Customer Churn Prediction using Logistic Regression, Classification Tree and Random Forest Models

Padmashri Saravanan

26 May 2020

Contents

1 Overview

1.1 Introduction to Churn Prediction

Churn prediction means detecting which customers are likely to cancel a subscription to a service based on how they use the service. It is a critical prediction for many businesses because acquiring new clients often costs more than retaining existing ones. Once you can identify those customers that are at risk of cancelling, you should know exactly what marketing action to take for each individual customer to maximise the chances that the customer will remain.

Different customers exhibit different behaviours and preferences, so they cancel their subscriptions for various reasons. It is critical, therefore, to proactively communicate with each of them in order to retain them in your customer list.

Harnessed properly, churn prediction can be a major asset in getting a clearer picture of your customers' experience with your product. Although the range of potential factors behind churn can be complex, stopping churn often revolves around a tailored approach to improving customer experience. Churn prediction gives you the chance to improve a customer's experience before they leave for good.

This data set contains details of a bank's customers and the target variable is a binary variable reflecting the fact whether the customer left the bank (closed his account) or he continues to be a customer. In this analysis we predict the customer churning using Logistic Regression, Classification Tree and Random Forest algorithms and conclude that the logistic regression model and random forest model work better than the Classification Tree model. The accuracies are **0.78** for Logistic Regression, **0.78** for Classification Tree and **0.79** for Random Forest, with 0.5 as the threshold value.

1.2 The Dataset

Source

https://www.kaggle.com/shrutimechlearn/churn-modelling

Dimensions

Length	Columns
10000	14

Variables

```
##  [1] "RowNumber"       "CustomerId"      "Surname"         "CreditScore"
##  [5] "Geography"       "Gender"          "Age"             "Tenure"
##  [9] "Balance"         "NumOfProducts"   "HasCrCard"       "IsActiveMember"
## [13] "EstimatedSalary" "Exited"
```

```
# counting the customers who churn
sum(bank$Exited==1)
```

```
## [1] 2037
```

Thus, we have **2037** customers that churned, that's about 20%! Now, let's analyse the dataset to predict future customers that would churn using 3 different models.

2 Methods & Analysis

2.1 Exploring the Data

```
## tibble [10,000 x 14] (S3: tbl_df/tbl/data.frame)
##  $ RowNumber      : num [1:10000] 1 2 3 4 5 6 7 8 9 10 ...
```

```
##  $ CustomerId     : num [1:10000] 15634602 15647311 15619304 15701354 15737888 ...
##  $ Surname        : chr [1:10000] "Hargrave" "Hill" "Onio" "Boni" ...
##  $ CreditScore    : num [1:10000] 619 608 502 699 850 645 822 376 501 684 ...
##  $ Geography      : chr [1:10000] "France" "Spain" "France" "France" ...
##  $ Gender         : chr [1:10000] "Female" "Female" "Female" "Female" ...
##  $ Age            : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:10000] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
##  $ HasCrCard      : num [1:10000] 1 0 1 0 1 1 1 1 0 1 ...
##  $ IsActiveMember : num [1:10000] 1 1 0 0 1 0 1 0 1 1 ...
##  $ EstimatedSalary: num [1:10000] 101349 112543 113932 93827 79084 ...
##  $ Exited         : num [1:10000] 1 0 1 0 0 1 0 1 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   RowNumber = col_double(),
##   ..   CustomerId = col_double(),
##   ..   Surname = col_character(),
##   ..   CreditScore = col_double(),
##   ..   Geography = col_character(),
##   ..   Gender = col_character(),
##   ..   Age = col_double(),
##   ..   Tenure = col_double(),
##   ..   Balance = col_double(),
##   ..   NumOfProducts = col_double(),
##   ..   HasCrCard = col_double(),
##   ..   IsActiveMember = col_double(),
##   ..   EstimatedSalary = col_double(),
##   ..   Exited = col_double()
##   .. )

##    RowNumber       CustomerId         Surname           CreditScore
##  Min.   :    1   Min.   :15565701   Length:10000       Min.   :350.0
##  1st Qu.: 2501   1st Qu.:15628528   Class :character   1st Qu.:584.0
##  Median : 5000   Median :15690738   Mode  :character   Median :652.0
##  Mean   : 5000   Mean   :15690941                      Mean   :650.5
##  3rd Qu.: 7500   3rd Qu.:15753234                      3rd Qu.:718.0
##  Max.   :10000   Max.   :15815690                      Max.   :850.0
##   Geography            Gender               Age            Tenure
##  Length:10000       Length:10000       Min.   :18.00   Min.   : 0.000
##  Class :character   Class :character   1st Qu.:32.00   1st Qu.: 3.000
##  Mode  :character   Mode  :character   Median :37.00   Median : 5.000
##                                        Mean   :38.92   Mean   : 5.013
##                                        3rd Qu.:44.00   3rd Qu.: 7.000
##                                        Max.   :92.00   Max.   :10.000
##     Balance       NumOfProducts    HasCrCard      IsActiveMember
##  Min.   :     0   Min.   :1.00   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:     0   1st Qu.:1.00   1st Qu.:0.0000   1st Qu.:0.0000
##  Median : 97199   Median :1.00   Median :1.0000   Median :1.0000
##  Mean   : 76486   Mean   :1.53   Mean   :0.7055   Mean   :0.5151
##  3rd Qu.:127644   3rd Qu.:2.00   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :250898   Max.   :4.00   Max.   :1.0000   Max.   :1.0000
##  EstimatedSalary         Exited
##  Min.   :    11.58   Min.   :0.0000
```

```
##  1st Qu.: 51002.11   1st Qu.:0.0000
##  Median :100193.91   Median :0.0000
##  Mean   :100090.24   Mean   :0.2037
##  3rd Qu.:149388.25   3rd Qu.:0.0000
##  Max.   :199992.48   Max.   :1.0000
```

