University of Westminster

Data Mining and Machine Learning Coursework 2

Prepared for
Dr Panagiotis Chountas

By
Mohamed BaniHani
174359162

7th January 2020

Data Set Selection and Visualisation

1.0 Introduction:

The dataset that was selected is the Online Shoppers Purchasing Intention from the UCI website. The dataset contains 12330 sessions, and each session would belong to a different user in one year period. The dataset consists of 10 numerical attributes and eight categorical attributes. The 'Revenue' attribute is the class label which contains two classes 'yes' and 'no.'

Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" and "Product-Related Duration" attributes represent the type of pages visited by the user. The value of "Bounce Rate" feature for a web page refers to the percentage of visitors who enter the site. The amount of "Exit Rate" feature for a specific web page is calculated as for all pageviews to the page, the percentage that was the last in the session. The "Page Value" feature represents the average value for a web page that a user visited before completing an e-commerce transaction. The "Special Day" feature indicates the closeness of the site visiting time to a specific special day (e.g. Mother's Day, Valentine's Day) in which the sessions are more likely to be finalised with the transaction. The dataset also includes an operating system, browser, region, traffic type, visitor type as returning or new visitor, a Boolean value indicating whether the date of the visit is weekend, and month of the year.

```
head(shop)
```

```
head(shop)
 Administrative Administrative_Duration Informational Informational_Duration ProductRelated ProductRelated_Duration BounceRates
              0                       0             0                      0              1                0.000000  0.20000000
              0                       0             0                      0              2               64.000000  0.00000000
              0                       0             0                      0              1                0.000000  0.20000000
              0                       0             0                      0              2                2.666667  0.05000000
              0                       0             0                      0             10              627.500000  0.02000000
              0                       0             0                      0             19              154.216667  0.01578947
 ExitRates PageValues SpecialDay       VisitorType Revenue
 0.2000000          0          0 Returning_Visitor      No
 0.1000000          0          0 Returning_Visitor      No
 0.2000000          0          0 Returning_Visitor      No
 0.1400000          0          0 Returning_Visitor      No
 0.0500000          0          0 Returning_Visitor      No
 0.0245614          0          0 Returning_Visitor      No
```

2.0 Pre-Processing:

Before building an Ensemble type Classifier, there are three pre-processing steps must be considered

2.1 Removing unwanted columns and N/A Values:

There are six columns in the dataset which is unnecessary and will affect the analysis

Month, operating systems, Browser, Region, TrafficType and Weekend

The below code will remove the unwanted columns and show if there are missing values

```
shop[11:15] <- NULL

shop[12] <- NULL

anyNA(shop)
```

The above code will result in having 12330 obs. of 12 variables and return FALSE for N/A values

2.2 changing data types:

The Class label "Revenue " should be a factor and the rest of the data set is numeric

```
str(shop)
```

```
'data.frame':	12330 obs. of  12 variables:
 $ Administrative        : int  0 0 0 0 0 0 0 1 0 0 ...
 $ Administrative_Duration: num  0 0 0 0 0 0 0 0 0 0 ...
 $ Informational         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Informational_Duration : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ProductRelated        : int  1 2 1 2 10 19 1 0 2 3 ...
 $ ProductRelated_Duration: num  0 64 0 2.67 627.5 ...
 $ BounceRates           : num  0.2 0 0.2 0.05 0.02 ...
 $ ExitRates             : num  0.2 0.1 0.2 0.14 0.05 ...
 $ PageValues            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ SpecialDay            : num  0 0 0 0 0 0 0.4 0 0.8 0.4 ...
 $ VisitorType           : Factor w/ 3 levels "New_Visitor",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ Revenue               : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

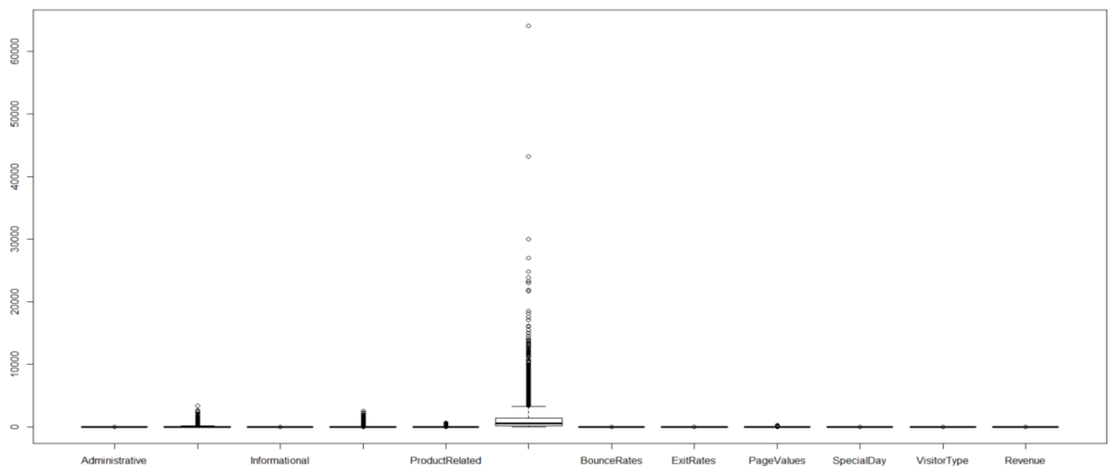
Visitor type have a factor type which will affect the model next step is to convert it to an integer ty pe

```
shop$VisitorType<-as.integer(shop$VisitorType)
```

2.3 Removing outliers:
The last step in pre-processing is to remove outliers

```
boxplot(shop)
```

The above code will generate a graph which contains outliers for each attribute



The above graph shows that Product-Related Duration has the most outlier
The below code will result in deleting the outlier from the Product Related Duration column

```
outliers <- boxplot(shop$ProductRelated_Duration, plot=FALSE)$out

shop[which(shop$ProductRelated_Duration %in% outliers),]

shop<- shop[-which(shop$ProductRelated_Duration %in% outliers),]
```

The total number of outliers removed for the original dataset is 961 which
3.0 Formation of Training and Test Set:
After having the data set ready the next step is to split the data into training and testing

```
set.seed(3033)

intrain <- createDataPartition(y = shop$Revenue, p= 0.75, list = FALSE)

training <- shop[intrain,]

testing <- shop[-intrain,]
```

The training set contains 75% of the original data-set, and that leaves 20% for the testing set.

The training set contains 9248 obs.85 present is ‘No’ value and 15 present is ‘True’ value

```
percentagetrain <- prop.table(table(training$Revenue)) * 100

cbind(freq=table(training$Revenue), percentage=percentagetrain)
```

Output of the above code:

```
freq percentage
No  7817   84.52638
Yes 1431   15.47362
```

The Step after splitting the data is to build a training control for Bagging Algorithms and Stacking Algorithms

```
# define training control for bagging

bagging.control <- trainControl(method="repeatedcv", number=10,
repeats=3,preProc=c("center","scale"))

# define training conteol of stacking

stack.control <- trainControl(method="repeatedcv", number=10, repeats=3,

                    savePredictions=TRUE, classProbs=TRUE,preProc=c("center","scale"))
```

The above code is will generate a Repeated K-fold Cross Validation for both bagging and stacking algorithms.

4.0 Bagging Type Classifier:

After building the training model the next phase is to construct train and test Bagging type classifier based on Bagged CART and Random Forest.

4.1 Bagged Cart Model:

```
library(caret)

seed <- 7

metric <- "Accuracy"

set.seed(seed)

modle.treebag <- train(Revenue~., data=training, method="treebag", metric=metric,
trControl=bagging.control,preProc=c("center","scale"))

print(modle.treebag)
```

The above will create Bagged Cart Model based on training data which will result in:

```
Bagged CART

8528 samples
  11 predictor
   2 classes: 'No', 'Yes'

Pre-processing: centered (11), scaled (11)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 7675, 7676, 7675, 7676, 7676, 7675, ...
Resampling results:

  Accuracy  Kappa
  0.90502   0.5786408
```

4.2 Random Forest model:

```
seed <- 7

metric <- "Accuracy"

set.seed(seed)

model.rf <- train(Revenue~., data=training, method="rf", metric=metric,
trControl=control,preProc=c("center","scale"))

print(model.rf)
```

The above code will create Random Forest Model based on training data which will result in:

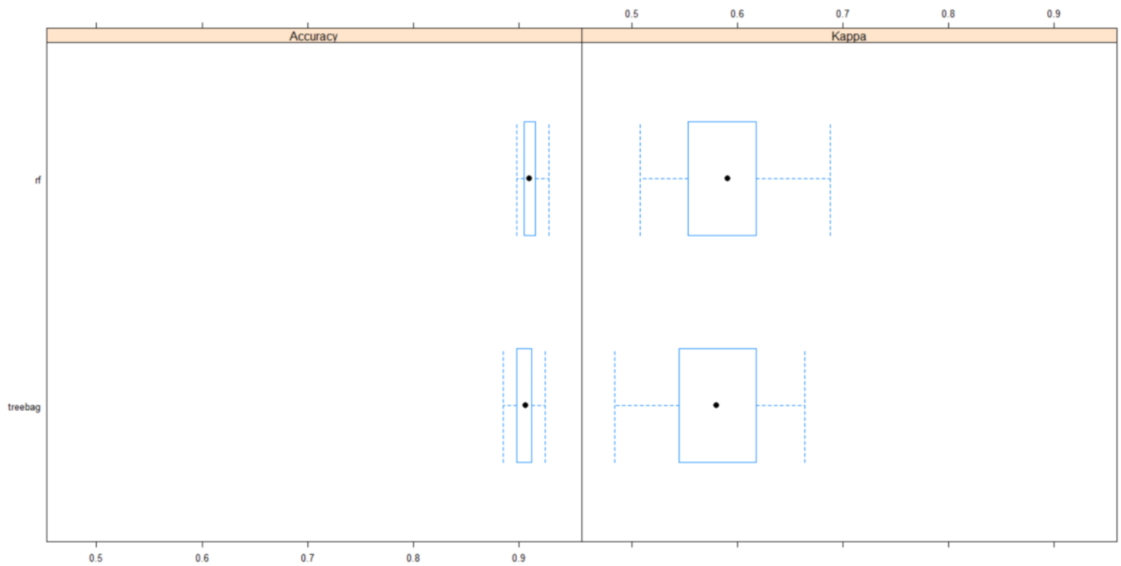
```
Random Forest

