LEEDS BECKETT UNIVERSITY

School of Computing, Creative Technology and Engineering

Student ID	c7466889
Student Name	Ojaswi Shrestha
Module Name & CRN	Applied Machine Learning 18909
Level	L5
Assessment Name & Part No.	Coursework Part 2
Project Title	Bank Marketing Data Analysis
Date of Submission	2025/05/09
Course	Applied Machine Learning
Academic Year	2025

Table of Contents

Part II

1. Introduction

In the coursework part I, originally, there were 45211 rows with 17 columns of data in bank dataset, with the removal of missing values there are 45058 rows of data with same 17 columns now in clean_bank_data.

```
> dim(bank)
[1] 45211    17
> dim(clean_bank_data)
[1] 45058    17
```

Figure 1: Dimension of dataset before and after cleansing.

The clean_bank_data has variables such as y (target variable) means whether the term deposit was subscribed or not, pdays represents last contact to the customer, previous is the number of contacts before campaign, poutcome is the result of previous campaign, and many other variables as well.

```
> colnames(clean_bank_data)
 [1] "age"       "job"       "marital"   "education" "default"   "balance"   "housing"
 [8] "loan"      "contact"   "day"       "month"     "duration"  "campaign"  "pdays"
[15] "previous"  "poutcome"  "y"
```

Figure 2: Column names of pre-processed dataset

With the use of bank marketing dataset, data cleansing process such as finding and handling missing values, detecting outliers, imputing them through median and capping method, scaling the data, reducing the columns as in pca were done from which various interpretations and hypothesis from that particular dataset were taken.

Previously, data partition was done from pca data which now have been partitioned from clean_bank_data into training and test set in 80 to 20 ratio with a particular seed of 1.

```
> library(caret)
> trainIndex<-createDataPartition(clean_bank_data$y,p=0.8,list=FALSE)
> train_data<-clean_bank_data[trainIndex,]
> test_data<-clean_bank_data[-trainIndex,]
> train_data$y<-as.factor(train_data$y)
> test_data$y<-as.factor(test_data$y)
```

Figure 3: Data partition into 80:20 for train and test with caret package

2. Modelling

2.1. Logistic regression

Logistic regression a supervised learning model for binary classification. It calculates probability of class label based on linear combination of input variables (Hosmer et al., 2013). It is easy to implement, interpret and train. It is a supervised machine learning also known as discriminative model for distinguishing between classes or categories (Bishop, 2006). This model is used for binary classification problem which applies to the target variable 'y' in the dataset with 'yes' or 'no' outcome for subscription to the term deposit. It helps to predict the probability of the result on the basis of other variables in dataset.

Logistic regression model.

Data partitioning:

```
> library(caret)
> trainIndex<-createDataPartition(clean_bank_data$y,p=0.8,list=FALSE)
> train_data<-clean_bank_data[trainIndex,]
> test_data<-clean_bank_data[-trainIndex,]
```

Figure 4: Data partition with caret package

The figure above shows the usage of caret library package for splitting data. First, it takes target variable (y) of the pre-processed dataset then creates balanced split of data into train data as 80% and test data as 20% maintaining the proportion of 'yes', 'no' values in both train, test data.

Target variable as categorical for classification:

```
> train_data$y<-as.factor(train_data$y)
> test_data$y<-as.factor(test_data$y)
```

Figure 5: y variable as factor

The target y variable is converted to factor and placed in train and test data in order to make classification possible in logistic regression.

Proportion of categorical value in train-test data:

```
> prop.table(table(train_data$y)) * 100

      no      yes
88.35687 11.64313
> prop.table(table(test_data$y)) * 100

      no      yes
88.35867 11.64133
```

Figure 6: Yes, No proportion in train test data

Table() function used in y variable counts the amount of yes, no outcomes in both train test data.

Prop.table converts the amount into proportion and finally *100 turn it into percentage. It is done to see the distribution of yes-no outcome in both train and test data.

Model training:

```
> caret_glm_mod=train(
+      form=y ~ .,
+      data=train_data,
+      trControl=trainControl(method="cv",number=5),
+      method="glm",
+      family="binomial"
+ )
```

Figure 7: Training the logistic regression model with caret package

The figure above trains the model where y is predicted from all other remaining columns, with use of train data through 5-fold cross validation, using binary classification in generalized linear model.

Class prediction:

```
> predicted_test<-predict(caret_glm_mod, newdata = test_data)
```

Figure 8: Predict class probabilities of test_data

The above code line uses trained logistic regression model (caret_glm_mod) to predict class labels (yes, no) of the test data in vector form based on probability being greater than 0.5 for 'yes' or else 'no'.

```
> predicted_test
   [1] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
  [30] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
  [59] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
  [88] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
 [117] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
 [146] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
 [175] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
 [958] no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no no
 [987] no no no no no no no no no no no no no no
 [ reached getOption("max.print") -- omitted 8011 entries ]
Levels: no yes
```

Figure 9: Probability output of yes-no of test_data

Confusion Matrix for accuracy:

```
> confusionMatrix(as.factor(predicted_test),test_data$y,positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7755  681
       yes  207  368

               Accuracy : 0.9015
                 95% CI : (0.8951, 0.9075)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 3.469e-08

                  Kappa : 0.4041

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

            Sensitivity : 0.35081
            Specificity : 0.97400
         Pos Pred Value : 0.64000
         Neg Pred Value : 0.91927
             Prevalence : 0.11641
         Detection Rate : 0.04084
   Detection Prevalence : 0.06381
      Balanced Accuracy : 0.66241

       'Positive' Class : yes
```

Figure 10: Confusion matrix for comparison of predicted and actual class labels

The above code line for confusion matrix takes on vector of predicted_test, converting into factor for confusion matrix, then comparing with actual labels of test data with ‘yes’ being the positive class.

The outcome shows:

True Negative-7755

False Negative-681

False Positive-207

True Positive-368

with accuracy of 90.15% but with sensitivity at only 35.08% meaning with our imbalanced dataset it is struggling to detect “yes” cases which is vital for marketing study. It is only predicting 35 yes cases out of all 100 yes cases so to manage the sensitivity, threshold tuning will be done.

Predicted probabilities of 'yes' class label:

```
> predicted_probs <- predict(caret_glm_mod, newdata = test_data, type = "prob")[, "yes"]
> predicted_probs
   [1] 0.0120006444 0.0060373786 0.0188367139
   [4] 0.0116068600 0.0066352443 0.0066379767
   [7] 0.2126510234 0.1230381679 0.0177706972
  [10] 0.0458480126 0.2576142871 0.1631801152
  [13] 0.1053004524 0.0484520098 0.0028954129
  [16] 0.0100435303 0.0363675668 0.0092691112
  [19] 0.0026863033 0.0649159975 0.0022944126
  [22] 0.0105886983 0.1229861089 0.0018053464
 [991] 0.0237949382 0.0039914474 0.0107248022
 [994] 0.0041572338 0.0150247765 0.0318033939
 [997] 0.1572297362 0.0192767841 0.1330744524
[1000] 0.0129615450
 [ reached getOption("max.print") -- omitted 8011 entries ]
```

Figure 11: Predicted Probabilities from LR model with 'yes' class of test_data.

The above given code takes out predicted probability from logistic regression model (caret_glm_model) for "yes" class of test_data where the outcome is in vector form with each representing probability of "yes" class only.

Applying threshold of 0.3 for yes class in predicted_classes

```
> predicted_classes <- ifelse(predicted_probs > 0.3, "yes", "no")
> predicted_classes
   [1] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
  [16] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
  [31] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
  [46] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
  [61] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
  [76] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
  [91] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
 [106] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
```

Figure 12: Tuning for predicted_classes

The predicted_classes puts on condition where threshold of predicted_probs is kept at 0.3 for "yes" values in order to increase the model's sensitivity.

Confusion matrix with 0.3 threshold

```
> confusionMatrix(as.factor(predicted_classes), test_data$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7394  434
       yes  568  615

               Accuracy : 0.8888
                 95% CI : (0.8821, 0.8952)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 0.06262

                  Kappa : 0.4879

 Mcnemar's Test P-Value : 2.65e-05

            Sensitivity : 0.58627
            Specificity : 0.92866
         Pos Pred Value : 0.51986
         Neg Pred Value : 0.94456
             Prevalence : 0.11641
         Detection Rate : 0.06825
   Detection Prevalence : 0.13128
      Balanced Accuracy : 0.75747

       'Positive' Class : yes
```

Figure 13: Confusion matrix after threshold tuning

The above code line for confusion matrix takes on predicted_classes, converting into factor for confusion matrix, then comparing with actual labels of test data with ‘yes’ being the positive class.

The outcome shows:

True Negative-7394

False Negative-434

False Positive-568

True Positive-615

with accuracy of 88.88% and with sensitivity at 0.58 meaning it got better sensitivity than before tuning and is prediction more “yes” cases better than previous one.

ROC:

```
> library(pROC)
> roc_curve <- roc(test_data$y, predicted_probs)
Setting levels: control = no, case = yes
Setting direction: controls < cases
> roc_curve

Call:
roc.default(response = test_data$y, predictor = predicted_probs)

Data: predicted_probs in 7962 controls (test_data$y no) < 1049 cases (test_data$y yes).
Area under the curve: 0.9089
```

```
> auc_value <- auc(roc_curve)
> auc_value
Area under the curve: 0.9089
```

Figure 14: ROC and AUC of logistic regression.