2.1.1 Cleaning the data

From the summary above, we can see that there are no missing values under any variable.

Now, we remove the `CustomerId` and `Surname` variable, since they won't be of any help for the analysis.

```
bank_clean <- bank [,c(-2,-3)]
```

2.1.2 Discrete Variables

Geography, Gender, Number of Products, Credit Card and Active Membership

We see that there are 3 classes in `Geography`,2 labels in `Gender` and 4 classes in `NumOfProducts`.

```
unique(bank_clean$Gender)
```

```
## [1] "Female" "Male"
```

```
unique(bank_clean$Geography)
```

```
## [1] "France"  "Spain"   "Germany"
```

```
unique(bank_clean$NumOfProducts)
```

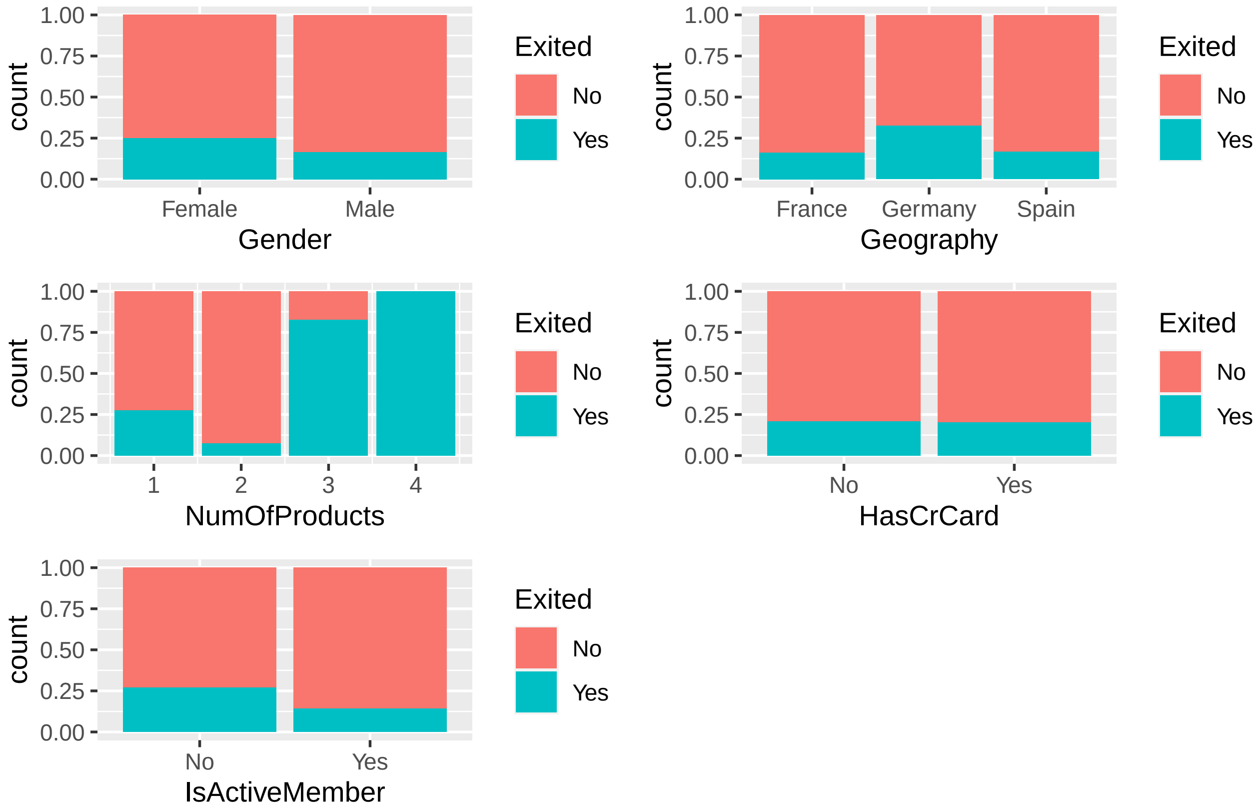
```
## [1] 1 3 2 4
```

Checking the Churn Distributions

We first convert any binary values to factors to help visualize the data.

```
categorical <- bank_clean %>%
  mutate(Exited = ifelse(Exited==1,"Yes","No")) %>%
  mutate(HasCrCard = ifelse(HasCrCard==1,"Yes","No")) %>%
  mutate(IsActiveMember = ifelse(IsActiveMember==1,"Yes","No"))
```

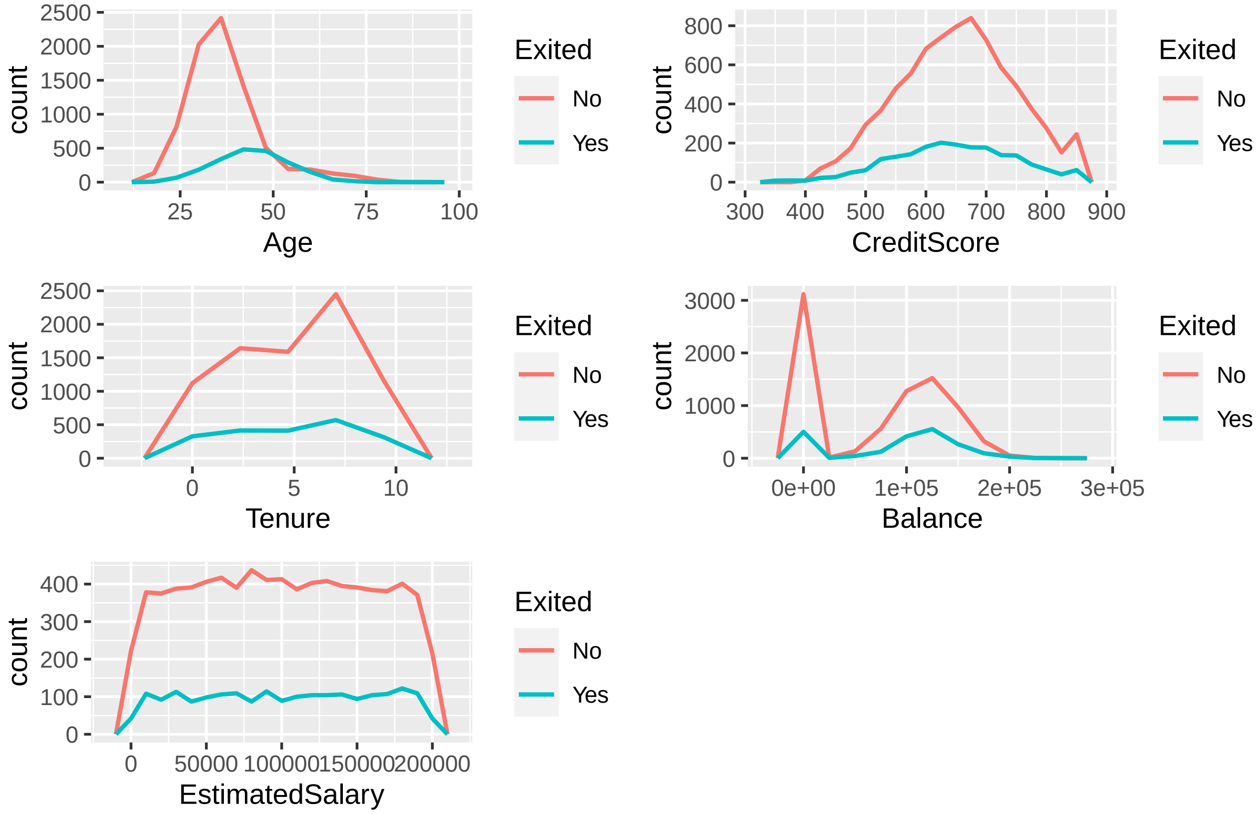
Next, we check the churn rate (that is, the customers who have exited) against each variable.