8528 samples
  11 predictor
   2 classes: 'No', 'Yes'

Pre-processing: centered (11), scaled (11)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 7675, 7676, 7675, 7676, 7676, 7675, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.9109211  0.5910406
   6    0.9092797  0.5935890
  11    0.9078337  0.5872961

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

After having the two models ready the next step is to combine the two models using the method resamples and present the accuracy of the two models in a graph

```
# combine the models

bagging_results <- resamples(list(treebag=fit.treebag, rf=fit.rf))

summary(bagging_results)

bwplot(bagging_results)
```

The output of this code is:

```
Call:
summary.resamples(object = bagging_results)

Models: treebag, rf
Number of resamples: 30

Accuracy
             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
treebag 0.8849765 0.8984192 0.9061583 0.9050200 0.9117045 0.9249707    0
rf      0.8980070 0.9052230 0.9098361 0.9109211 0.9153240 0.9284038    0

Kappa
             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
treebag 0.4837910 0.5458137 0.5800023 0.5786408 0.6146263 0.6641112    0
rf      0.5081162 0.5559042 0.5904827 0.5910406 0.6156844 0.6879158    0
```



As shown in the graph Random Forest and Bagged Cart have a close accuracy

With 0.9050200 for Bagged Cart and 0.9109211 for Random Forest

5.0 Stacking Type Classifier:

Stack Type Classifier is used to combine models and show the best accuracy between the number of models, in this report we are using three models

1. CART
2. KNN
3. NB

```
library(caretEnsemble)
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=3,

                   savePredictions=TRUE, classProbs=TRUE,preProc=c("center","scale"))

algorithmList <- c( 'rpart', 'knn', 'nb')

set.seed(seed)

models <- caretList(Revenue~., data=training, trControl=stack.control,
methodList=algorithmList)
```

The output of this code is:

```
$rpart
CART

8528 samples
  11 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 7674, 7675, 7676, 7675, 7675, 7676, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.02798354  0.9061920  0.5697790
  0.07654321  0.8977475  0.5885508
  0.27407407  0.8850087  0.4613263

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.02798354.

$knn
k-Nearest Neighbors

8528 samples
  11 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 7674, 7675, 7676, 7675, 7675, 7676, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.8830148  0.3832008
  7  0.8835221  0.3567648
  9  0.8822712  0.3304890

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.

$nb
Naive Bayes

8528 samples
  11 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
```

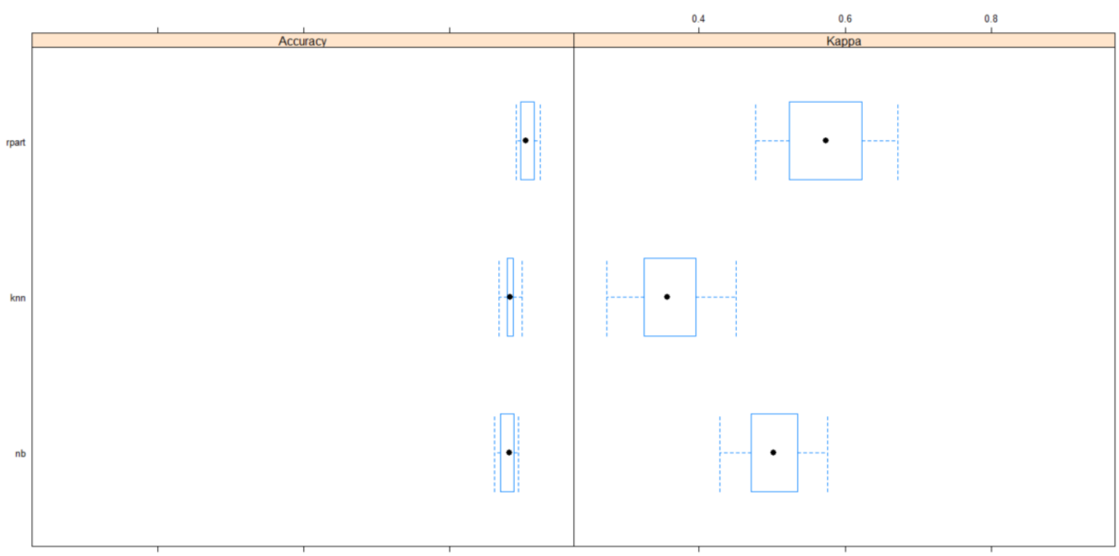
```
Summary of sample sizes: 7674, 7675, 7676, 7675, 7675, 7676, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.7934976  0.3831438
   TRUE      0.8794952  0.4999045

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = TRUE and a
djust = 1.
```

After having the three models ready the next step is to combine the two models using the method resamples and present the accuracy of the three models in a graph

```
results <- resamples(models)

summary(results)

bwplot(results)
```

The above code will show the summary of all three models and show a graph that contains each model and its accuracy

```
Models: rpart, knn, nb
Number of resamples: 30

Accuracy
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rpart 0.8909730 0.8968649 0.9044554 0.9061920 0.9152246 0.9238876    0
knn   0.8675264 0.8792851 0.8826291 0.8835221 0.8874230 0.8992974    0
nb    0.8616647 0.8702406 0.8815944 0.8794952 0.8884977 0.8943662    0

Kappa
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rpart 0.4773671 0.5250693 0.5739547 0.5697790 0.6224141 0.6724262    0
knn   0.2735720 0.3264462 0.3561119 0.3567648 0.3934796 0.4512961    0
nb    0.4285673 0.4720985 0.5015100 0.4999045 0.5328320 0.5753259    0
```



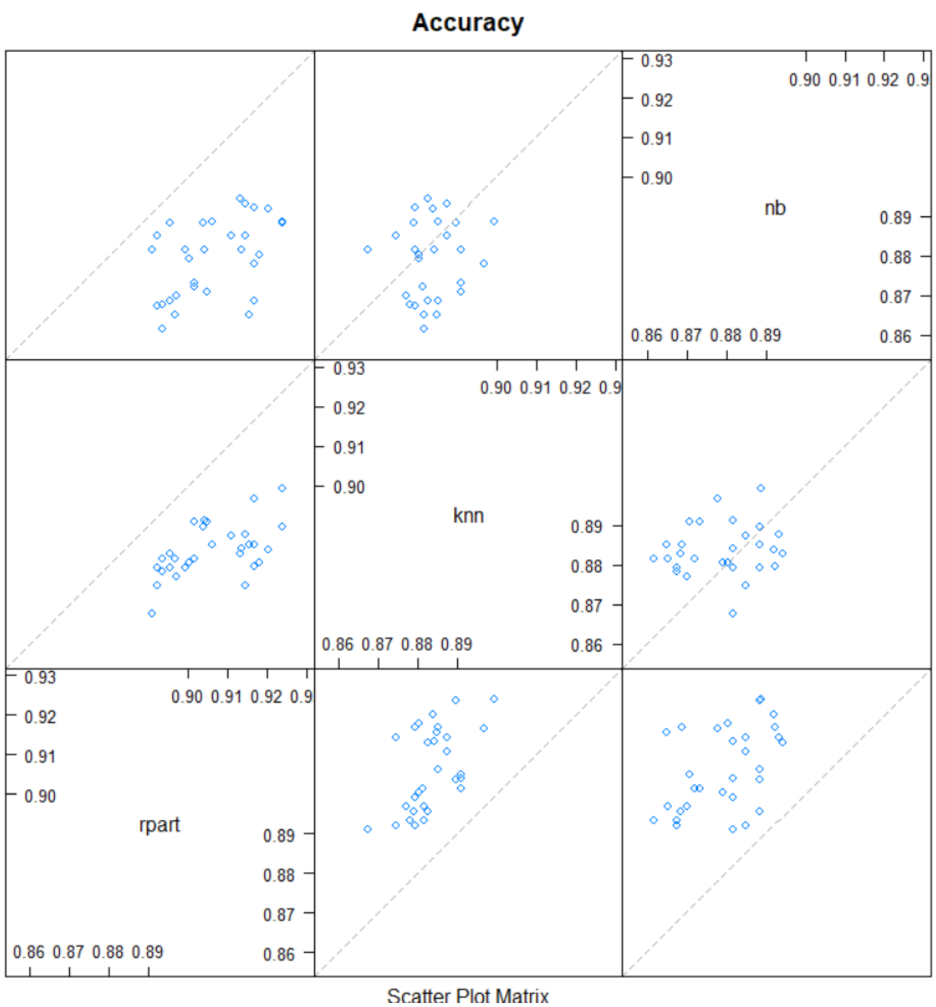
In conclusion, based on the Stacking model the method rpart have the best accuracy with a value equals to **0.9061920**

Next is to check the correlation between these models and pick the two models that have the high est correlation for the purpose of increasing the accuracy.

```
modelCor(results)

splom(results)
```

The output of the above code is:

```
      rpart       knn         nb
rpart 1.0000000 0.5424737 0.4755461
knn   0.5424737 1.0000000 0.1119082
nb    0.4755461 0.1119082 1.0000000
```



Based on the previous graph, the best correlation is between rpart and knn.

