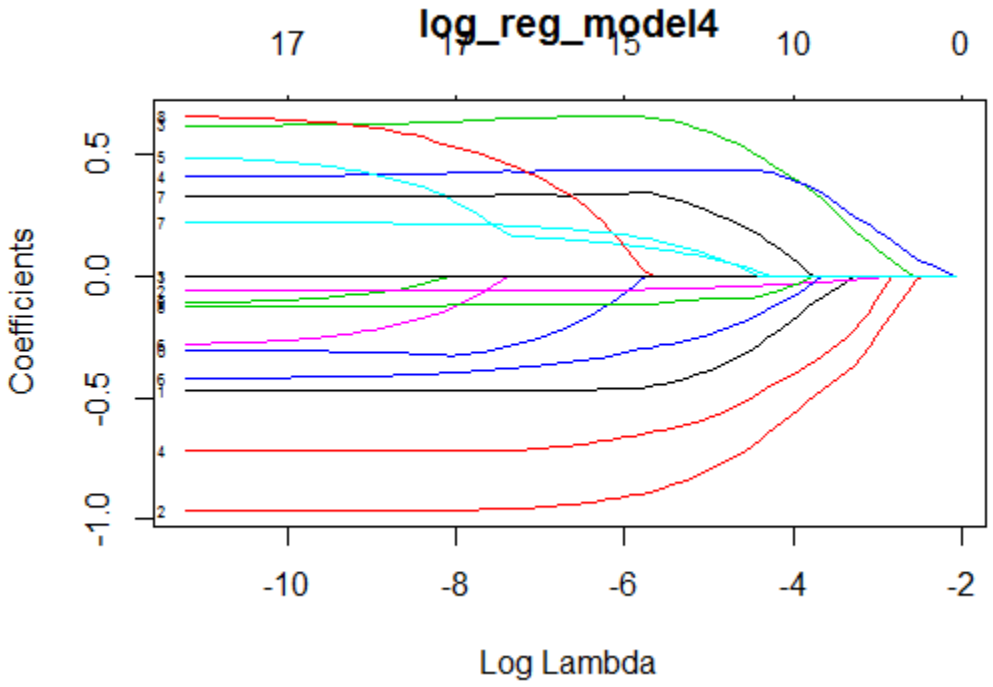
Converting the train and test data as matrix

```
#train1 <- as.matrix(data.frame(num_atr,xfactors))[trainRows,]
#test1 <- as.matrix(data.frame(num_atr,xfactors))[-trainRows,]

#yTrain <- data_trans$Target[trainRows]
#yTest <- data_trans$Target[-trainRows]
```

Implementing the lasso logistic regression

```
#log_reg_model4 <- glmnet(train1,yTrain,alpha = 1,family = "binomial")
#summary(log_reg_model4)
#plot(log_reg_model4,xvar="lambda",label=TRUE,main="log_reg_model4")
#coef(log_reg_model4)
```