ROC Curve:

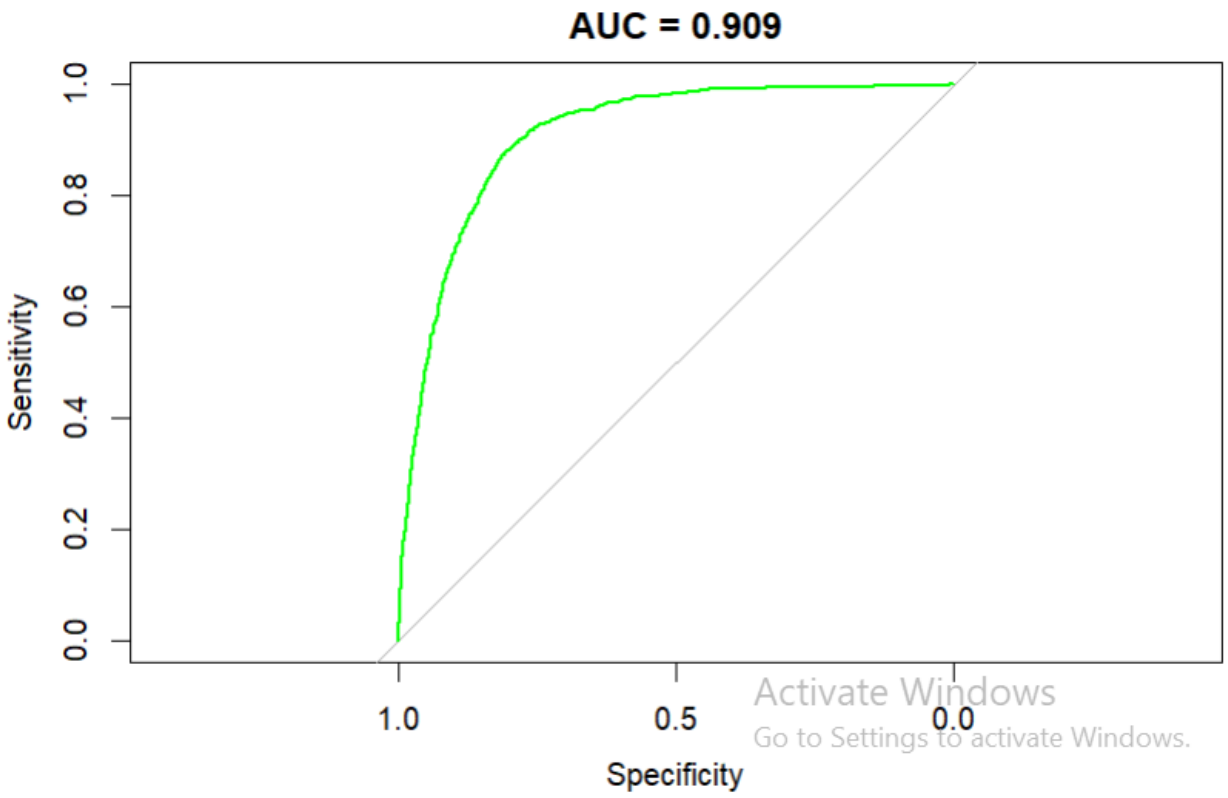

Figure 15: ROC curve of logistic regression

The above figure shows ROC curve of logistic regression model with Sensitivity on Y axis also called True Positive Rate which represents identification of actual positive cases then Specificity on X axis also called False Positive Rate which represents identification of negative label as positive. The above green curve bends at top left corner meaning the model is good at distinguishing classes. The AUC value is 0.909 meaning the model ranks true positive higher than false positive.

Model tuning

```
> library(glmnet)
Loading required package: Matrix
Loaded glmnet 4.1-8
>
> # Convert data to matrix form (glmnet requires this)
> x <- model.matrix(y ~ ., data = train_data)[,-1]
> y <- train_data$y
```

Figure 16: library for model tuning

Glmnet library is used in order to have hyperparameters to tune the logistic regression model. It provides hyperparameters tuning of lambda and alpha. As the bank marketing dataset has many predictors, some of which are correlated so the glmnet library is required. As glmnet requires numeric matrix of predictors the x variable converts whole train_data into dummy numeric matrix while removing the intercept.

```
> set.seed(123)
> tuned_model <- train(
+     x = x,
+     y = y,
+     method = "glmnet",
+     trControl = trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction = twoC
lassSummary),
+     metric = "ROC",
+     tuneLength = 10
+ )
```

Figure 17: Training model with glmnet for tuning

It uses a fix seed of 123 for same split and results every time it is run. The train method uses glmnet method with ‘x’ predictor matrix and ‘y’ as target vector. It computes the probability and evaluates model based on ROC, sensitivity and specificity also uses 10 various values of glmnet hyperparameters.

```
> x_test <- model.matrix(y ~ ., data = test_data)[,-1]
> predicted_probs <- predict(tuned_model, newdata = x_test, type = "prob")[, "yes"]
> predicted_classes <- ifelse(predicted_probs > 0.3, "yes", "no")
> confusionMatrix(as.factor(predicted_classes), test_data$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7385  431
       yes  577  618

               Accuracy : 0.8881
                 95% CI : (0.8814, 0.8946)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 0.09117

                  Kappa : 0.4872

 Mcnemar's Test P-Value : 4.946e-06

            Sensitivity : 0.58913
            Specificity : 0.92753
         Pos Pred Value : 0.51715
         Neg Pred Value : 0.94486
             Prevalence : 0.11641
         Detection Rate : 0.06858
   Detection Prevalence : 0.13262
      Balanced Accuracy : 0.75833

       'Positive' Class : yes
```

Figure 18: Confusion matrix of tuned model

Similarly, x_text is numeric matrix of test_data, with predicted_probs from new tuned model then in predicted_classes, threshold greater than 0.3 being 'yes' or else 'no'. Confusion matrix outcome shows:

True Negative-7385

False Negative-431

False Positive-577

True Positive-618

with accuracy of 88.81% and sensitivity at 0.58. With this we can conclude that the model is really good at predicting the customers who won't subscribe to the campaign as specificity is 0.92. As for predicting subscribing customers, it's still lagging behind with only 0.58 sensitivity of which 51% is only correct most of the time. Overall, the model is great at discarding non-subscribing customers.

```
> roc_curve <- roc(test_data$y, predicted_probs)
Setting levels: control = no, case = yes
Setting direction: controls < cases
> roc_curve

Call:
roc.default(response = test_data$y, predictor = predicted_probs)

Data: predicted_probs in 7962 controls (test_data$y no) < 1049 cases (test_data$y yes).
Area under the curve: 0.9064
> auc_value <- auc(roc_curve)
> auc_value
Area under the curve: 0.9064
> plot(roc_curve, col = "blue", main = paste("AUC =", round(auc_value, 3)))
```

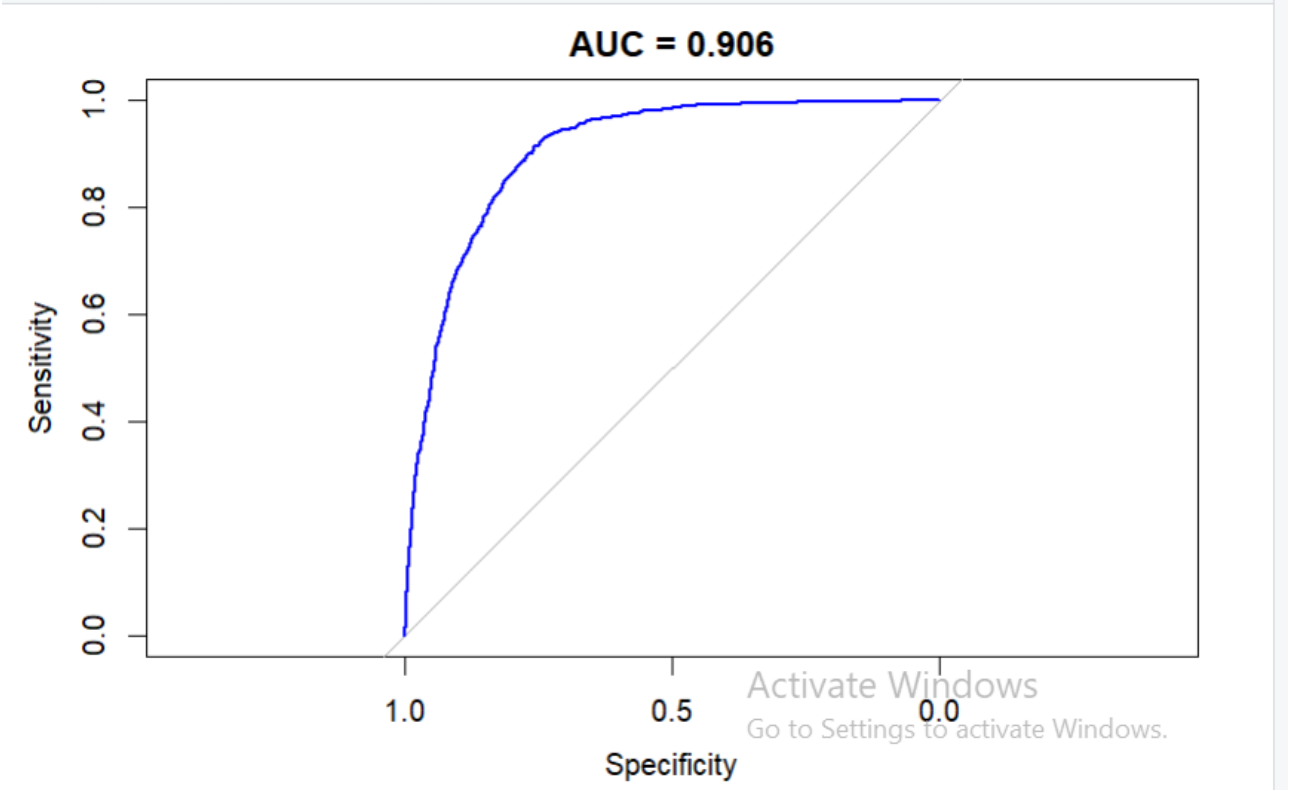

Figure 19: ROC and AUC values with plot

The plot shows AUC of 0.906 which means it can distinguish between subscribers and non-subscribers. The ROC is steep at beginning but later at top left corner it bends meaning the model has high sensitivity and specificity at various points of threshold. The model will make lesser false alarms and will help to target customers who are likely to subscribe.

2.2 K-nearest neighbour

K-Nearest Neighbours is a non-parametric technique which groups instances on the basis of majority class of k nearest training examples in the feature space (Cover & Hart, 1967). It requires no training phase and adjusts to several data types which is also why it's called "LAZY" learner.

```
library(caret)
trainIndex<-createDataPartition(clean_bank_data$y,p=0.8,list=FALSE)
train_data<-clean_bank_data[trainIndex,]
test_data<-clean_bank_data[-trainIndex,]
test_label<-as.factor(clean_bank_data$y[-trainIndex])
```

Figure 20: Data partition for train-test data

The above figure shows data partition done on clean_bank_data as 80% for train data and 20% for test data, also the test_label is a factor of y variable of clean_bank_data with 20% partition.

```
ctrl <- trainControl(method = "cv", number = 5)
knnFit <- train(y ~ ., data = train_data, method = "knn", trControl = ctrl)
```

Figure 21: KNN train method

It trains the method through 5-fold cross validation from caret package where data splits into 5 parts with 4:1 ratio for train and test, iterating 5 times. Then, trains a KNN classifier using all predictor variables in order to predict for subscribing customers.

```
> knnFit
k-Nearest Neighbors

36047 samples
   16 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 28837, 28837, 28838, 28838, 28838
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.8766056  0.2155937
  7  0.8808223  0.2210786
  9  0.8812939  0.2099891

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

Figure 22: Knn Fit output

It shows that 36047 samples with 16 predictors were trained on in Knn model where number of k were 5,7 and 9 with accuracy above 85% but kappa at just above 0.20 meaning that model is biased towards majority class of no so, more biased to discover customers who are unlikely to subscribe to the campaign.