Next step is combing the prediction of the Stacking classifier with CART and Knn

```
set.seed(seed)

stack.rf <- caretStack(models, method="rpart", metric="Accuracy", trControl=stackControl)

print(stack.rf)
```

```
A rpart ensemble of 3 base models: rpart, knn, nb

Ensemble results:
CART

25584 samples
    3 predictor
    2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 23026, 23027, 23025, 23025, 23026, 23026, ...
Resampling results across tuning parameters:

  cp           Accuracy   Kappa
  0.008504801  0.9058919  0.5700778
  0.068861454  0.8995339  0.5745767
  0.274622771  0.8761469  0.3198156

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.008504801.
```

By using the rpart model the accuracy dropped from 0**.9061920 to 0.9058919**

Next is using KNN to improve the accuracy

```
set.seed(seed)
stack.nb <- caretStack(models, method="knn", metric="Accuracy", trControl=stackControl)
print(stack.nb)
```

```
A knn ensemble of 3 base models: rpart, knn, nb

Ensemble results:
k-Nearest Neighbors

25584 samples
    3 predictor
    2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 23025, 23026, 23025, 23026, 23025, 23026, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9042500  0.5728679
  7  0.9056440  0.5790975
  9  0.9066473  0.5833099

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

By using Knn to improve the accuracy, we can notice that the accuracy grew from **0.9061920 to 0.9066473 by using k = 9**

6.0 Measure Performance:

The following metrics are used in measuring the performance of each ensemble type classifier:

1. Confusion matrix

A) Confusion matrix for Bagging Algorithm

Bagging Algorithm contains the Bagged CART model and Random Forest Model

```
treeteest <- predict(modle.treebag,newdata = testing)

confusionMatrix(data = treeteest,testing$Revenue)
```

The output of the above code is:

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  2331  177
       Yes  106  227

               Accuracy : 0.9004
                 95% CI : (0.8888, 0.9112)
    No Information Rate : 0.8578
    P-Value [Acc > NIR] : 6.111e-12

                  Kappa : 0.5594

 Mcnemar's Test P-Value : 3.168e-05

            Sensitivity : 0.9565
            Specificity : 0.5619
         Pos Pred Value : 0.9294
         Neg Pred Value : 0.6817
             Prevalence : 0.8578
         Detection Rate : 0.8205
   Detection Prevalence : 0.8828
      Balanced Accuracy : 0.7592

       'Positive' Class : No
```

```
baggingtest <- predict(model.rf,newdata = testing)

confusionMatrix(data = baggingtest,testing$Revenue)
```

The output of the above code is:

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  2352  186
       Yes   85  218

               Accuracy : 0.9046
                 95% CI : (0.8932, 0.9152)
    No Information Rate : 0.8578
    P-Value [Acc > NIR] : 3.348e-14
```

```
                  Kappa : 0.5635

 Mcnemar's Test P-Value : 1.243e-09

            Sensitivity : 0.9651
            Specificity : 0.5396
         Pos Pred Value : 0.9267
         Neg Pred Value : 0.7195
             Prevalence : 0.8578
         Detection Rate : 0.8279
   Detection Prevalence : 0.8933
      Balanced Accuracy : 0.7524

       'Positive' Class : No
```

B) Confusion matrix for Stacking algorithm:

Stacking Algorithm contains the KNN, Naïve Bayes and Stacking CART

```
#rpart

model.rpart <- predict(models$rpart,newdata = testing)

confusionMatrix(data = model.rpart,testing$Revenue)

#Knn

model.knn <- predict(models$knn,newdata = testing)

confusionMatrix(data = model.knn,testing$Revenue)

#NB

model.nb <- predict(models$nb,newdata = testing)

confusionMatrix(data = model.nb,testing$Revenue)
```

The output for Rpart

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  2376  198
       Yes   61  206

               Accuracy : 0.9088
                 95% CI : (0.8976, 0.9192)
    No Information Rate : 0.8578
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5648

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9750
            Specificity : 0.5099
         Pos Pred Value : 0.9231
         Neg Pred Value : 0.7715
```

```
           Prevalence : 0.8578
       Detection Rate : 0.8363
 Detection Prevalence : 0.9060
    Balanced Accuracy : 0.7424

     'Positive' Class : No
```

The output for Knn

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  2390  280
       Yes   47  124

               Accuracy : 0.8849
                 95% CI : (0.8726, 0.8964)
    No Information Rate : 0.8578
    P-Value [Acc > NIR] : 1.215e-05

                  Kappa : 0.3788

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9807
            Specificity : 0.3069
         Pos Pred Value : 0.8951
         Neg Pred Value : 0.7251
             Prevalence : 0.8578
         Detection Rate : 0.8413
   Detection Prevalence : 0.9398
      Balanced Accuracy : 0.6438

       'Positive' Class : No
```

The output for NB

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  2276  169
       Yes  161  235

               Accuracy : 0.8838
                 95% CI : (0.8715, 0.8954)
    No Information Rate : 0.8578
    P-Value [Acc > NIR] : 2.556e-05

                  Kappa : 0.5199

 Mcnemar's Test P-Value : 0.7

            Sensitivity : 0.9339
            Specificity : 0.5817
         Pos Pred Value : 0.9309
         Neg Pred Value : 0.5934
             Prevalence : 0.8578
         Detection Rate : 0.8011
   Detection Prevalence : 0.8606
      Balanced Accuracy : 0.7578

       'Positive' Class : No
```

C) Confusion matrix for the prediction of Stacking classifier and the KNN mdoel:

```
models.rf <- predict(stack.rf,newdata = testing)
confusionMatrix(data = models.rf,testing$Revenue)
```

```
the above code will result in:
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  2376  198
       Yes   61  206

               Accuracy : 0.9088
                 95% CI : (0.8976, 0.9192)
    No Information Rate : 0.8578
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5648

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9750
            Specificity : 0.5099
         Pos Pred Value : 0.9231
         Neg Pred Value : 0.7715
             Prevalence : 0.8578
         Detection Rate : 0.8363
   Detection Prevalence : 0.9060
      Balanced Accuracy : 0.7424

       'Positive' Class : No
```

2. Precision VS Recall

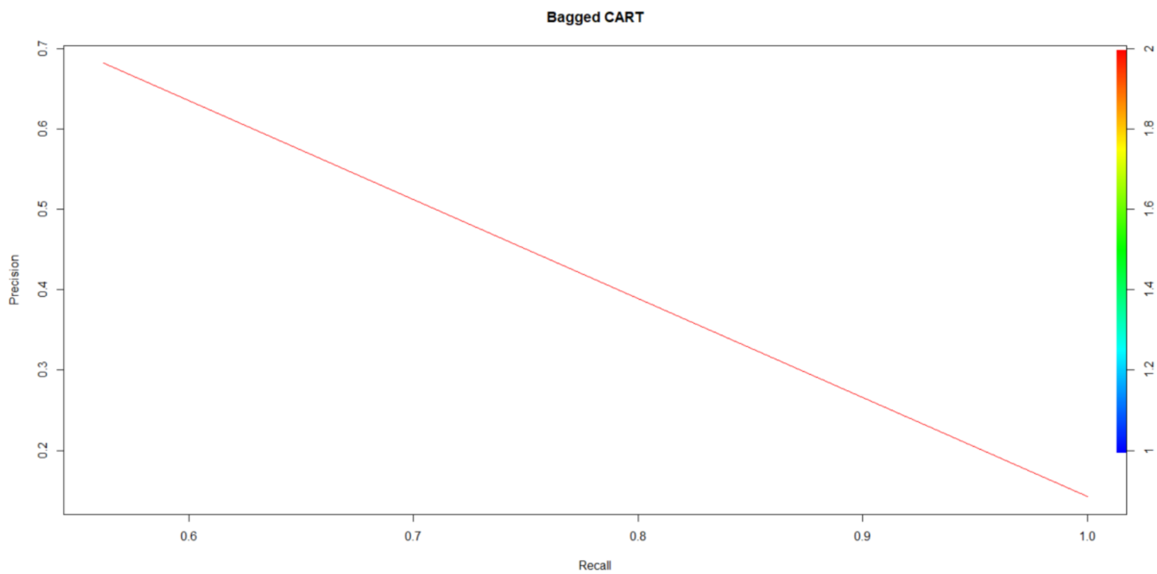
A) Precision VS Recall for Bagging Algorithm:

Bagging Algorithm contains the Bagged CART model and Random Forest Model

The code below is for Bagged CART

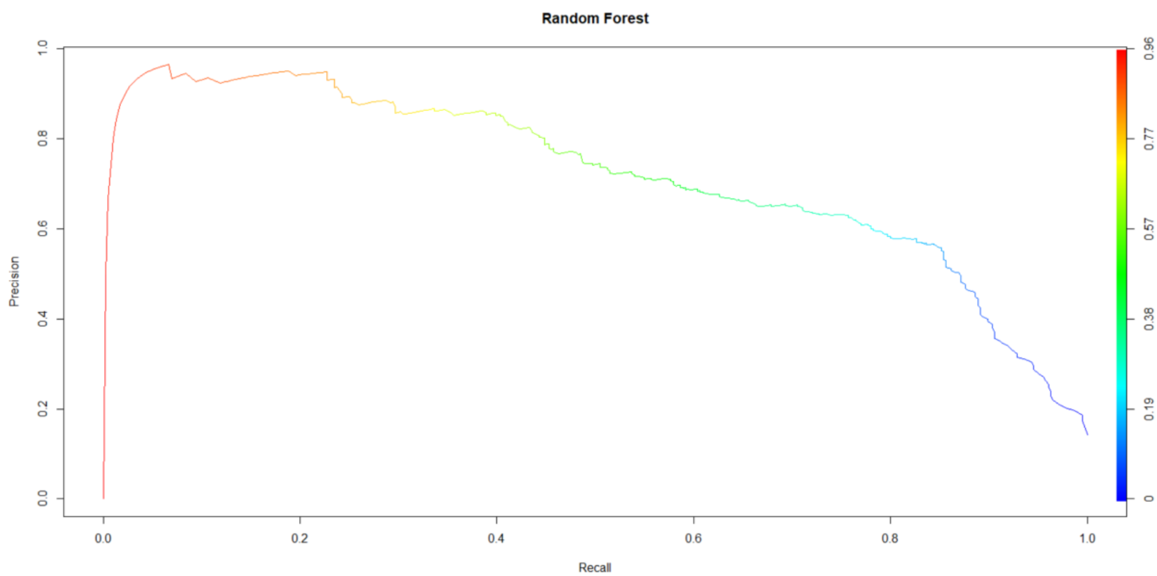
```
pre.tree <- predict(modle.treebag,newdata = testing,type = "prob")
prediction.tree <- prediction(pre.tree[,2], testing$Revenue)
#pre vs recall
perform.tree <- performance(prediction.tree,"prec","rec")
plot(perform,colorize = T,main = "Bagged CART")
```

the output of the above code is a graph that show the Precision and Recall for Bagged CART.