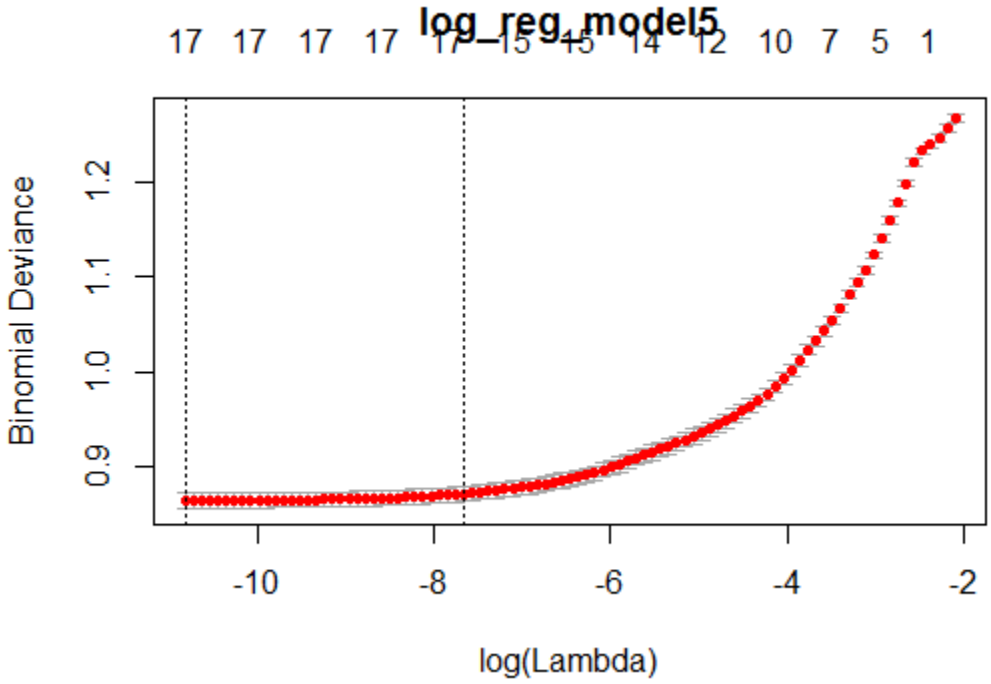


Model/Parameter Tuning :

Selecting best lambda using cross validation

```
#log_reg_model5 <- cv.glmnet(train1,yTrain,alpha = 1,family = "binomial")
#coef(log_reg_model5)
```

```
#plot(log_reg_model5,main="log_reg_model5")
```



Evaluating the error metrics

```
#prob_train <- predict(log_reg_model5,train1,type = "response")
#prob_test <- predict(log_reg_model5,test1,type = "response")
```

Calculating the auc

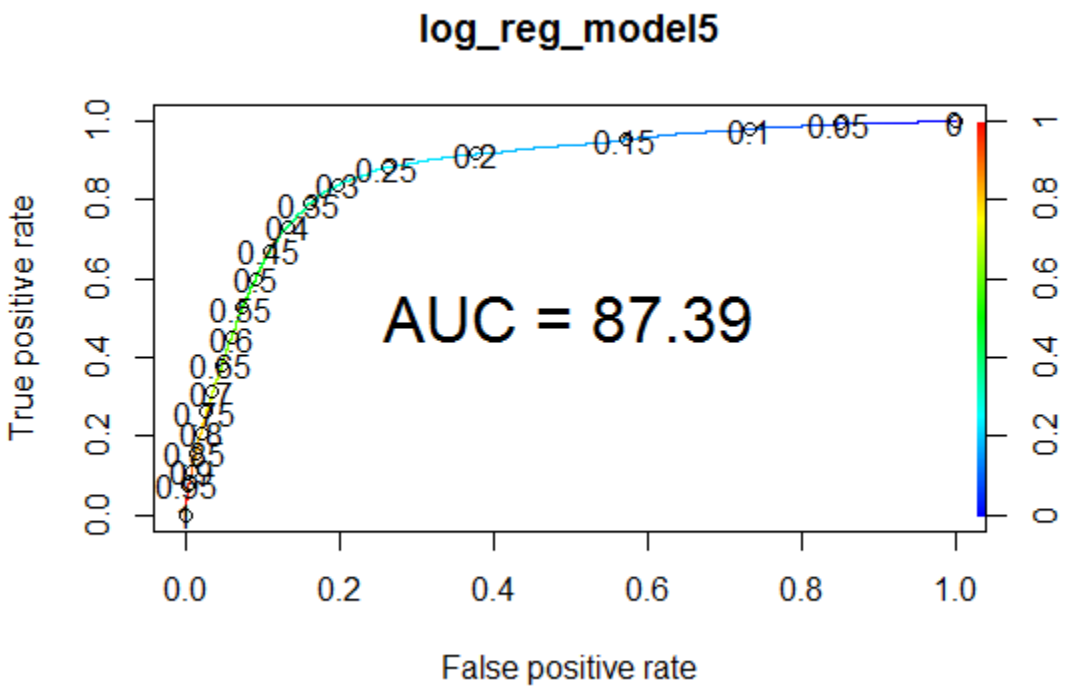
```
#library(ROCR)
#pred <- prediction(prob_train, yTrain)
#perf <- performance(pred, measure="tpr", x.measure="fpr")
```