```
> knnPredict <- predict(knnFit,newdata = test_data )
> knnPredict
   [1] no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  yes no  no  no
  [23] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no
  [45] no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
  [67] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
  [89] no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [111] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no
 [133] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no  no
 [155] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [177] no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no
 [969] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [991] no  no  no  no  no  no  no  no  no  no
 [ reached getOption("max.print") -- omitted 8011 entries ]
Levels: no yes
```

Figure 23: Prediction of knnFit

It uses the trained knn model (knnFit) to make predictions on the test dataset where no being non-subscribers and yes for subscribers.

```
> confusionMatrix(knnPredict,test_label, positive="yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7762  875
       yes  200  174

               Accuracy : 0.8807
                 95% CI : (0.8738, 0.8873)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 0.8082

                  Kappa : 0.1953

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.16587
            Specificity : 0.97488
         Pos Pred Value : 0.46524
         Neg Pred Value : 0.89869
             Prevalence : 0.11641
         Detection Rate : 0.01931
   Detection Prevalence : 0.04150
      Balanced Accuracy : 0.57038

       'Positive' Class : yes
```

Figure 24: Confusion matrix of knn model

The confusion matrix shows accuracy of 89.8% with sensitivity of 0.22 for positive 'yes' class and 852 false negatives so, it still misses most of the potential subscribers with specificity of 0.97 meaning model predicts non-subscribers almost perfectly. Kappa of 0.19 indicates bad agreement between predicted and actual values so, it's being biased towards negative class and mostly predicting no in the model. The positive and negative prediction value are 0.46 and 0.89 respectively which means model predicts "yes" 46.5% correctly and predicts "no" 89.8% correctly all the time.

The outcome of confusion matrix:

True Negative-7735

False Negative-852

False Positive-227

True Positive-197

```
> knnPredict <- predict(knnFit,newdata = test_data , type="prob")
> knnPredict
           no       yes
1   0.8888889 0.1111111
2   1.0000000 0.0000000
3   1.0000000 0.0000000
4   1.0000000 0.0000000
5   1.0000000 0.0000000
6   1.0000000 0.0000000
7   1.0000000 0.0000000
8   1.0000000 0.0000000
9   1.0000000 0.0000000
10  1.0000000 0.0000000
11  1.0000000 0.0000000
12  0.6666667 0.3333333
498 1.0000000 0.0000000
499 0.8888889 0.1111111
500 1.0000000 0.0000000
 [ reached 'max' / getOption("max.print") -- omitted 8511 rows ]
```

Figure 25: Prediction probability of yes-no classes

It shows probability of each class of test data with knn trained model.

```
> library(pROC)
> res.roc <- roc(test_label, knnPredict[,1])
Setting levels: control = no, case = yes
Setting direction: controls > cases
> auc <-res.roc$auc
> auc
Area under the curve: 0.7635
```

Figure 26: ROC and AUC values of Knn model

It shows roc and auc values of y varible from test set through knn prediction while removing the first column.

```
> plot.roc(res.roc, print.auc = TRUE, col="red")
```

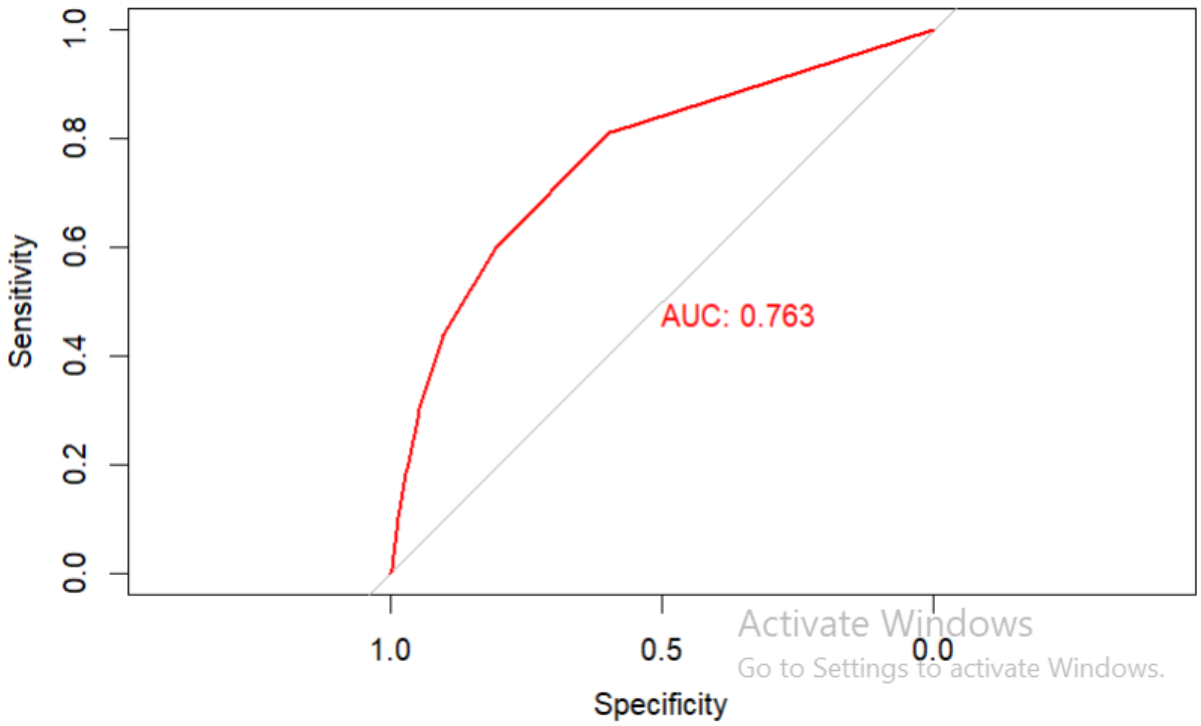

Figure 27: ROC and AUC curve of Knn model.

It shows the AUC value of 0.763 which means being good at distinguishing the subscribers or non-subscribers better than a random guessing. However, the ROC curve doesn't bend at most portion of top left corner meaning the curve is not that good with its prediction.

Model Tuning

```
tuneGrid <- expand.grid(k = seq(10, 20, by = 2))
ctrl <- trainControl(
    method = "cv",
    number = 5,
    classProbs = TRUE,
    summaryFunction = twoClassSummary
)

knnFit <- train(
    y ~ .,
    data = train_data,
    method = "knn",
    trControl = ctrl,
    tuneGrid = tuneGrid,
    metric = "ROC"
)
```

Figure 28: Tuning the Knn model

The above code does tuning for Knn model using cross validation, while optimizing for ROC and defines the value of k from 10 to 20 with difference of 2 which helps to find the best number of neighbors. Ctrl sets up how to train and evaluate the model where classProbs compute the probabilities, and twoClassSummary evaluates model through ROC, Sensitivity and Specificity. Lastly the knnFit trains the Knn model with multiple k values and choses best model with highest AUC score.

```
> knnFit
k-Nearest Neighbors

36047 samples
   16 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 28838, 28837, 28837, 28838, 28838
Resampling results across tuning parameters:

  k   ROC        Sens       Spec
  10  0.7664598  0.9729042  0.1817989
  12  0.7738862  0.9733438  0.1746535
  14  0.7821755  0.9758242  0.1706036
  16  0.7856923  0.9758556  0.1737006
  18  0.7893757  0.9756358  0.1736994
  20  0.7921305  0.9762323  0.1748927

ROC was used to select the optimal model using the largest value.
The final value used for the model was k = 20.
```

Figure 29: Output of knnFit

It shows number of k were 10,12,14,16,18 and 20 with ROC of above 0.76. sensitivity of 0.97 on the basis of “no” class and specificity of 0.17 meaning that model is biased towards majority class of no.

```
> knnPredict <- predict(knnFit,newdata = test_data )
> knnPredict
   [1] no  no  no  no  no  no  no  no  no  no  no  yes no  yes no  no  no  no  yes no  no  no
  [23] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no
  [45] no  no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
  [67] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
  [89] no  yes no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [111] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no  no
 [133] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [155] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [177] no  no  no  no  no  no  no  no  no  no  yes no  no  no  no  no  no  no  no  no  no  no
 [947] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  yes no  no  no
 [969] no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no  no
 [991] no  no  no  no  no  no  no  no  no  no
 [ reached getOption("max.print") -- omitted 8011 entries ]
Levels: no yes
```

Figure 30: Prediction probability of yes-no classes after tuning

It shows probability of each class of test data with knn trained model.

```
> confusionMatrix(knnPredict,test_label)
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7777  879
       yes  185  170

               Accuracy : 0.8819
                 95% CI : (0.8751, 0.8885)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 0.6958

                  Kappa : 0.1948

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9768
            Specificity : 0.1621
         Pos Pred Value : 0.8985
         Neg Pred Value : 0.4789
             Prevalence : 0.8836
         Detection Rate : 0.8631
   Detection Prevalence : 0.9606
      Balanced Accuracy : 0.5694

       'Positive' Class : no
```

Figure 31: Confusion matrix of knn model after tuning

The confusion matrix shows accuracy of 88.19% with sensitivity of 0.97 for ‘no’ class and 879 false negatives so, misses a lot of potential subscribers with specificity of 0.16 meaning model predicts subscribers really badly. Kappa of 0.19 indicates poor agreement between predicted and actual

values. The positive and negative prediction value are 0.89 and 0.47 respectively which means model predicts “no” 89.8% correctly and predicts “yes” 47.8% correctly all the time, as for the sensitivity and positive prediction value; its taking ‘no’ as positive class so its high in regard to other models.

The outcome of confusion matrix:

True Negative-7777

False Negative-879

False Positive-185

True Positive-170

```
> knnPredict <- predict(knnFit,newdata = test_data , type="prob")
> knnPredict
           no        yes
1   0.9000000 0.10000000
2   1.0000000 0.00000000
3   1.0000000 0.00000000
499 0.9000000 0.10000000
500 0.9000000 0.10000000
 [ reached 'max' / getOption("max.print") -- omitted 8511 rows ]
```

Figure 32: Prediction probability of yes-no classes after tuning

It shows probability of each class of test data with knn trained model after tuning.