Next is to find the Precision and Recall for Random Forest model

```
pred.rf <- predict(model.rf,newdata = testing,type = "prob")
pred.rf <- prediction(as.numeric( pred.rf[,2]), testing$Revenue)
perform.rf <- performance(pred.rf,"prec","rec")
plot(perform.rf,colorize = T,main = "Random Forest")
```



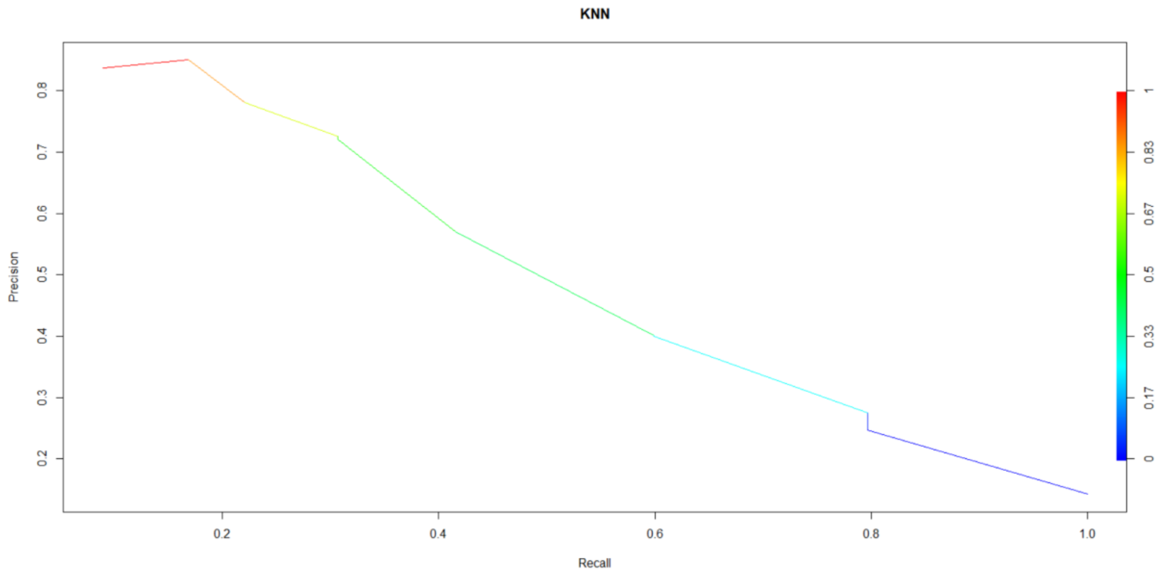
B) Precision VS Recall for Stacking Algorithm:

Stacking Algorithm contains the KNN, Naïve Bayes and Stacking CART

Firstly, KNN

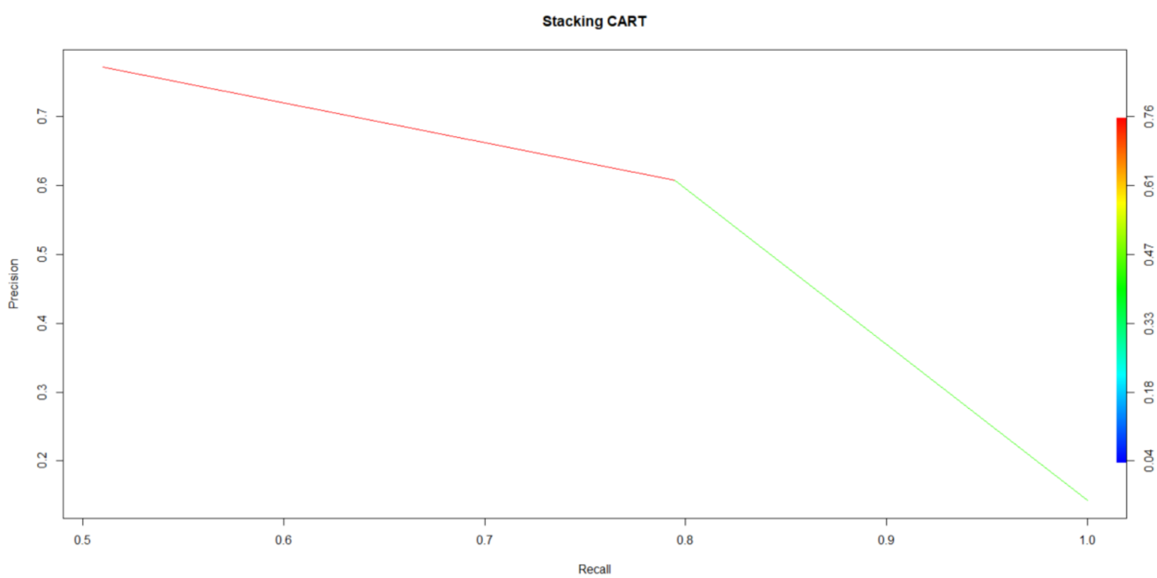
```
pred.knn <- predict(models$knn,newdata = testing,type = "prob")
```

```
pred.knn <- prediction(as.numeric( pred.knn[,2]), testing$Revenue)
perform.knn <- performance(pred.knn,"prec","rec")
plot(perform.knn,colorize = T,main = "KNN")
```



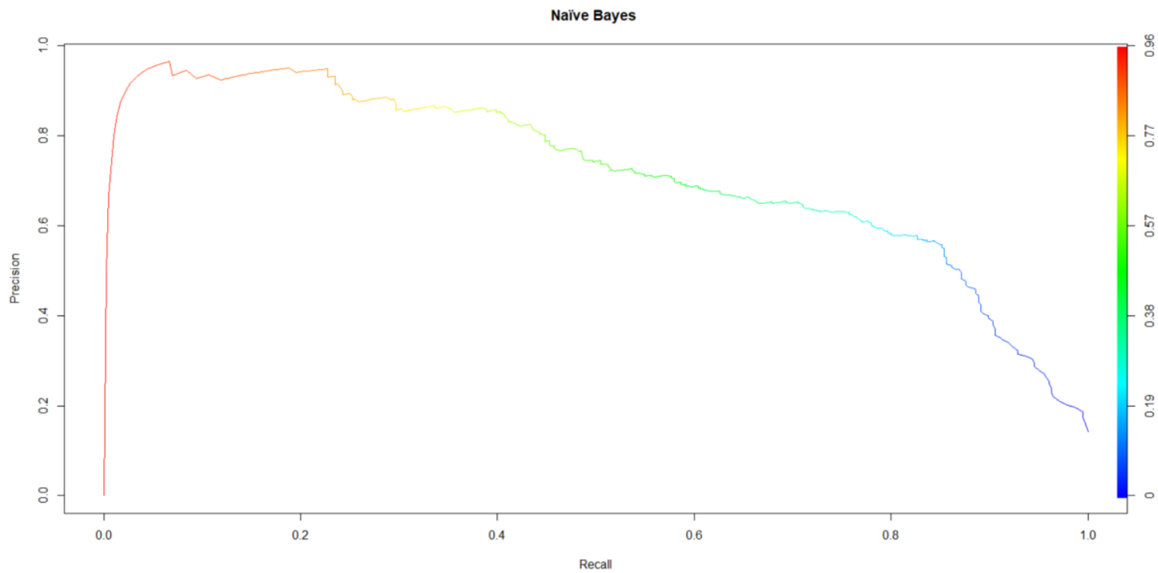
Secondly, Stacking CART

```
pred.rpart <- predict(models$rpart,newdata = testing,type = "prob")

pred.rpart <- prediction(as.numeric( pred.rpart[,2]), testing$Revenue)

perform.rpart <- performance(pred.rpart,"prec","rec")

plot(perform.rpart,colorize = T,main = "Stacking CART")
```



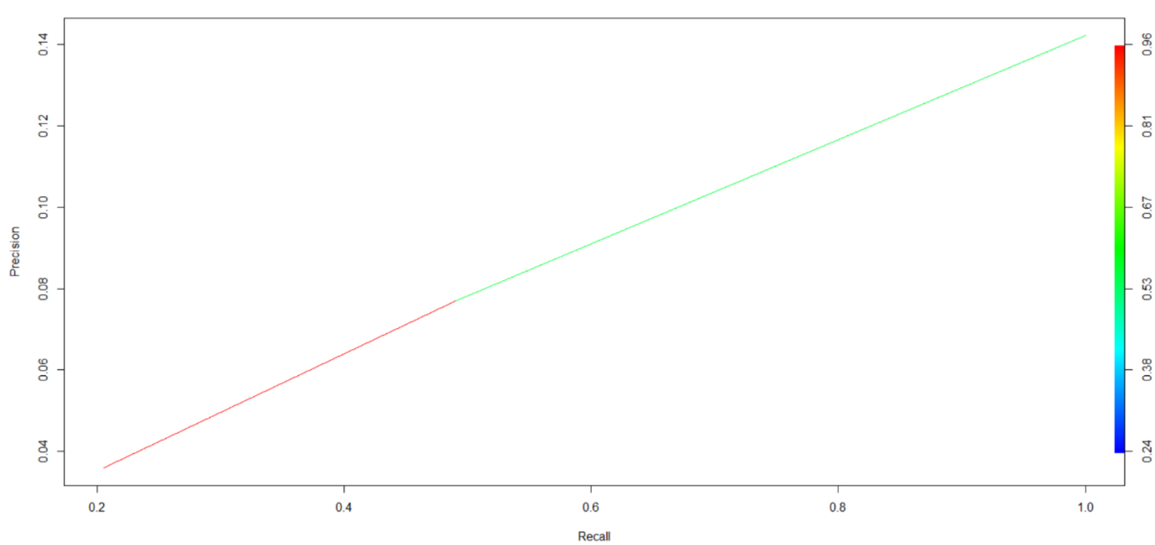
Lastly, Naïve Bayes

```
pred.nb <- predict(models$nb,newdata = testing,type = "prob")

pred.nb <- prediction(as.numeric( pred.nb[,2]), testing$Revenue)

perform.nb <- performance(pred.nb,"prec","rec")

plot(perform.rf,colorize = T,main = "Naïve Bayes ")
```



C) Precision VS Recall for the enhanced mdoel:

```
pre.rf <- predict(stack.rf,newdata = testing,type = "prob")
pre <- prediction(as.numeric( pre.rf), testing$Revenue)
pre2 <- performance(pre,"prec","rec")
plot(pre2,colorize = T)
```