Plot the ROC curve using the extracted performance measures (TPR and FPR)

```
#plot(perf, main="log_reg_model5", col=rainbow(10), colorize=T,
#print.cutoffs.at=seq(0,1,0.05))
#perf_auc <- performance(pred, measure="auc")
#auc <- perf_auc@y.values[[1]]
#print(auc)
```

 # AUC - 87.38

```
#lbl <- paste0("AUC = ",round(auc*100,digits = 2))
#text(x = 0.5,y = 0.5,labels = lbl ,cex = 2)
```



Here fpr is imp so choosing the cutoff at 0.4

```
#pred_class <- ifelse(prob_train>0.4,1,0)
#pred_test <- ifelse(prob_test>0.4,1,0)
```

Training metrics

```
#confusionMatrix(yTrain,pred_class,positive = "1")
```

 # Accuracy - 82.3, kappa - 0.599
 # Sensitivity : 0.7312 , Specificity : 0.8683

Test metrics

```
#confusionMatrix(yTest,pred_test,positive = "1")
```

 # Accuracy - 81.6 , kappa - 0.585
 # Sensitivity : 0.7199 , Specificity : 0.8646

Almost similar to model1.

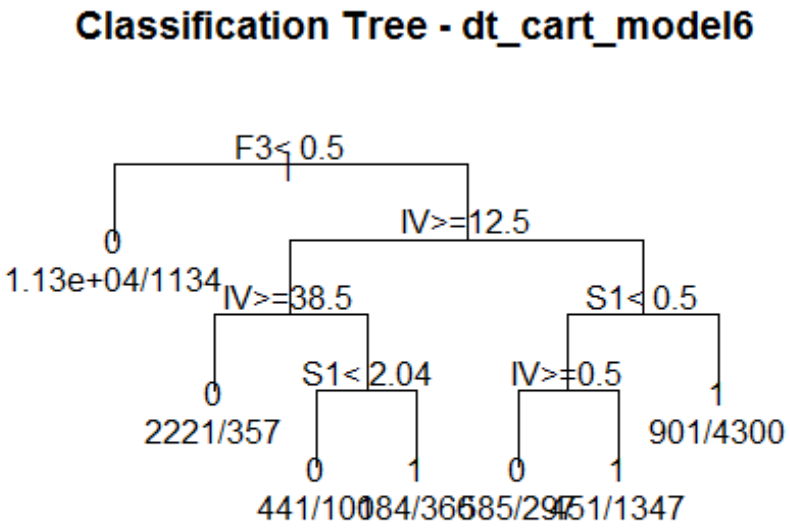
3. Model-3(Decision Trees Using rpart):

Decision trees classification

```
#library(rpart)
#dt_cart_model6 <- rpart(Target~.,data = trainData,method = "class")
```

Plotting the classification tree

```
#plot(dt_cart_model6,main="Classification Tree -
#dt_cart_model6",margin=0.15,uniform=TRUE)
#text(dt_cart_model6,use.n=T)
#summary(dt_cart_model6)
```



Error metrics on train and test

```
#trainpred <- predict(dt_cart_model6,trainData,type = "class")
#testpred <- predict(dt_cart_model6,testData,type = "class")
```