```
> library(pROC)
> res.roc <- roc(test_label, knnPredict[,1])
Setting levels: control = no, case = yes
Setting direction: controls > cases
> plot.roc(res.roc, print.auc = TRUE, col="red")
> auc <-res.roc$auc
> auc
Area under the curve: 0.7889
```

Figure 33: ROC and AUC values after tuning

It shows roc and auc values of y variable from test set through knn prediction while removing the first column.

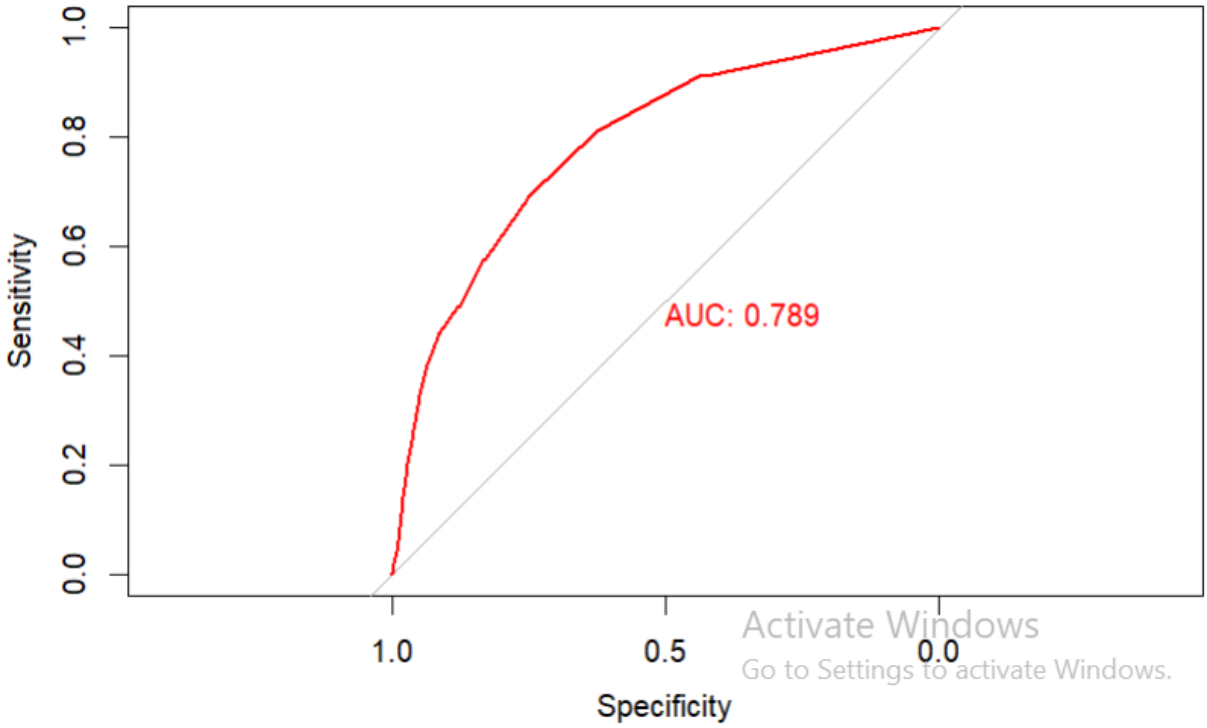

Figure 34: ROC and AUC curve of tuned Knn model

It shows the AUC value of 0.789 which is better than the previous ROC curve before tuning. However, the ROC curve doesn't bend at top left corner meaning the curve is not that good with its prediction. Even after tuning the model still can't make predictions properly for positive classes.

2.3 Naive Bayes

Based on Bayes' Theorem, Naive Bayes is a Probabilistic classifier that assumes predictor independence (Zhang, 2004). It works extremely well in many real-world applications especially with high dimensional data even though it's based on 'naive' assumptions. This model is called "Naive" as it assumes all variables in dataset are equally vital and independent.

```
> library(caret)
> trainIndex<-createDataPartition(clean_bank_data$y,p=0.8,list=FALSE)
> train_data<-clean_bank_data[trainIndex,]
> test_data<-clean_bank_data[-trainIndex,]
>
> train_data$y<-as.factor(train_data$y)
> test_data$y<-as.factor(test_data$y)
```

Figure 35: Data partition into train-test for naive model

As in other models, its same for naive model as well. Data partition into train-test in 80:20 ratio then conversion of y variable of both train-test into a factor.

```
> control <- trainControl(
+     method = "cv",
+     number = 5,
+     sampling = "up",
+     classProbs = TRUE,
+     summaryFunction = twoClassSummary)
>
```

Figure 36: Train control method for naive bayes model

It shows the control method while training the naïve model by up sampling (increasing the minority class to manage class imbalance) and on the basis of ROC and AUC.

```
> nb_model <- train(
+     y ~ .,
+     data = train_data,
+     method = "naive_bayes",
+     trControl = control,
+     metric = "ROC"
+ )
>
```

Figure 37: Training the naïve bayes model

The train_data has been trained by Naïve Bayes model with the control as mentioned above then on the basis of ROC and AUC.

```
> nb_predictions <- predict(nb_model, test_data)
> nb_probabilities <- predict(nb_model, test_data, type = "prob")
```

Figure 38: Prediction and probability of the naïve model

Due to class imbalance the model is bound to predict 'no' class more than 'yes'. But due to use of up sampling as training control method for the model, it got slightly better at predicting 'yes' class than before.

```
> confusionMatrix(nb_predictions, test_data$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7852  809
       yes  110  240

               Accuracy : 0.898
                 95% CI : (0.8916, 0.9042)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 7.466e-06

                  Kappa : 0.3025

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

            Sensitivity : 0.22879
            Specificity : 0.98618
         Pos Pred Value : 0.68571
         Neg Pred Value : 0.90659
             Prevalence : 0.11641
         Detection Rate : 0.02663
   Detection Prevalence : 0.03884
      Balanced Accuracy : 0.60749

       'Positive' Class : yes
```

Figure 39: Confusion matrix of Naïve Bayes model

The confusion matrix shows accuracy of 89.8% with sensitivity of 0.22 for positive 'yes' class and 809 false negatives so, it still misses over 75% of potential subscribers with specificity of 0.98 meaning model predicts non-subscribers almost perfectly. Kappa of 0.30 indicates fair agreement between predicted and actual values but usual aim for kappa is 6 or above so, it's being biased towards negative class and mostly predicting no in the model. The positive and negative prediction value are 0.68 and 0.90 respectively which means model predicts "yes" 68.6% correctly and predicts "no" 90.7% correctly all the time. Although we require model to look for subscribers, its just simply looking more of non-subscribers whom we need to ignore. In general, the model is still not good.

The outcome of confusion matrix:

True Negative-7852

False Negative-809

False Positive-110

True Positive-240

```
> library(pROC)
> roc_obj <- roc(test_data$y, nb_probabilities$yes, levels = c("no", "yes"))
Setting direction: controls < cases
> plot(roc_obj,print.auc = TRUE, col = "cyan")
> auc_val <- auc(roc_obj)
> auc_val
Area under the curve: 0.878
```

Figure 40: ROC, AUC of Naïve model

The above code takes on y variable from test set, use yes probabilities with levels mentioned of yes-no in order to calculate ROC and AUC.

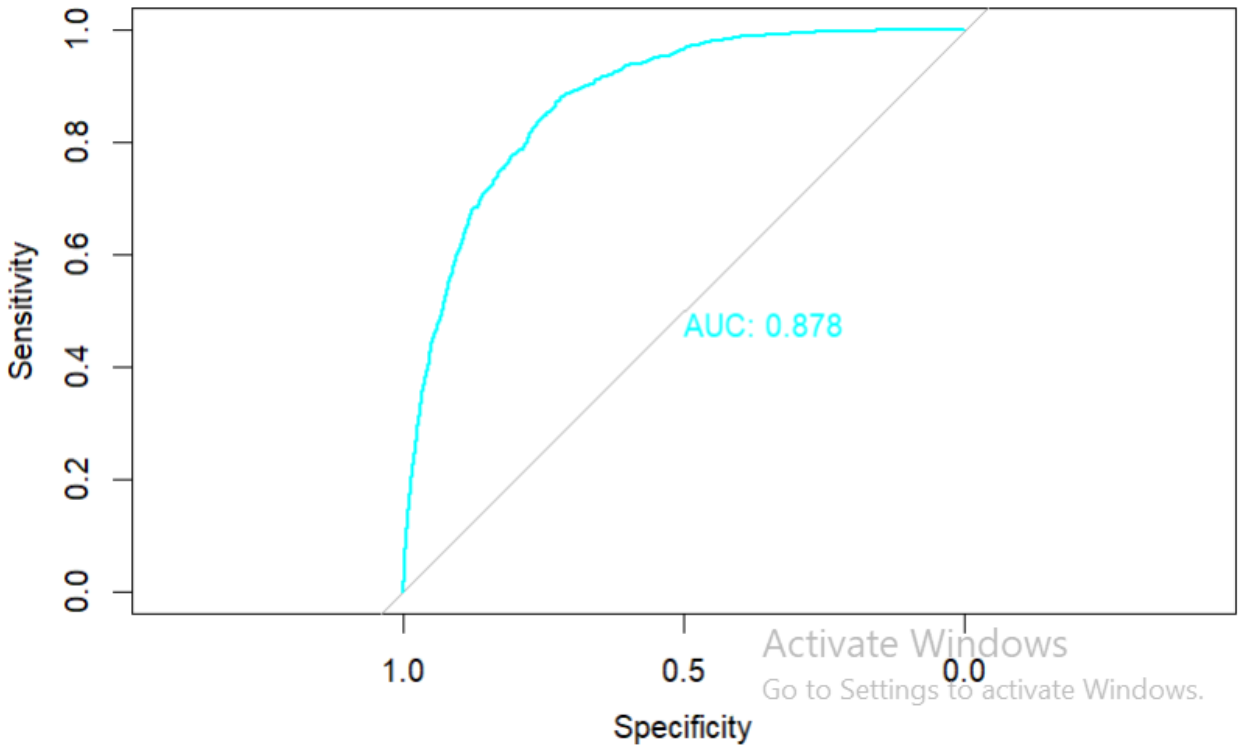

Figure 41: ROC, AUC curve of Naïve Bayes model

The above ROC curve of Naïve Bayes model shows it has relevant predictive power in identifying the subscribers for the term deposit. The AUC on the curve is 0.87 so, model assigns higher predicted probability to random chosen person who subscribed rather than non-subscriber. Although the default threshold underachieves on sensitivity, the AUC value shows that in general model can differentiate between subscribers and non-subscribers.

Model tuning

```
> nb_model <- train(
+     y ~ .,
+     data = train_data,
+     method = "naive_bayes",
+     trControl = control,
+     metric = "ROC",
+     tuneLength=5
+ )
```

Figure 42: Training the naïve model by tuning with 5 combinations of hyperparameters

The model is trained the same way as before in naïve model, but the extra thing added is the tune length of 5, which tries 5 combinations of hyperparameter.