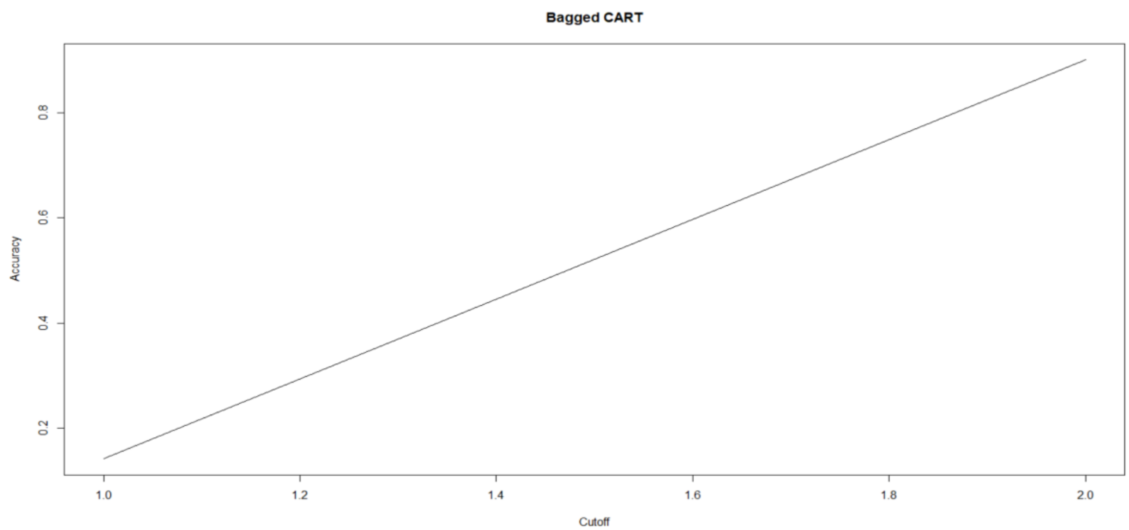


3. Accuracy:

A) Accuracy for Bagging Algorithm

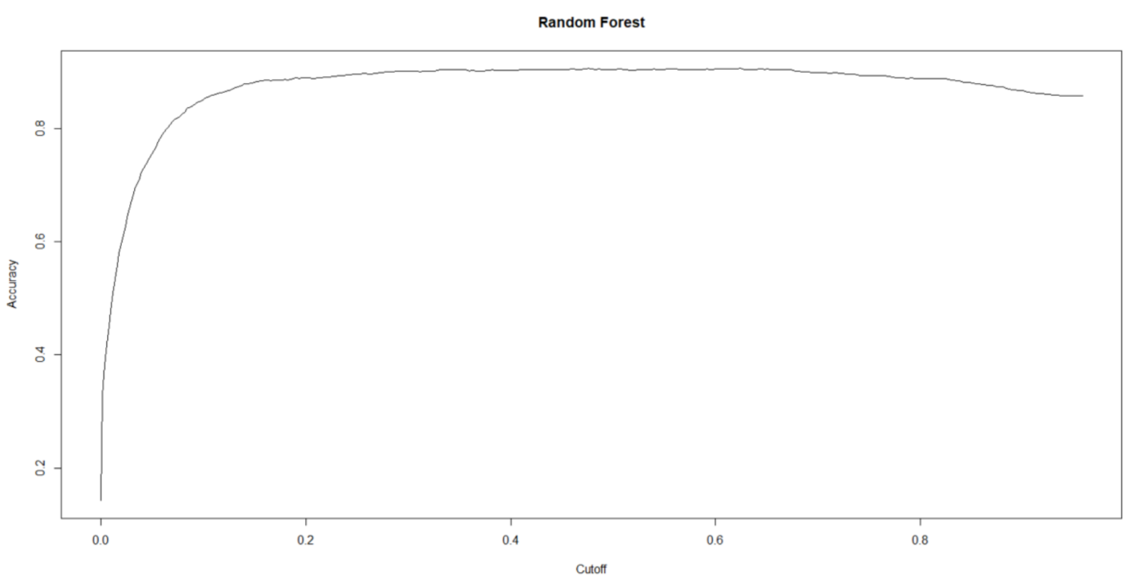
Bagging Algorithm contains the Bagged CART model and Random Forest Model
The code below is for Bagged CART

```
pre.tree <- predict(modle.treebag,newdata = testing,type = "prob")
prediction.tree <- prediction(pre.tree[,2], testing$Revenue)
acc.trr <- performance(prediction.tree,"acc")
plot(acc.trr,main = "Bagged CART")
```



The code below is for Random Forest

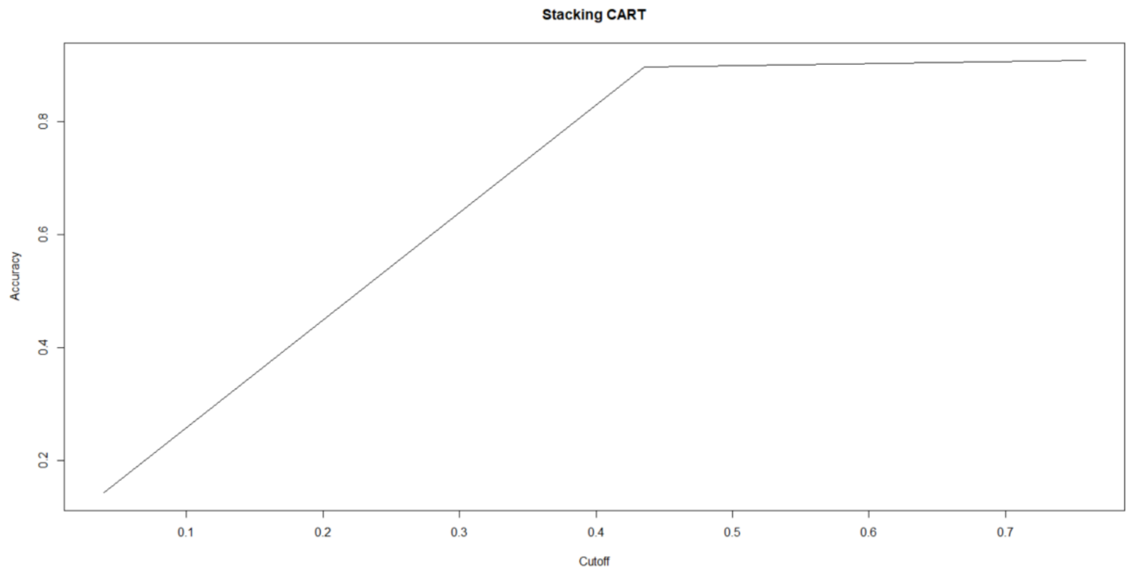
```
pred.rf <- predict(model.rf,newdata = testing,type = "prob")
pred.rf <- prediction(as.numeric( pred.rf[,2]), testing$Revenue)
acc.rf <- performance(pred.rf,"acc")
plot(acc.rf,main = "Random Forest")
```



B) Accuracy for Stacking Algorthim:
Stacking Algorithm contains the KNN, Naïve Bayes and Stacking CART

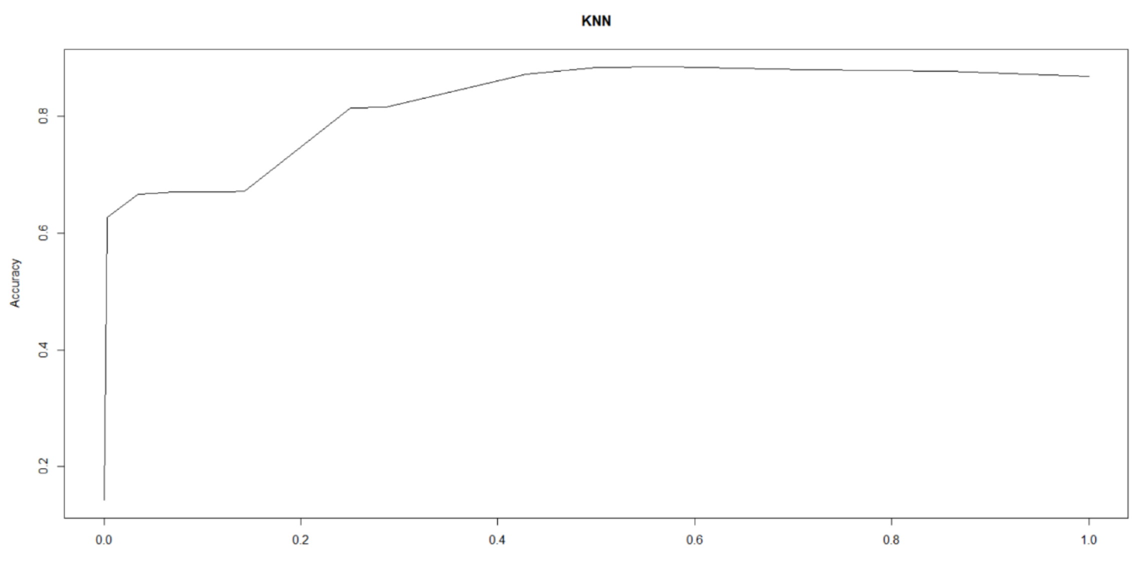
Firstly, Stacking Cart

```
pred.rpart <- predict(models$rpart,newdata = testing,type = "prob")
pred.rpart <- prediction(as.numeric( pred.rpart[,2]), testing$Revenue)
acc.rpart <- performance(pred.rpart,"acc")
plot(acc.rpart,main = "Stacking CART")
```



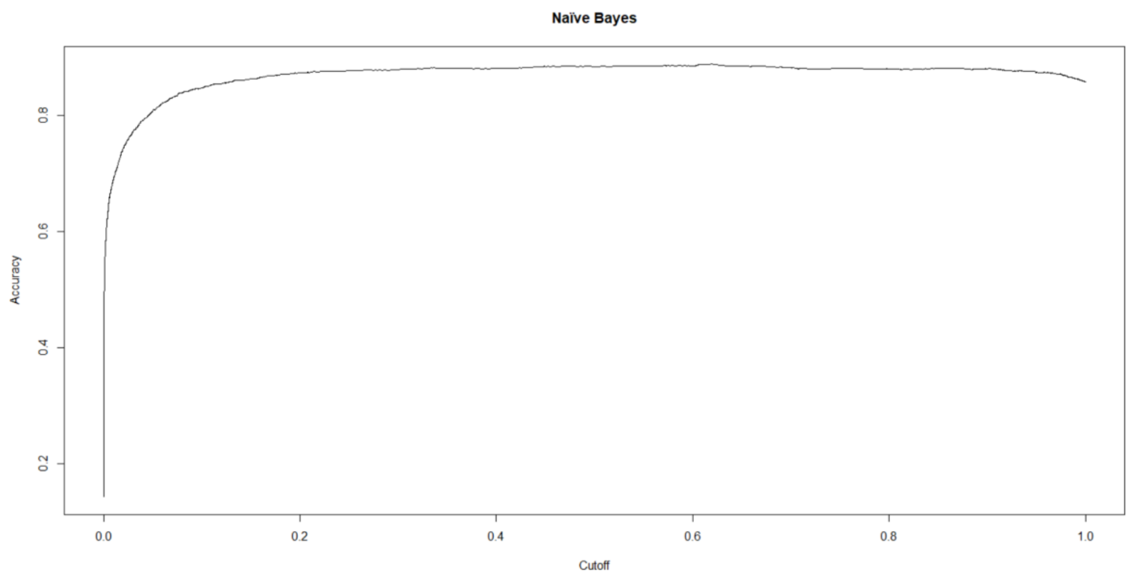
Secondly, KNN

```
pred.knn <- predict(models$knn,newdata = testing,type = "prob")
pred.knn <- prediction(as.numeric( pred.knn[,2]), testing$Revenue)
acc.knn <- performance(pred.knn,"acc")
plot(acc.knn,main = "KNN")
```