Training Metrics

```
#confusionMatrix(trainData$Target,trainpred)
```

 # Accuracy - 85.7 , kappa - 0.673

 # Sensitivity : 0.8851 , Specificity : 0.7965

Testing Metrics

```
#confusionMatrix(testData$Target,testpred)
```

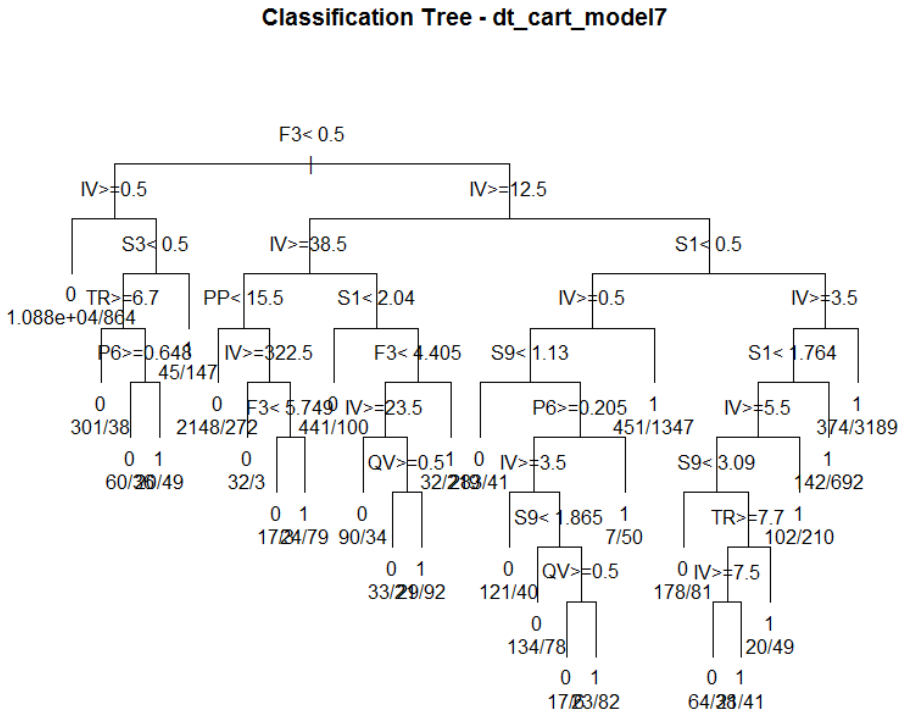
 # Accuracy - 85.11 , kappa - 0.659

 # Sensitivity : 0.8813 , Specificity : 0.7858

Parameter Tuning:

Updating the cp value and building the model using best cp

```
##Importance of cp parameter
#dtCart=rpart(Target ~.,data=trainData,method="class", cp=0.001)
#printcp(dtCart)
#dt_cart_model7 <- rpart(Target~.,trainData,method = "class",cp=0.00139)
#plot(dt_cart_model7,main="Classification Tree -
#dt_cart_model7",margin=0.15,uniform=TRUE)
#text(dt_cart_model7,use.n=T)
#summary(dt_cart_model7)
```



Evaluating the model

```
#trainpred <- predict(dt_cart_model7,trainData,type = "class")
#testpred <- predict(dt_cart_model7,testData,type = "class")
```

Training Metrics

```
#confusionMatrix(trainData$Target,trainpred)
```

 # Accuracy - 87.72 , kappa - 0.718
 # Sensitivity : 0.8994 , Specificity : 0.8288

Testing Metrics

```
#confusionMatrix(testData$Target,testpred)
```

 # Accuracy - 86.96 , kappa - 0.701
 # Sensitivity : 0.8949 , Specificity : 0.8149

4. Model-4(Decision Tress using c5.0):

Loading library for C5.0

```
#library(C50)
#calling C5.0 function
#dt_C50_model8= C5.0(Target ~ ., data = trainData, rules=TRUE)
#summary(dt_C50_model8)
```

Printing the importance of attributes used for building tree

```
#C5imp(dt_C50_model8,pct = T)
```

Evaluating the model

```
#trainpred <- predict(dt_C50_model8,trainData,type = "class")
#testpred <- predict(dt_C50_model8,testData,type = "class")
```

Training Metrics

```
#confusionMatrix(trainData$Target,trainpred)
```

 # Accuracy - 90.14 , kappa - 0.776
 # Sensitivity : 0.9235 , Specificity : 0.8556

Testing Metrics

```
#confusionMatrix(testData$Target,testpred)
```

 # Accuracy - 88.57 , kappa - 0.741
 # Sensitivity : 0.9159 , Specificity : 0.8246

Conclusion:

So finally decision trees models are giving better accuracy for this problem

Model : dt_c50_model8 is the best model
for predicting whether the customers return the product or not