```
> nb_predictions <- predict(nb_model, test_data)
> nb_probabilities <- predict(nb_model, test_data, type = "prob")
> confusionMatrix(nb_predictions, test_data$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7865  862
       yes   97  187

               Accuracy : 0.8936
                 95% CI : (0.887, 0.8999)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 0.001474

                  Kappa : 0.243

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

            Sensitivity : 0.17827
            Specificity : 0.98782
         Pos Pred Value : 0.65845
         Neg Pred Value : 0.90123
             Prevalence : 0.11641
         Detection Rate : 0.02075
   Detection Prevalence : 0.03152
      Balanced Accuracy : 0.58304

       'Positive' Class : yes
```

Figure 43: Prediction, probability and confusion matrix of tuned naïve bayes model

The prediction and probability are the same as before but changes in confusion matrix can be seen.

The confusion matrix shows accuracy of 89.3% with sensitivity of 0.17 for positive 'yes' class and 862 false negatives where all of the factors have degraded after tuning with specificity of 0.98 meaning model predicts non-subscribers almost perfectly. Kappa of 0.24 indicates worse agreement between predicted and actual values than before. The positive and negative prediction value are 0.65 and 0.90 respectively which means model predicts "yes" 65.8% correctly and predicts "no" 90.1% correctly all the time. In comparison to the model before tuning, it had degraded way more in all aspects making prediction worse and biased toward "no" class.

The outcome of confusion matrix:

True Negative-7865

False Negative-862

False Positive-97

True Positive-187

```
> roc_obj <- roc(test_data$y, nb_probabilities$yes, levels = c("no", "yes"))
Setting direction: controls < cases
> plot(roc_obj,print.auc = TRUE, col = "darkgreen")
> auc_val <- auc(roc_obj)
> auc_val
Area under the curve: 0.8778
```

Figure 44: The ROC and AUC calculation after tuning the model.

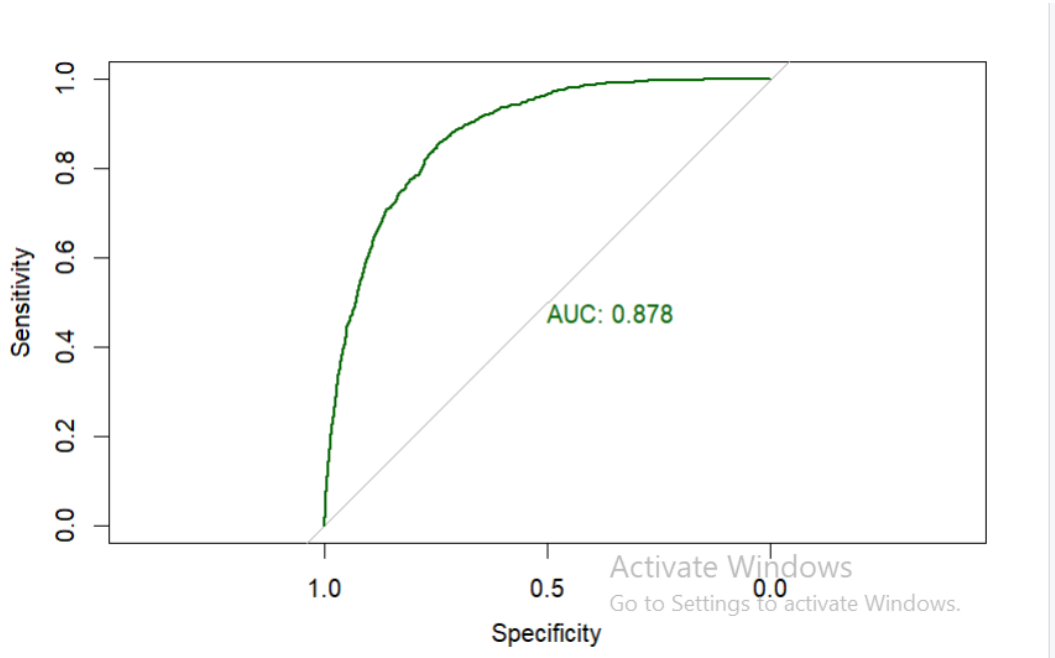

Figure 45: ROC curve of Naïve Bayes model after tuning

The above ROC curve of Naïve Bayes model after tuning shows it has relevant predictive power in identifying the subscribers for the term deposit. The AUC on the curve is 0.878 and through calculation its 0.8778 which is almost similar to the previous model. The AUC value shows that in general model can differentiate between subscribers and non-subscribers.

2.4 Decision Tree

Decision Tree is a non-parametric model that repetitively splits data into subsets based on feature values in order to make predictions (Quinlan, 1986). It is easy to comprehend and apply but can cause overfitting if pruning is not done precisely.

```
> library(caret)
> trainIndex<-createDataPartition(clean_bank_data$y,p=0.8,list=FALSE)
> train_data<-clean_bank_data[trainIndex,]
> test_data<-clean_bank_data[-trainIndex,]
```

Figure 46: Data partition for decision tree model

```
> library(rpart)
> cart_fit<-rpart(y ~ ., data=train_data, method="class")
```

Figure 47: Training the decision tree model

The given code uses rpart library to train the decision tree model. The cart_dit variable uses rpart function for y variable with all predictor variables making use of train_data set and the “class” method for categorical values in y variable.

```
> library(rpart.plot)
> rpart.plot(cart_fit)
```

Figure 48: Plot for the trained model

Making use of plot function of rpart library to create tree diagram.

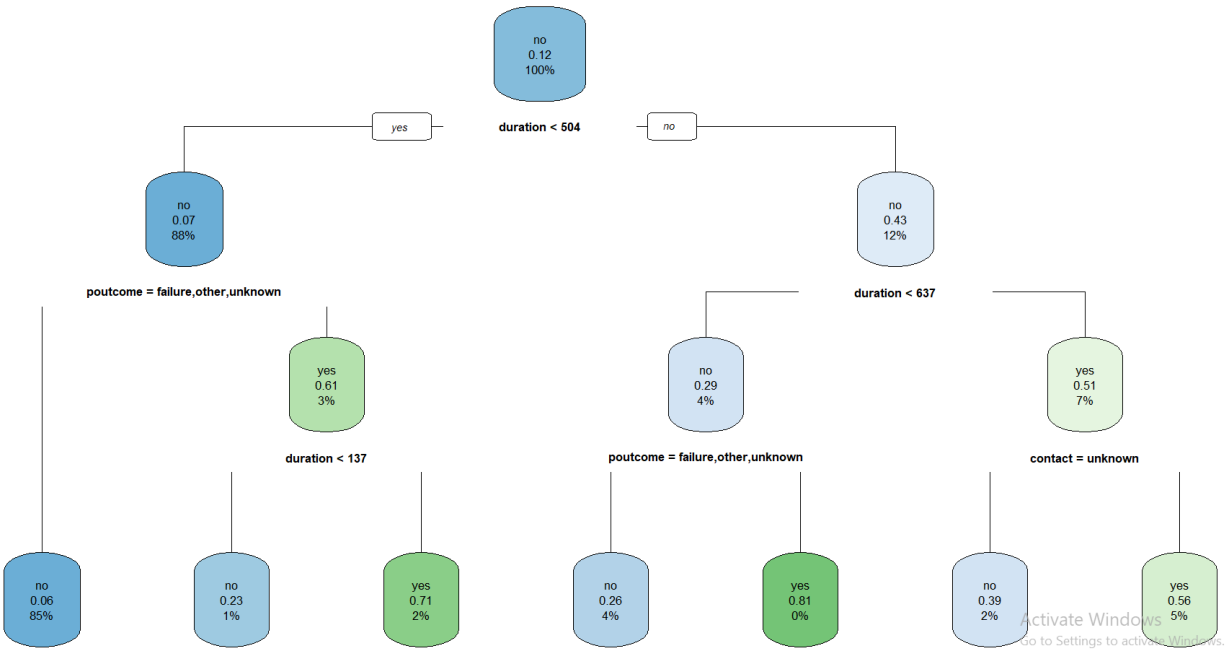

Figure 49: Tree Diagram of Decision tree model

The above tree diagram has duration (last contact duration) as vital feature at the top of tree where its branched out as call duration less than 504 sec then most likely non-subscriber else more evaluation is done. In the left branch, if poutcome (previous campaign result) is failed or unknown then it's evaluated as call duration less than 137 sec then non-subscriber, else greater than 137 to 504 sec then most likely a subscriber but if poutcome is already a success, then the prediction is confirmed to be a subscriber. In the right branch, there are high chances of subscriber; if duration less than 637 sec and poutcome failure or unknown then those are non-subscriber else prediction is being a subscriber. If the duration is greater than 637 sec with contact being unknown then prediction is non-subscriber, otherwise they are subscribers.

In conclusion, duration is the strongest and most vital predictor where longer the call is, high chance of being a subscriber. Similarly, poutcome being failed or unknown leads to high chance of non-subscribers.

```
> dt_pred<-predict(cart_fit,test_data,type="class")
> confusionMatrix(data=dt_pred, reference = test_data$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7681  636
       yes  281  413

               Accuracy : 0.8982
                 95% CI : (0.8918, 0.9044)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 5.464e-06

                  Kappa : 0.4201

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

            Sensitivity : 0.39371
            Specificity : 0.96471
         Pos Pred Value : 0.59510
         Neg Pred Value : 0.92353
             Prevalence : 0.11641
         Detection Rate : 0.04583
   Detection Prevalence : 0.07702
      Balanced Accuracy : 0.67921

       'Positive' Class : yes
```

Figure 50: Prediction and Confusion matrix of Decision tree

The above figure has prediction for the test_data set with use of cart_fit; our trained decision tree model, with type class for categorical data, which predicts yes-no values for each condition on our test set.

The confusion matrix shows accuracy of 89.82% with sensitivity of 0.39 for positive 'yes' class and 636 false negatives with specificity of 0.96 meaning model predicts non-subscribers almost all correctly. Kappa of 0.42 indicates fair agreement between predicted and actual values although good values for kappa is 6 or above. The positive and negative prediction value are 0.59 and 0.92 respectively which means model predicts "yes" 59.5% correctly and predicts "no" 92.3% correctly all the time.

The outcome of confusion matrix:

True Negative-7681

False Negative-636

False Positive-281

True Positive-413

```
> dt_prob<-predict(cart_fit,test_data,type="prob")[,"yes"]
```

Figure 51: Prediction with probabilities

It does same as doing predictions but does prediction on the basis of probabilities but only for "yes" values.