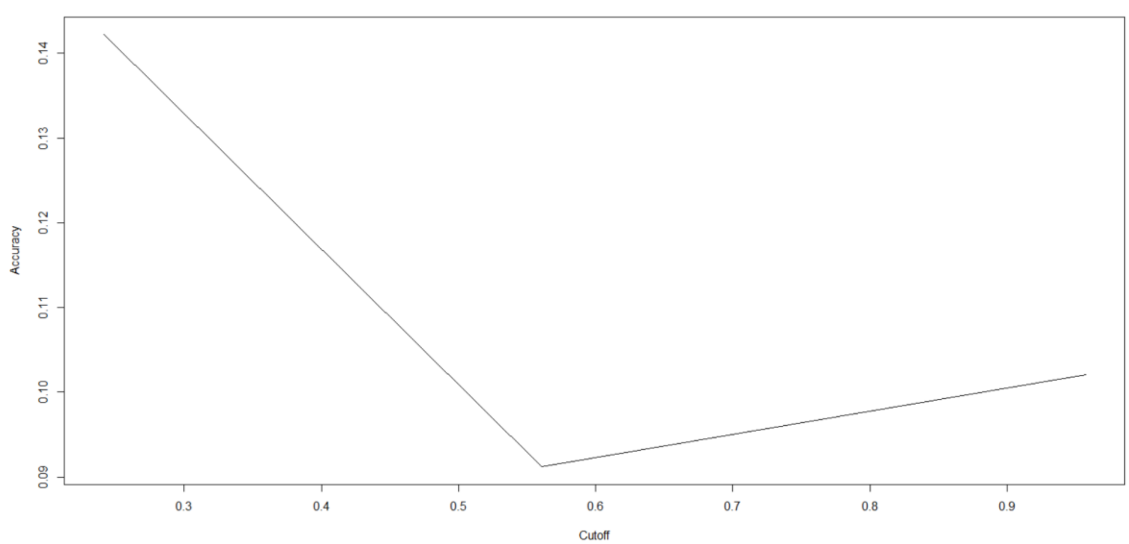
Lastly, Naïve Bayes

```
pred.nb <- predict(models$nb,newdata = testing,type = "prob")

pred.nb <- prediction(as.numeric( pred.nb[,2]), testing$Revenue)

acc.nb <- performance(pred.nb,"acc")

plot(acc.nb,main = "Naïve Bayes")
```



C) Accuracy for the enhanced:

```
pre.rf <- predict(stack.rf,newdata = testing,type = "prob")
pre <- prediction(as.numeric( pre.rf), testing$Revenue)
acc.rf <- performance(pre,"acc")
plot(acc.rf)
```



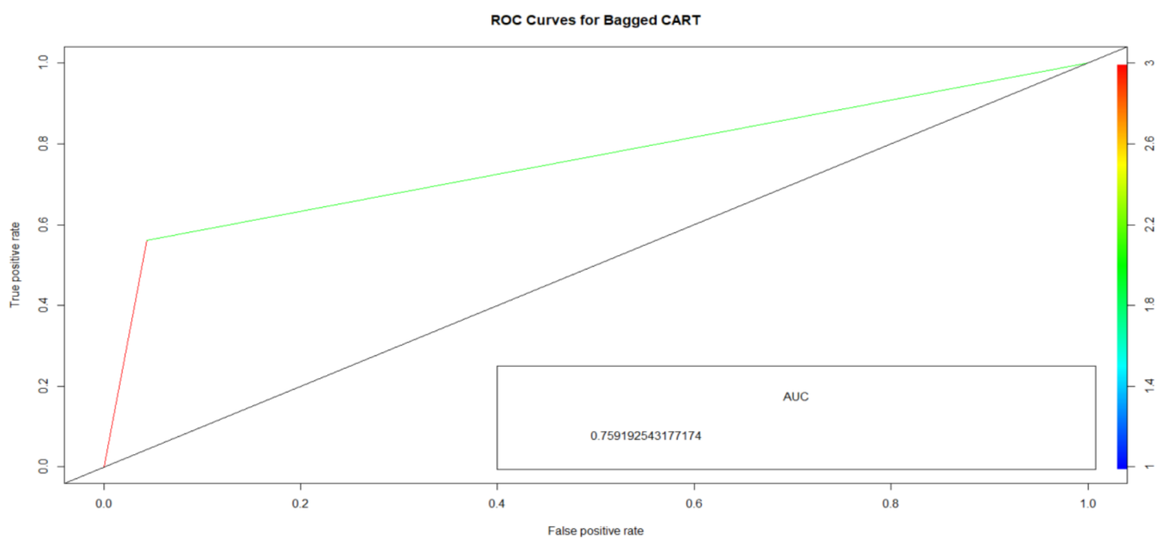
4. ROC and AUC

A) Accuracy for Bagging Algorithm

Bagging Algorithm contains the Bagged CART model and Random Forest Model

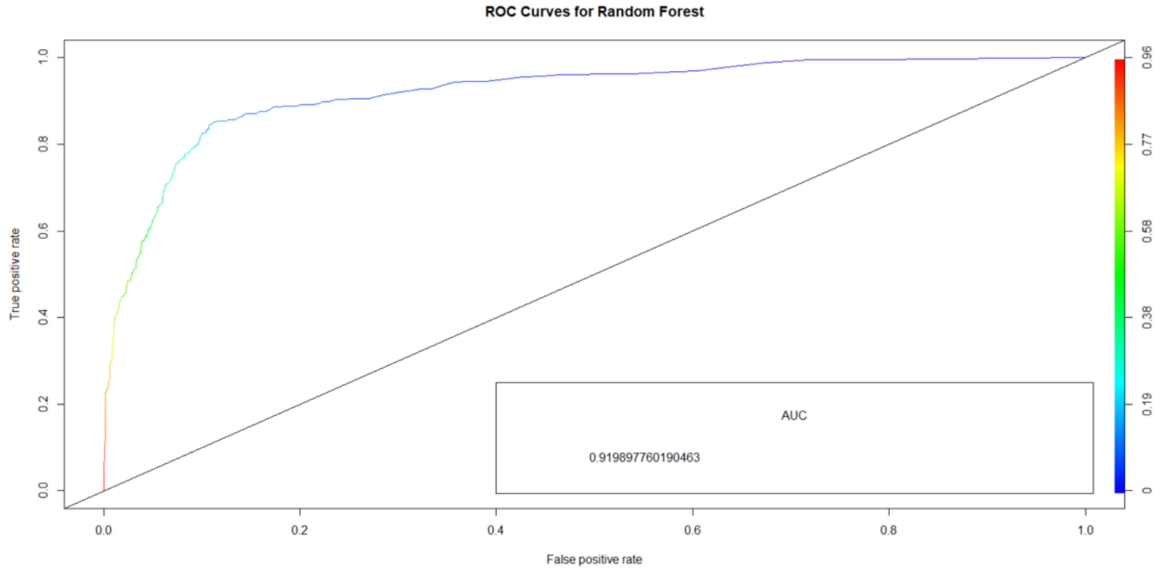
The code below is for Bagged CART

```
pre.tree <- predict(modle.treebag,newdata = testing,type = "prob")
prediction.tree <- prediction(pre.tree[,2], testing$Revenue)
pre.roc <- performance(prediction.tree,"tpr","fpr")
plot(pre.roc,colorize=T,main = "ROC Curves for Bagged CART")
abline(a= 0,b=1)

pre.auc <- performance(prediction.tree,measure="auc")

auc <- slot(pre.auc,"y.values")[[1]]

legend(.4,.25,auc,title = "AUC")
```



Next is to calculate the ROC and AUC to the Random Forest model

```
pred.rf <- predict(model.rf,newdata = testing,type = "prob")

pred.rf <- prediction(as.numeric( pred.rf[,2]), testing$Revenue)

pre.rf <- performance(pred.rf,"tpr","fpr")

plot(pre.rf,colorize=T,main = "ROC Curves for Random Forest")

abline(a= 0,b=1)

pre1.auc <- performance(pred.rf,measure="auc")

auc <- slot(pre1.auc,"y.values")[[1]]

legend(.4,.25,auc,title = "AUC")
```