```
> library(pROC)
> roc_obj<-roc(test_data$y,dt_prob,levels=c("no","yes"))
Setting direction: controls < cases
> plot(roc_obj, col = "pink", print.auc = TRUE)
> auc_val <- auc(roc_obj)
> auc_val
Area under the curve: 0.7565
```

Figure 52: ROC and AUC value

It shows the calculation of ROC with y variable of test data, using the prediction probabilities of yes class with levels of both yes and no. Then the ROC curve is plotted, and AUC value is calculated.

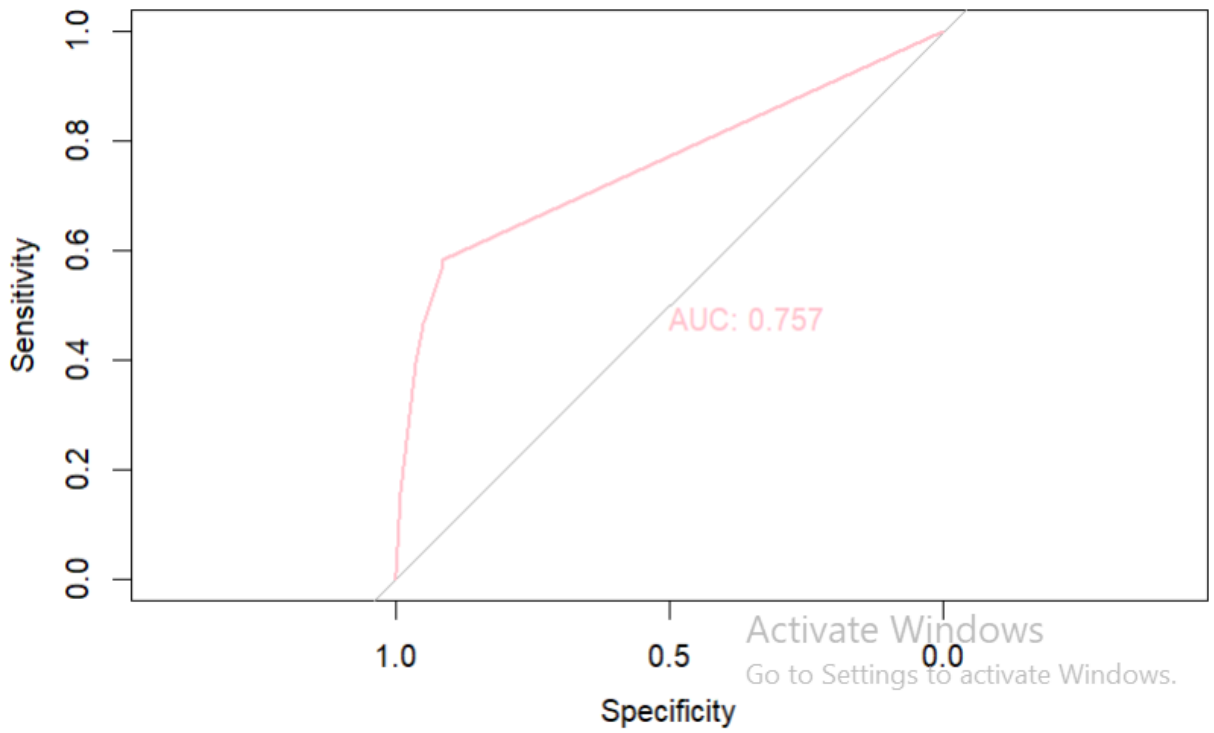

Figure 53: ROC curve of Decision Tree

The decision tree model uses variables like duration, poutcome, contact and many other variables to predict for customers subscription. The AUC values of 0.757 represents the discriminatory power of 75.7% for the model and predicts on the basis of longer calls having high chance of subscription. Due to this, the model would be best for marketing calls and encourage more engagement in calls for subscription.

Model Tuning

```
> ctrl <- trainControl(
+     method = "cv",
+     number = 5,
+     classProbs = TRUE,
+     summaryFunction = twoClassSummary)
>
> cart_model <- train(
+     y ~ .,
+     data = train_data,
+     method = "rpart",
+     trControl = ctrl,
+     metric = "ROC",
+     tuneLength = 10
+ )
```

Figure 54: Tuning the Decision Tree model

The model control is done the usual way with 5-fold cross validation with probability and twoClassSummary as in ROC, AUC. For tuning, using the caret package, train_data set is trained with rpart method, and tune length of 10 which tries 10 combinations of hyperparameter.

```
> rpart.plot(cart_model$finalModel)
```

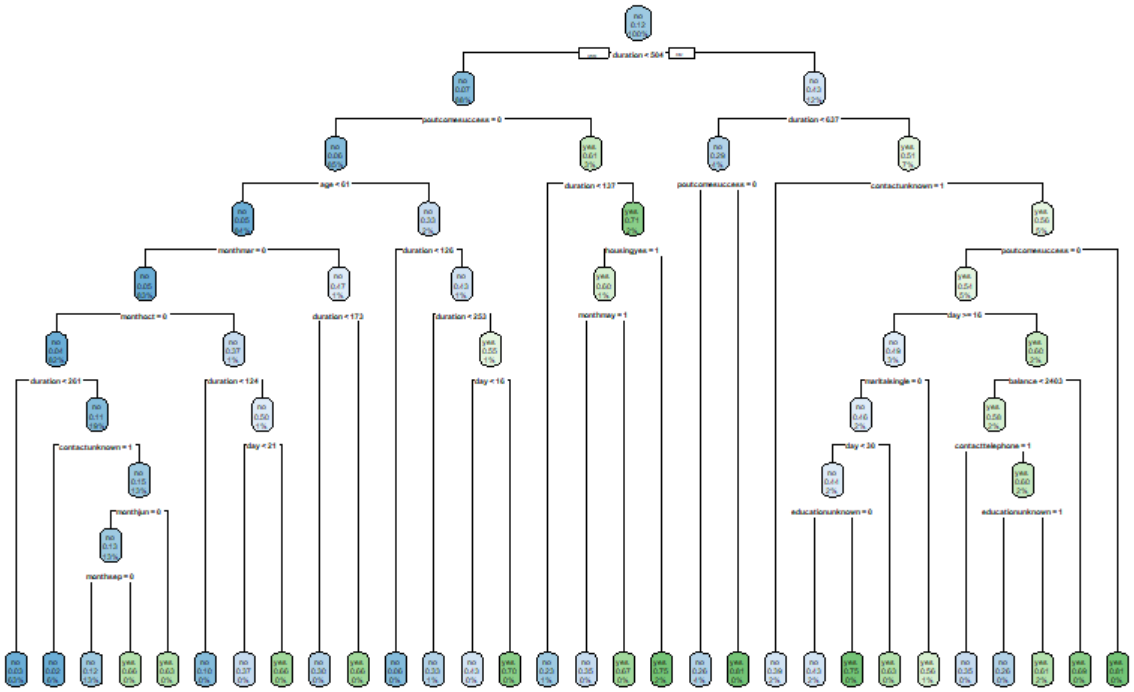

Figure 55: Tree diagram of decision tree model after tuning

From the figure, the verdict of the strongest predictor being duration is clear as the link to longer calls leads to subscriber, other variables like poutcome, contact, day, month and education also help to make clear prediction regarding the subscribers. The path where call durations are short, poutcome and contact are unknown; it most likely leads to a non-subscriber. On the other hand, path with long call duration, positive poutcome and contact with cellular and telephone leads to subscriber in most cases.

```
> tune_dt_pred<-predict(cart_model,test_data)
> tune_dt_prob<-predict(cart_model,test_data,type="prob")[,"yes"]
```

Figure 56: Tuned models prediction and probability

It does predictions and calculates probability on the tuned model for confusion matrix, and roc value.

```
> confusionMatrix(data=tune_dt_pred, reference = test_data$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7718  637
       yes  244  412

               Accuracy : 0.9022
                 95% CI : (0.8959, 0.9083)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 8.789e-09

                  Kappa : 0.4324

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

            Sensitivity : 0.39276
            Specificity : 0.96935
         Pos Pred Value : 0.62805
         Neg Pred Value : 0.92376
             Prevalence : 0.11641
         Detection Rate : 0.04572
   Detection Prevalence : 0.07280
      Balanced Accuracy : 0.68105

       'Positive' Class : yes
```

Figure 57: Confusion matrix of tuned decision tree model

The confusion matrix shows accuracy of 90.22% better than the model before tuning, with sensitivity of 0.39 for positive 'yes' class and 637 false negatives with specificity of 0.96 meaning model predicts non-subscribers almost all correctly. Kappa of 0.43 indicates fair agreement between predicted and actual values although good values for kappa is 6 or above. The positive and negative prediction value are 0.62 and 0.92 respectively which means model predicts "yes" 62.80% correctly meaning it became better at predicting the subscribers and predicts "no" 92.3% correctly all the time.

The outcome of confusion matrix:

True Negative-7718

False Negative-637

False Positive-244

True Positive-412

```
> tune_roc<-roc(test_data$y,tune_dt_prob,levels=c("no","yes"))
Setting direction: controls < cases
> plot(tune_roc, col = "purple", print.auc = TRUE)
> auc_val <- auc(tune_roc)
> auc_val
Area under the curve: 0.8565
```

Figure 58: ROC and AUC for tuned model

It shows the calculation of ROC with y variable of test data, using the prediction probabilities of yes class with levels of both yes and no. Then the ROC curve is plotted, and AUC value is calculated.

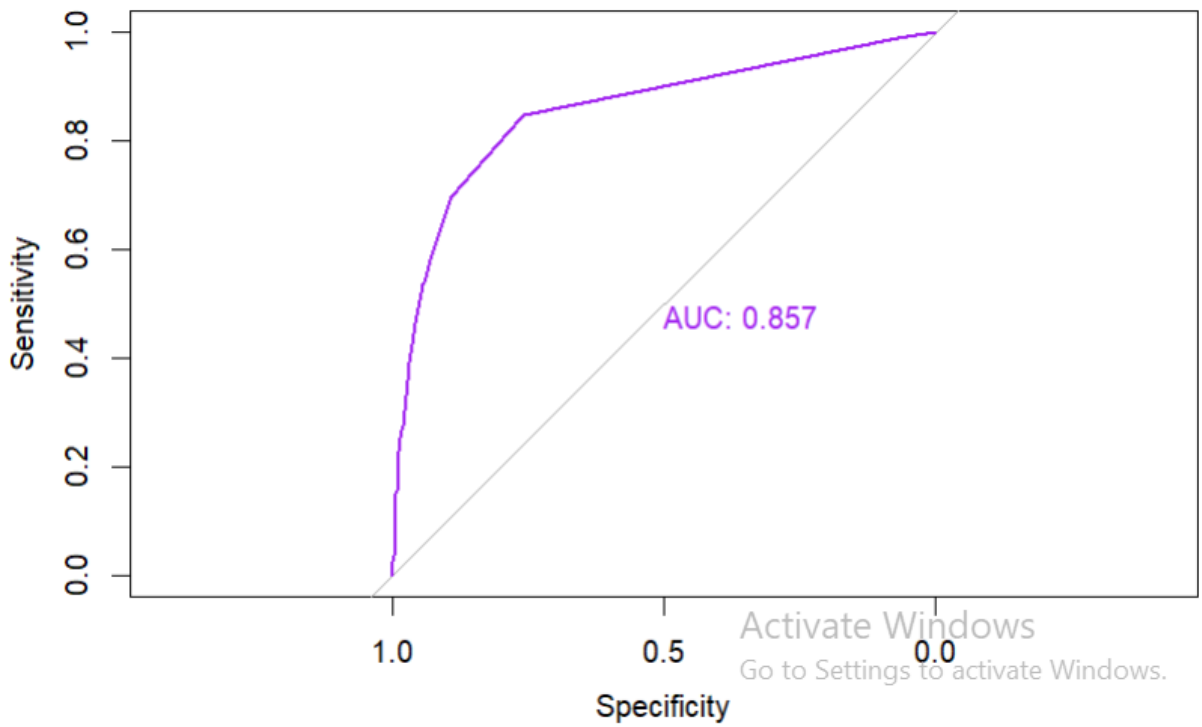

Figure 59: ROC curve of tuned decision tree model

The ROC curve has shown the AUC of 0.857 which is way better than the previous model without tuning with such AUC in the range of ~80 makes the model reliable to identify our subscribers. The decision tree model uses variables like duration, poutcome, contact and many other variables to predict for customers subscription. The model predicts on the basis of longer calls having high chance of subscription.

2.5. Combined model of Logistic regression and Decision Tree

When logistic regression and decision tree model is combined, it merges the perks of both models in order to improve the prediction performance for classification problems. This process can be done by stacking the models; first training by decision tree then fitting logistic regression model on each leaf to refine predictions. This improves accuracy, robustness and interpretability of the model.