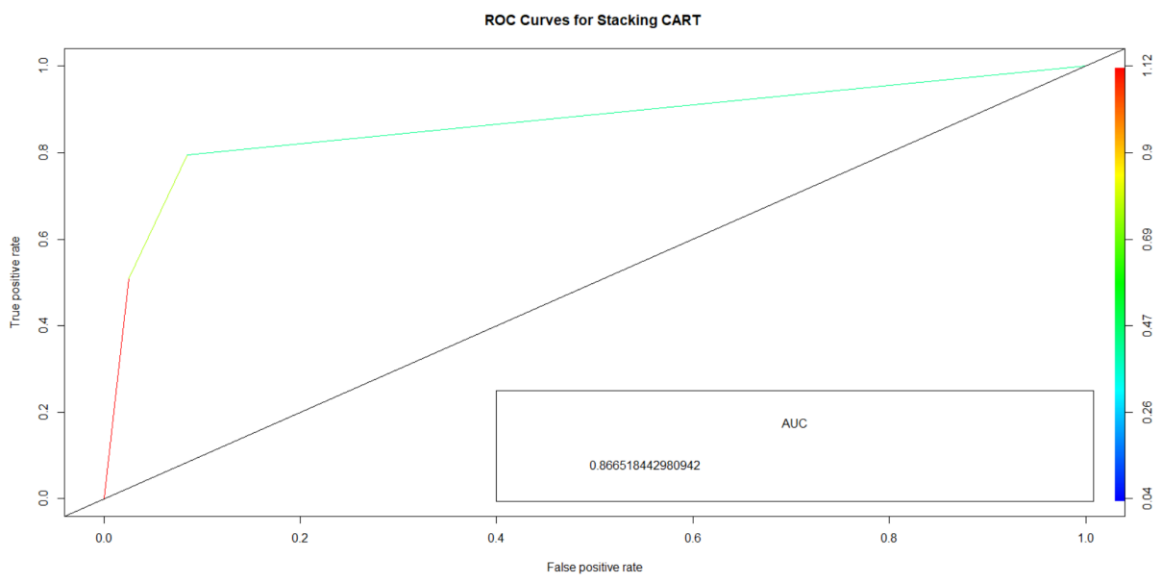


B) Accuracy for Stacking algorithm

Stacking Algorithm contains the KNN, Naïve Bayes and Stacking CART

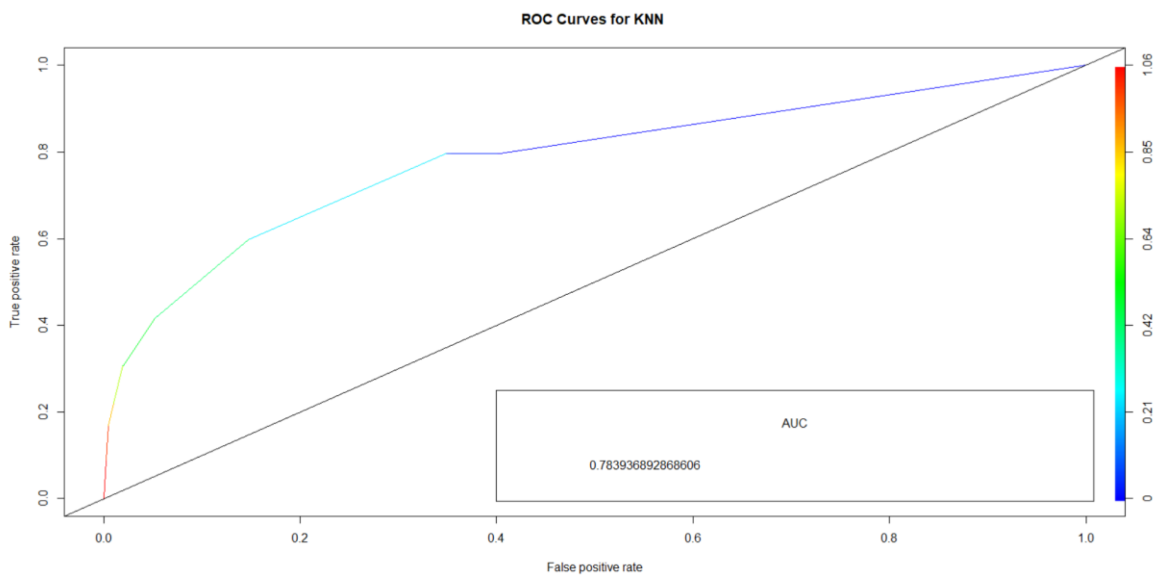
Firstly, Stacking Cart model

```
pred.rpart <- predict(models$rpart,newdata = testing,type = "prob")
pred.rpart <- prediction(as.numeric( pred.rpart[,2]), testing$Revenue)
pre.rpart <- performance(pred.rpart,"tpr","fpr")
plot(pre.rpart,colorize=T,main = "ROC Curves for Random Forest")
abline(a= 0,b=1)
pre3.auc <- performance(pred.rpart,measure="auc")
auc <- slot(pre3.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
```



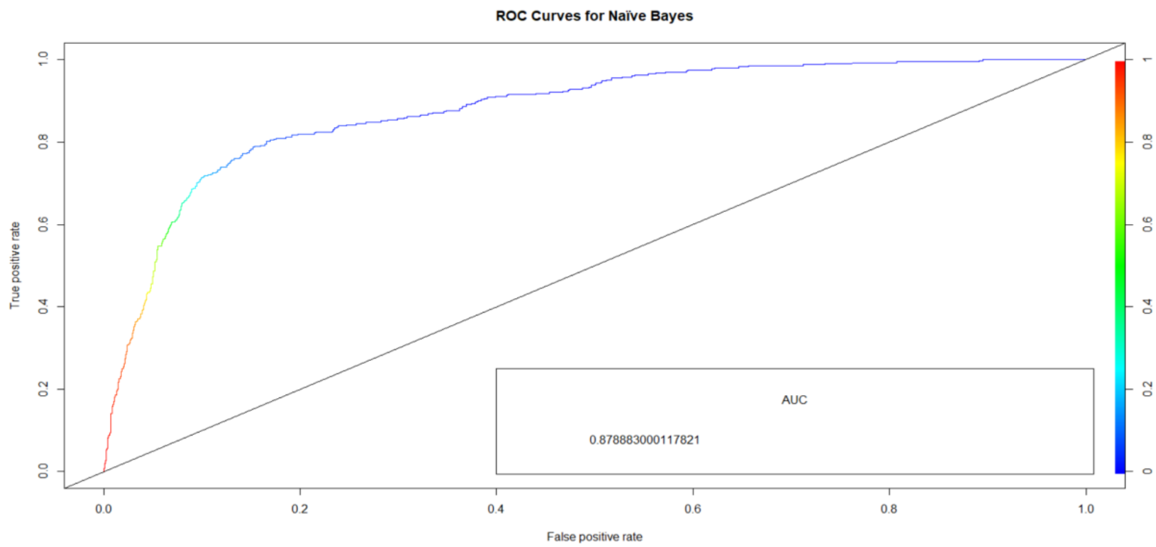
Secondly, Knn model

```
pred.knn <- predict(models$knn,newdata = testing,type = "prob")
pred.knn <- prediction(as.numeric( pred.knn[,2]), testing$Revenue)
pre.knn <- performance(pred.knn,"tpr","fpr")
plot(pre.knn,colorize=T,main = "ROC Curves for KNN ")
abline(a= 0,b=1)
pre4.auc <- performance(pred.knn,measure="auc")
auc <- slot(pre4.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
```



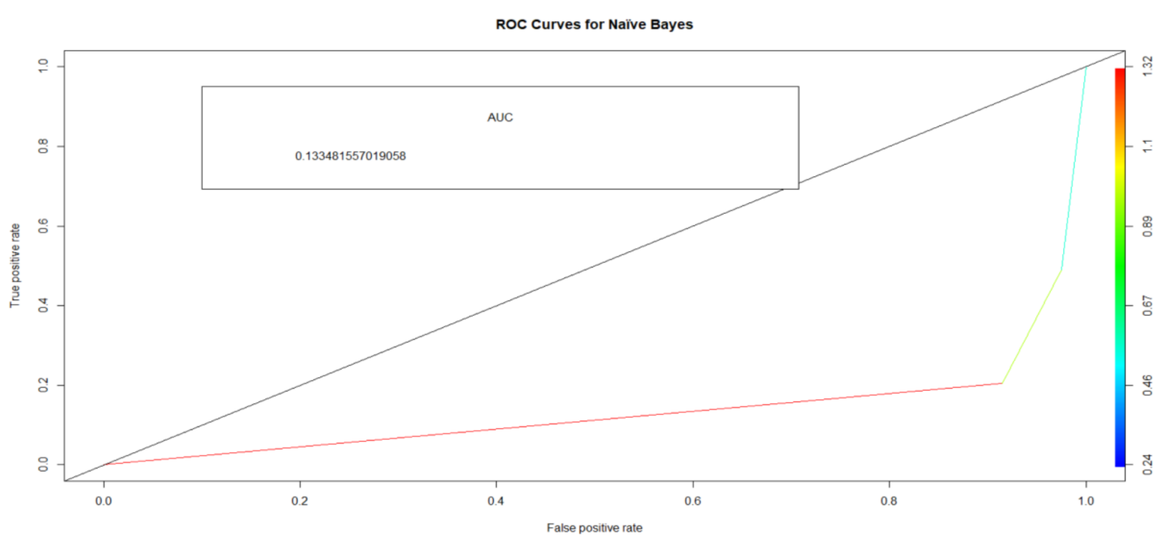
Lastly, Naïve Bayes model

```
pred.nb <- predict(models$nb,newdata = testing,type = "prob")
pred.nb <- prediction(as.numeric( pred.nb[,2]), testing$Revenue)
pre.nb <- performance(pred.nb,"tpr","fpr")
plot(pre.nb,colorize=T,main = "ROC Curves for Naïve Bayes")
abline(a= 0,b=1)
pre5.auc <- performance(pred.nb,measure="auc")
auc <- slot(pre5.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
```



C) ROC and AUC for the enhanced mdoel:

```
pre.rf <- predict(stack.rf,newdata = testing,type = "prob")

pre <- prediction(as.numeric( pre.rf), testing$Revenue)

r.pre <- performance(pre,"tpr","fpr")

plot(r.pre,colorize=T,main = "ROC Curves for Naïve Bayes")

abline(a= 0,b=1)

pre6.auc <- performance(pre,measure="auc")


auc <- slot(pre6.auc,"y.values")[[1]]

legend(.1,.95,auc,title = "AUC")
```



5. Training Time and testing time

First Training time

Bagging Algorithm

```
system.time(modle.treebag <- train(Revenue~., data=training, method="treebag", metric=metric,
                 trControl=bagging.control,preProc=c("center","scale")))
```

```
user  system elapsed
37.71    0.03   38.14
```

Random Forest

```
system.time(model.rf <- train(Revenue~., data=training, method="rf",
metric=metric,trControl=bagging.control,preProc=c("center","scale")))
```

```
user  system elapsed
401.98    5.43  411.16
```

Stacking(KNN, Naïve Bayes and Stacking CART)

```
System.time(models <- caretList(Revenue~., data=training, trControl=stack.control,
methodList=algorithmList))
```

```
user  system elapsed
85.75    0.06   86.52
```

Enchanced Model

```
system.time(stack.rf <- caretStack(models, method="rpart", metric="Accuracy",
trControl=stackControl))
```

```
user  system elapsed
3.54    0.02    3.59
```

Second Testing time

Bagging Algorithm

```
system.time(treeteest <- predict(modle.treebag,newdata = testing))
```

```
user  system elapsed
 0.17    0.00    0.17
```

Random Forest

```
system.time(baggingtest <- predict(model.rf,newdata = testing))
```

```
user  system elapsed
 0.14    0.00    0.14
```