```
#logistic regression
library(caret)
trainIndex<-createDataPartition(clean_bank_data$y,p=0.8,list=FALSE)
train_data<-clean_bank_data[trainIndex,]
test_data<-clean_bank_data[-trainIndex,]

train_data$y<-as.factor(train_data$y)
test_data$y<-as.factor(test_data$y)

#to train the model
caret_glm_mod=train(
  form=y ~ .,
  data=train_data,
  trControl=trainControl(method="cv",number=5),
  method="glm",
  family="binomial"
)

predicted_test<-predict(caret_glm_mod, newdata = test_data)
predicted_test

#threshold tuning
predicted_probs <- predict(caret_glm_mod, newdata = test_data, type = "prob")[, "yes"]
predicted_classes <- ifelse(predicted_probs > 0.3, "yes", "no")

#model tuning
library(glmnet)

# Convert data to matrix form (glmnet requires this)
x <- model.matrix(y ~ ., data = train_data)[,-1]
y <- train_data$y

# Set up cross-validation with glmnet
set.seed(123)
tuned_model <- train(
  x = x,
  y = y,
  method = "glmnet",
```

Figure 60: Logistic regression model part 1

```
# Set up cross-validation with glmnet
set.seed(123)
tuned_model <- train(
  x = x,
  y = y,
  method = "glmnet",
  trControl = trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction = twoClassSummary),
  metric = "ROC",
  tuneLength = 10
)

x_test <- model.matrix(y ~ ., data = test_data)[,-1]
predicted_probs <- predict(tuned_model, newdata = x_test, type = "prob")[, "yes"]
predicted_classes <- ifelse(predicted_probs > 0.3, "yes", "no")
```

Figure 61: Logistic regression model part 2

In the above figure of 51 and 52, it is the repeat process of what was done in logistic regression model, data split into 80:20, training the model with caret package, doing threshold tuning and finally again training it with hyperparameter with tune length of 10 and extracting the predictions and probability of model after tuning.

```
#decision tree

library(rpart)
cart_fit<-rpart(y ~ ., data=train_data, method="class")

dt_pred<-predict(cart_fit,test_data,type="class")
dt_prob<-predict(cart_fit,test_data,type="prob")[,"yes"]

#tuning
ctrl <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary)

cart_model <- train(
  y ~ .,
  data = train_data,
  method = "rpart",
  trControl = ctrl,
  metric = "ROC",
  tuneLength = 10
)

tune_dt_pred<-predict(cart_model,test_data)
tune_dt_prob<-predict(cart_model,test_data,type="prob")[,"yes"]
```

Figure 62: Decision Tree model

Similarly, in figure 53; it is the repeat process of what was done in decision tree model. Using the rpart library, training the model then tuning it again with hyperparameter and extracting the prediction and probability.

```
> train_dt_prob<-predict(cart_model,train_data,type="prob")[,"yes"]
> test_dt_prob<-predict(cart_model,test_data,type="prob")[,"yes"]
```

Figure 63: Probability of train and test data from decision tree model

The code predicts the decision tree model in both train and test data with probability type and for only “yes” class labels.

```
> glm_train_prob<-predict(tuned_model, newdata = model.matrix(y ~ ., train_data)[,-1], type = "prob")[, "yes"]
> glm_test_prob<-predict(tuned_model, newdata = model.matrix(y ~ ., test_data)[,-1], type = "prob")[, "yes"]
```

Figure 64: Probability of train and test data from logistic regression model

The code predicts the logistic regression model in both train and test data and works on data in the form of matrix including all predictor variable in it, with probability type and for only “yes” class labels.

```
> combined_train<-data.frame(
+     dt_probability=train_dt_prob,
+     glm_probability=glm_train_prob,
+     y=train_data$y
+ )
```

Figure 65: Combining the training probability of DT and LR

The code creates a data frame with the combination of probabilities from both decision tree and logistic regression with y variable of train_data set.

```
> combined_test<-data.frame(
+     dt_probability=test_dt_prob,
+     glm_probability=glm_test_prob,
+     y=test_data$y
+ )
```

Figure 66: Combining the testing probability of DT and LR

The code creates a data frame with the combination of probabilities from both decision tree and logistic regression with y variable of test_data set.

```
> combined_model<-train(
+     y ~ .,
+     data=combined_train,
+     method="glm",
+     family="binomial",
+     trControl=trainControl(method="cv", number = 5)
+ )
```

Figure 67: Training the combined model of DT and LR

It trains the combined model of decision tree and logistic regression for only train set using glm method and 5-fold cross validation.

```
> combined_prob<-predict(combined_model, newdata = combined_test, type = "prob")[,"yes"]
> combined_pred <- factor(ifelse(combined_prob > 0.3, "yes", "no"), levels = levels(combined_test$y))
> confusionMatrix(combined_pred,combined_test$y, positive="yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7552  497
       yes  410  552

               Accuracy : 0.8993
                 95% CI : (0.8929, 0.9055)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 1.069e-06

                  Kappa : 0.4925

 Mcnemar's Test P-Value : 0.004296

            Sensitivity : 0.52622
            Specificity : 0.94851
         Pos Pred Value : 0.57380
         Neg Pred Value : 0.93825
             Prevalence : 0.11641
         Detection Rate : 0.06126
   Detection Prevalence : 0.10676
      Balanced Accuracy : 0.73736

       'Positive' Class : yes
```

Figure 68: Probability and Confusion Matrix of combined model

The code predicts the combined_test dataset with combined_model which was trained for probability type and “yes” labels only. Then, storing the probability in combined_pred with the threshold of 0.3, if its greater than 0.3 then yes or else no in sense of probability.

The confusion matrix shows accuracy of 89.93% with sensitivity of 0.52 and 410 false positive for positive ‘yes’ class and 497 false negatives with specificity of 0.94. Kappa of 0.49 indicates fair agreement between predicted and actual values which suggest the model is moderately effective despite the imbalance. The positive and negative prediction value are 0.57 and 0.93 respectively which means model predicts “yes” 57.38% correctly at predicting the subscribers and predicts “no” 93.82% correctly most of the time.

The outcome of confusion matrix:

True Negative-7552

False Negative-497

False Positive-410

True Positive-552

```
> library(pROC)
> combined_roc <- roc(combined_test$y, combined_prob)
Setting levels: control = no, case = yes
Setting direction: controls < cases
> plot(combined_roc, col = "maroon", print.auc=TRUE)
> auc(combined_roc)
Area under the curve: 0.9106
```

Figure 69: ROC and AUC for combined model.

It shows the calculation of ROC with y variable of combined test data and probability of combined model. Then the ROC curve is plotted, and AUC value is calculated.

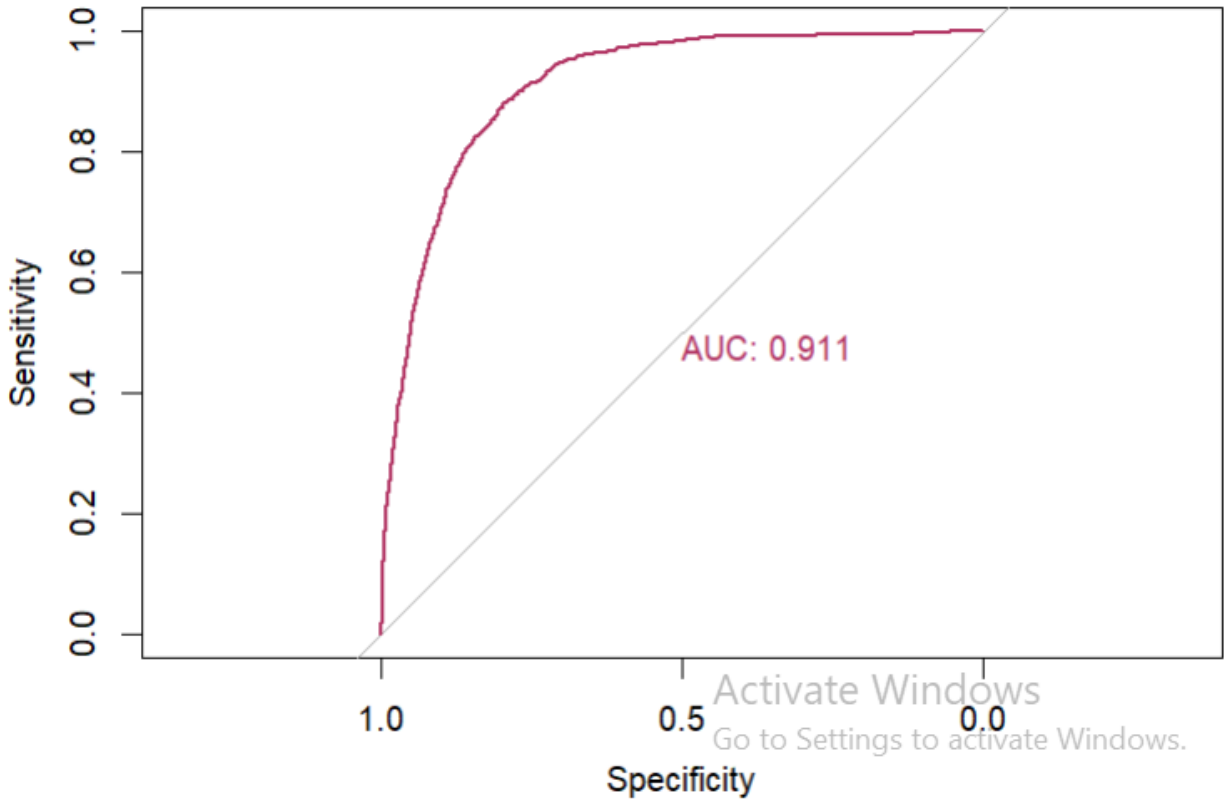

Figure 70: ROC curve of combined model

The curve shows the AUC of 0.911 means excellent discrimination of model and model being able to learn meaningful patterns from the dataset. There's a 91.1% chance the model will rank a randomly chosen positive instance (customer who subscribes) higher than a randomly chosen negative one. The model separates potential subscribers really well from non-subscribers.

Tuning the combined model

```
> library(glmnet)
> combined_train$y <- as.factor(combined_train$y)
> combined_test$y <- as.factor(combined_test$y)
```

Figure 71: Factor of y variable

Converting the y variable of both combined train test into a factor

```
> tuned_combined_model <- train(
+     y ~ .,
+     data = combined_train,
+     method = "glmnet",
+     trControl = trainControl(
+         method = "cv",
+         number = 5,
+         classProbs = TRUE,
+         summaryFunction = twoClassSummary
+     ),
+     metric = "ROC",
+     tuneLength = 10
+ )
```

Figure 72: Training the combined model for tuning

Model is trained through glmnet method on combined train dataset with train control method being 5-fold cross validation on the basis of probability and ROC, AUC then training output is done from AUC value and has hyperparameter tune length of 10.