Stacking CART

```
system.time(model.rpart <- predict(models$rpart,newdata = testing))
```

```
 user  system elapsed 
 0.02    0.00    0.01 
```

KNN

```
system.time(model.knn <- predict(models$knn,newdata = testing))
```

```
user  system elapsed 
0.27    0.00    0.27 
```

Naïve Bayes

```
system.time(model.nb <- predict(models$nb,newdata = testing))
```

```
 user  system elapsed 
 2.65    0.00    2.67 
```

Enchanced Model

```
system.time(models.rf <- predict(stack.rf,newdata = testing))
```

```
 user  system elapsed 
 2.86    0.00    2.89 
```

Appendix:

```
library(caret)
library(klaR)

shop <- read.csv("C:/Users/GTS/Downloads/online_shoppers_intention.csv")

#list types for each attribute
sapply(shop, class)
head(shop)


shop[11:15] <- NULL
shop[12] <- NULL
anyNA(shop)
#pre-processing
shop[,12] <- as.factor(shop[,12])
shop$VisitorType<-as.integer(shop$VisitorType)
#shop$Administrative <- as.numeric(shop$Administrative)
#shop$Informational <- as.numeric(shop$Informational)
#shop$ProductRelated <- as.numeric(shop$ProductRelated)
summary(shop)



boxplot(shop)
outliers <- boxplot(shop$ProductRelated_Duration, plot=FALSE)$out
shop[which(shop$ProductRelated_Duration %in% outliers),]
shop<- shop[-which(shop$ProductRelated_Duration %in% outliers),]
```

```
#Summarize class distribution
percentage <- prop.table(table(shop$Revenue)) * 100
cbind(freq=table(shop$Revenue), percentage=percentage)



#split
set.seed(3033)
intrain <- createDataPartition(y = shop$Revenue, p= 0.75, list = FALSE)
training <- shop[intrain,]
testing <- shop[-intrain,]

percentagetrain <- prop.table(table(training$Revenue)) * 100
cbind(freq=table(training$Revenue), percentage=percentagetrain)

percentagetest <- prop.table(table(testing$Revenue)) * 100
cbind(freq=table(testing$Revenue), percentage=percentagetest)



#Bagging
library(caret)
bagging.control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 7
metric <- "Accuracy"
# Bagged CART
```

```
set.seed(seed)

system.time(modle.treebag <- train(Revenue~., data=training, method="treebag",
metric=metric,

                    trControl=bagging.control,preProc=c("center","scale")))

print(modle.treebag)


system.time(treeteest <- predict(modle.treebag,newdata = testing))

confusionMatrix(data = treeteest,testing$Revenue)


pre.tree <- predict(modle.treebag,newdata = testing,type = "prob")

prediction.tree <- prediction(pre.tree[,2], testing$Revenue)

#pre vs recall

perform.tree <- performance(prediction.tree,"prec","rec")

plot(perform.tree,colorize = T,main = "Bagged CART")

#accuracy

acc.trr <- performance(prediction.tree,"acc")

plot(acc.trr,main = "Bagged CART")

#roc

pre.roc <- performance(prediction.tree,"tpr","fpr")

plot(pre.roc,colorize=T,main = "ROC Curves for Bagged CART")

abline(a= 0,b=1)


pre.auc <- performance(prediction.tree,measure="auc")


auc <- slot(pre.auc,"y.values")[[1]]

legend(.4,.25,auc,title = "AUC")

auc
```

```
set.seed(seed)
system.time(model.rf <- train(Revenue~., data=training, method="rf", metric=metric,
                  trControl=bagging.control,preProc=c("center","scale")))
print(model.rf)
system.time(baggingtest <- predict(model.rf,newdata = testing))
confusionMatrix(data = baggingtest,testing$Revenue)

pred.rf <- predict(model.rf,newdata = testing,type = "prob")
pred.rf <- prediction(as.numeric( pred.rf[,2]), testing$Revenue)
perform.rf <- performance(pred.rf,"prec","rec")
plot(perform.rf,colorize = T,main = "Random Forest")
#acc
acc.rf <- performance(pred.rf,"acc")
plot(acc.rf,main = "Random Forest")
#roc
pre.rf <- performance(pred.rf,"tpr","fpr")
plot(pre.rf,colorize=T,main = "ROC Curves for Random Forest")
abline(a= 0,b=1)
pre1.auc <- performance(pred.rf,measure="auc")

auc <- slot(pre1.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
auc



# combine the models
bagging_results <- resamples(list(treebag=modle.treebag, rf=model.rf))
summary(bagging_results)
dotplot(bagging_results)
bwplot(bagging_results)
```

```r
# Example of Stacking algorithms
# create submodels
library(caretEnsemble)
stack.control <- trainControl(method="repeatedcv", number=10, repeats=3,
                        savePredictions=TRUE, classProbs=TRUE,preProc=c("center","scale"))
algorithmList <- c( 'rpart', 'knn', 'nb')
set.seed(seed)
system.time (models <- caretList(Revenue~., data=training, trControl=stack.control,
methodList=algorithmList))
results <- resamples(models)
summary(results)
dotplot(results)
bwplot(results)

#coolelation
modelCor(results)
splom(results)


system.time(model.rpart <- predict(models$rpart,newdata = testing))
confusionMatrix(data = model.rpart,testing$Revenue)

pred.rpart <- predict(models$rpart,newdata = testing,type = "prob")
pred.rpart <- prediction(as.numeric( pred.rpart[,2]), testing$Revenue)
perform.rpart <- performance(pred.rpart,"prec","rec")
```

```
plot(perform.rpart,colorize = T,main = "Stacking CART")

acc.rpart <- performance(pred.rpart,"acc")
plot(acc.rpart,main = "Stacking CART")

#roc
pre.rpart <- performance(pred.rpart,"tpr","fpr")
plot(pre.rpart,colorize=T,main = "ROC Curves for Stacking CART ")
abline(a= 0,b=1)
pre3.auc <- performance(pred.rpart,measure="auc")

auc <- slot(pre3.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
auc



system.time(model.knn <- predict(models$knn,newdata = testing))
confusionMatrix(data = model.knn,testing$Revenue)

pred.knn <- predict(models$knn,newdata = testing,type = "prob")
pred.knn <- prediction(as.numeric( pred.knn[,2]), testing$Revenue)
perform.knn <- performance(pred.knn,"prec","rec")
plot(perform.knn,colorize = T,main = "KNN")

acc.knn <- performance(pred.knn,"acc")
plot(acc.knn,main = "KNN")
#roc
pre.knn <- performance(pred.knn,"tpr","fpr")
plot(pre.knn,colorize=T,main = "ROC Curves for KNN ")
```

```
abline(a= 0,b=1)
pre4.auc <- performance(pred.knn,measure="auc")

auc <- slot(pre4.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
auc



system.time(model.nb <- predict(models$nb,newdata = testing))
confusionMatrix(data = model.nb,testing$Revenue)

pred.nb <- predict(models$nb,newdata = testing,type = "prob")
pred.nb <- prediction(as.numeric( pred.nb[,2]), testing$Revenue)
perform.nb <- performance(pred.nb,"prec","rec")
plot(perform.rf,colorize = T,main = "Naïve Bayes ")
acc.nb <- performance(pred.nb,"acc")
plot(acc.nb,main = "Naïve Bayes")
#roc
pre.nb <- performance(pred.nb,"tpr","fpr")
plot(pre.nb,colorize=T,main = "ROC Curves for Naïve Bayes")
abline(a= 0,b=1)
pre5.auc <- performance(pred.nb,measure="auc")

auc <- slot(pre5.auc,"y.values")[[1]]
legend(.4,.25,auc,title = "AUC")
auc

library(caretEnsemble)
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3,
savePredictions=TRUE, classProbs=TRUE)
```

```
set.seed(seed)

stack.nb <- caretStack(models, method="knn", metric="Accuracy",
trControl=stackControl)

print(stack.nb)


set.seed(seed)

system.time(stack.rf <- caretStack(models, method="rpart", metric="Accuracy",
trControl=stackControl))

print(stack.rf)


system.time(models.rf <- predict(stack.rf,newdata = testing))

confusionMatrix(data = models.rf,testing$Revenue)


pre.rf <- predict(stack.rf,newdata = testing,type = "prob")

pre <- prediction(as.numeric( pre.rf), testing$Revenue)

pre2 <- performance(pre,"prec","rec")

plot(pre2,colorize = T)

acc.rf <- performance(pre,"acc")

plot(acc.rf)

#roc

r.pre <- performance(pre,"tpr","fpr")

plot(r.pre,colorize=T,main = "ROC Curves for Naïve Bayes")

abline(a= 0,b=1)

pre6.auc <- performance(pre,measure="auc")


auc <- slot(pre6.auc,"y.values")[[1]]

legend(.1,.95,auc,title = "AUC")

auc
```

```
library(ROCR)
#sol 1
lda.model <- predict(stack.rf,newdata = testing,type= "raw" )
head(lda.model)
lda.pre <- prediction(as.numeric(lda.model),as.numeric( testing$Revenue))
evl1 <- performance(lda.pre,"acc")
plot(evl1)

evl2 <- performance(pre,"tpr","fpr")
plot(evl2)

evl3 <- performance(pre,"sens","spec")
plot(evl3)


max <- which.max(slot(evl1,"y.values")[[1]])
max
acc <- slot(evl1,"y.values")[[1]][max]
cut <- slot(evl1,"x.values")[[1]][max]
print(c(Accuuracy=acc,cutoff=cut))

roc <- performance(lda.pre,"tpr","fpr")
plot(roc,colorize = T)

#sol2
library(ROCR)
pre.rf <- predict(stack.rf,newdata = testing,type = "prob")
pre <- prediction(as.numeric( pre.rf), testing$Revenue)
pre1 <- performance(pre,"acc")
```

```
plot(pre1)
abline(h=0.89)
#pre vs recall
pre2 <- performance(pre,"prec","rec")
plot(pre2,colorize = T)

#AUC
pre3 <- performance(pre,"tpr","fpr")
plot(pre3)
plot(pre3,colorize=T,main = "ROC Curves",
    ylab = "sensivity",
    xlab = "specifity")
abline(a= 0,b=1)

pre4 <- performance(pre,measure="auc")
pre4

auc <- pre4@y.values[[1]]
legend(.5,.25,auc,title = "AUC")
auc
acc1 <- slot(pre4,"y.values")[[1]]
acc1
```