```
> tuned_combined_prob <- predict(tuned_combined_model, newdata = combined_test, type = "prob")[,"yes"]
> tuned_combined_pred <- factor(ifelse(tuned_combined_prob > 0.3, "yes", "no"), levels = levels(combined_test$y))
```

Figure 73: Probability of tuned combined model.

The code predicts the combined_test dataset with tuned_combined_model which was trained for probability type and “yes” labels only. Then, storing the probability in tuned_combined_pred with the threshold of 0.3, if its greater than 0.3 then yes or else no in sense of probability.

```
> confusionMatrix(tuned_combined_pred, combined_test$y, positive = "yes")
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7596  537
       yes  366  512

               Accuracy : 0.8998
                 95% CI : (0.8934, 0.9059)
    No Information Rate : 0.8836
    P-Value [Acc > NIR] : 5.384e-07

                  Kappa : 0.4758

 Mcnemar's Test P-Value : 1.538e-08

            Sensitivity : 0.48808
            Specificity : 0.95403
         Pos Pred Value : 0.58314
         Neg Pred Value : 0.93397
             Prevalence : 0.11641
         Detection Rate : 0.05682
   Detection Prevalence : 0.09744
      Balanced Accuracy : 0.72106

       'Positive' Class : yes
```

Figure 74: Confusion matrix of tuned combined model

The confusion matrix shows accuracy of 89.98% with sensitivity of 0.48 and 366 false positive for positive 'yes' class and 537 false negatives with specificity of 0.95. Kappa of 0.47 indicates fair agreement between predicted and actual values which suggest the model is moderately effective despite the imbalance. The positive and negative prediction value are 0.58 and 0.93 respectively which means model predicts "yes" 58.31% correctly at predicting the subscribers and predicts "no" 93.39% correctly most of the time. Most of the aspects of tuned model have degraded with the exception of positive prediction value.

The outcome of confusion matrix:

True Negative-7596

False Negative-537

False Positive-366

True Positive-512

```
> tuned_combined_roc <- roc(combined_test$y, tuned_combined_prob)
Setting levels: control = no, case = yes
Setting direction: controls < cases
> plot(tuned_combined_roc, col = "purple", print.auc = TRUE)
> auc(tuned_combined_roc)
Area under the curve: 0.9105
```

Figure 75: ROC and AUC for tuned combined model

It shows the calculation of ROC with y variable of combined test data and probability of tuned combined model. Then the ROC curve is plotted, and AUC value is calculated.

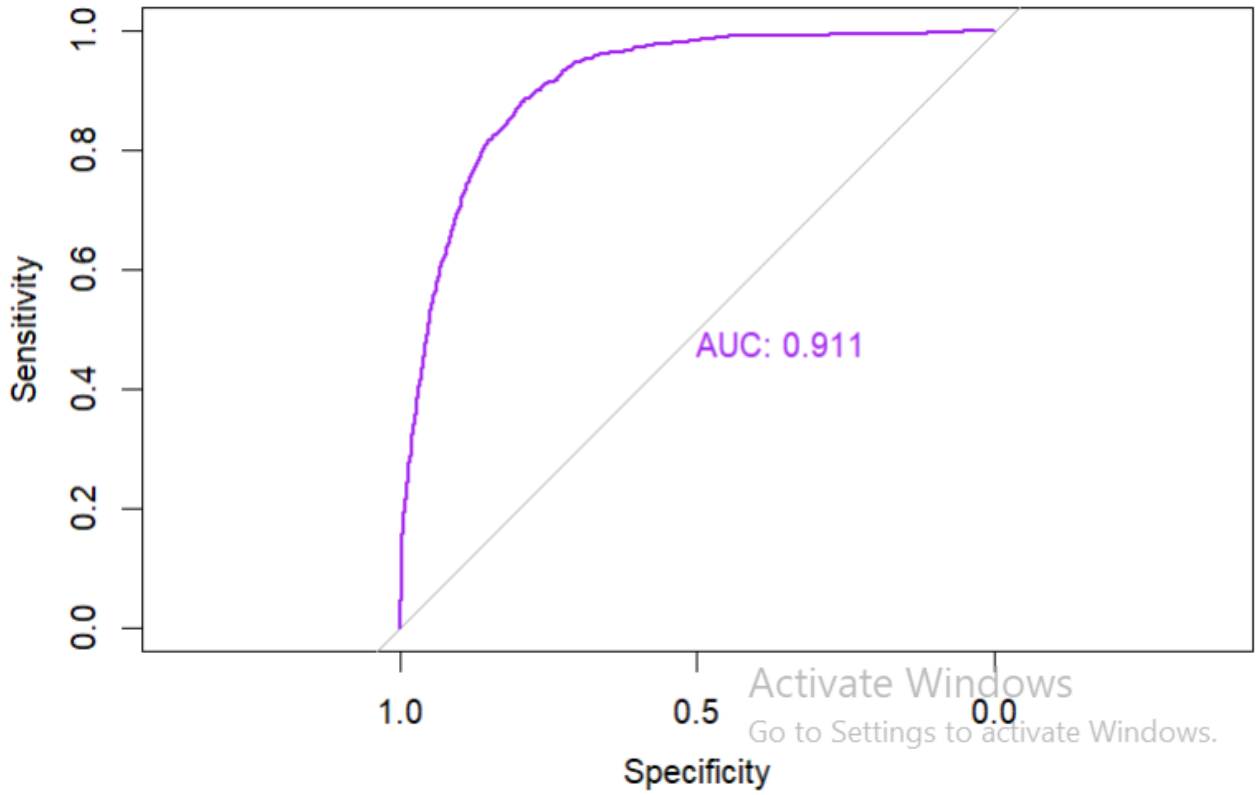

Figure 76: ROC curve of tuned combined model

The curve shows the AUC of 0.911, similar to the one before tuning, meaning it has excellent discrimination of model and model being able to learn meaningful patterns from the dataset. There's a 91.1% chance the model will rank a randomly chosen positive instance (customer who subscribes) higher than a randomly chosen negative one. The model separates potential subscribers really well from non-subscribers.

Results and Interpretation

The following table consist of all vital points require to analyse the prediction power of a models. Except for KNN model every other model has positive class “yes” but in knn its “no” for positive class.

Model	Accuracy	Kappa	Sensitivity	Specificity	Positive Predictive Value (PPV)	Negative Predictive Value (NPV)
LR	88.81%	0.48	0.58	0.92	0.51	0.94
KNN	88.19%	0.19	0.97	0.16	0.89	0.47
NB	89.36%	0.24	0.17	0.98	0.65	0.90
DT	90.22%	0.43	0.39	0.96	0.62	0.92
LR &DT	89.98%	0.47	0.48	0.95	0.58	0.93

3. Model Interpretation

The best model on the basis of confusion matrix for predicting subscriber and non-subscriber is the combination of Decision Tree and Logistic Regression. It works on the basis of probability of train and test data of both decision tree and logistic regression model which is combined then trained by glmnet method with 5-fold cross validation technique and ROC metric. This model performs good as the combine model complements each other; for example, DT handles the exceptions which LR cannot as it requires clear decision boundary, similarly LR smooths out predictions and reduces the overfitting tendency of DT. The accuracy of 89.98%, strong kappa of 0.47, balanced sensitivity and specificity of 0.48 and 0.95 respectively and finally satisfactory NPV and PPV of 0.93 and 0.58 is all due to combine ability of both models which couldn’t be achieved from other models singlehandedly. The model performance can be better if missing values are imputed by “rf” method rather than discarding them, use SMOTE method to manage the class imbalance, adjusting the prediction threshold through ROC curve and many more other ways.

4. Conclusion

The report critically evaluates the predictive performance of multiple machine learning models to identify the subscribers in a bank marketing dataset. The models include Logistic Regression, K-Nearest Neighbour, Naïve Bayes, Decision Tree and a hybrid ensemble of LR and DT. The overall evaluation includes accuracy, kappa, sensitivity, specificity and both positive and negative predictive values. The ensemble model outperformed other standalone models with accuracy of 89.98%, sensitivity of 0.48 and kappa of 0.47 which supports ensemble learning as a superior approach for complex classification problems (Cruz et al., 2020).

The model's predictive manner aligned well with the patterns identified during EDA. Variables like duration, poutcome, and contact appeared to be strong predictors which played huge role in the predictive supervised machine learning models. Though, there is no action of removing any variables to increase models' accuracy, yet the ensemble model turned out better in a simpler way. The accuracy of the ensemble model is high at 89% but with specificity 0.95 and NPV of 0.93, it tilts toward prediction of non-subscriber in a better way than for subscribers. Even if the sensitivity is 0.48 (on the low side), PPV is 0.58 and kappa is 0.47; this sums up that the model correctly identifies a fair portion of actual subscribers and moderate amount of kappa shows that model is still making meaningful predictions rather than random guessing in the heavy imbalanced dataset of bank marketing.

Bibliography

- Bishop, C.M., 2006. *Pattern recognition and machine learning*. New York: Springer.
- Cover, T. and Hart, P., 1967. Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), pp. 21–27.
- Cruz, L.A., Luna, J.M., Ventura, S. and Romero, C., 2020. *A genetic programming approach for feature engineering in educational data mining*. Knowledge-Based Systems, 199, p.105979.
- Hosmer, D.W., Lemeshow, S. and Sturdivant, R.X., 2013. *Applied logistic regression*. 3rd ed. Hoboken, NJ: Wiley.
- Quinlan, J.R., 1986. Induction of decision trees. Machine Learning, 1(1), pp. 81–106.
- Zhang, H., 2004. The optimality of naive Bayes. In: Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004). Miami Beach, Florida, USA, pp. 562–567.