We can clearly see that there are a greater number of females who Exited the service and customers from Germany Exited the most, followed by a close tie between France and Spain. It is also clear that customers with 4 products have the highest churn rate, followed by 3, 1 and 2 products. Customers having a credit card does not really affect the churn rate, but customers who are an active member have a lower churn rate than those who aren't.

2.1.3 Continuous Variables

Age, Credit Score, Tenure, Balance, Estimated Salary



The age of customers who exited (churned) are positively skewed, that is, customers who churned are more likely to close the account after the age of about 40 years. Contrastingly the customers who do not churn have a much higher peak, meaning a large group of current customers have been using the service till about 35 years of age.

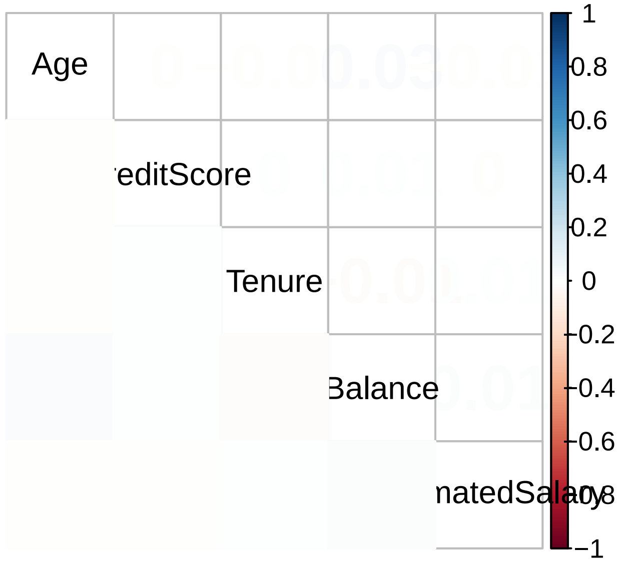
We can notice that customers who do not churn with a credit score of 600 are extremely high and customers who churn with a score of 600 are relatively low. The tenure of customers who did not exit is negatively skewed, that is, customers with about 7 years tenure are less likely to close the account but customers who do churn also have about 7 years of tenure.

Both customers who churn and do not churn regardless of Balance seem to have a similar exit rate distribution, though the customers who have a lower amount of Balance is also seen to not have exited the bank's services. Lastly, we can see that there is no particular distribution for the `EstimatedSalary` variable.

Correlations between Variables

Now we check for correlations among the variables.

```
categorical %>%
  dplyr::select(Age, CreditScore, Tenure, Balance, EstimatedSalary) %>%
  cor() %>%
  corrplot.mixed(upper = "number", lower = "color", tl.col = "black", number.cex=2)
```



It is vivid that there is a negligible amount of correlation among the variables.

Checking the Churn Rate for the complete dataset

Total	Churn_count	Churn_probability
10000	2037	0.2037

This tells us that there are about **20.3%** customers who churn!

2.2 Logistic Regression Model

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

2.2.1 Data Cleaning

We first create dummy variables for all character variables, after converting back the `Exited` variable to binary values.

```
bank_new <- categorical %>%
  mutate(Exited = ifelse(Exited=="Yes",1,0))
dummy <- dummyVars(" ~ .", data = bank_new)
dummy <- data.frame(predict(dummy, newdata = bank_new))
```

Now, we split the data into training and test sets (75% against 25%):

```
set.seed(818)
assignment <- sample(0:1, size= nrow(dummy), prob = c(0.75,0.25), replace = TRUE)
train <- dummy[assignment == 0, ]
test <- dummy[assignment == 1, ]
```

Let's also examine if the churn rates of both sets are not too far off.

The Training Set

Total	Churn_count	Churn_probability
7472	1517	0.2030246

The Test Set

Total	Churn_count	Churn_probability
2528	520	0.2056962

2.2.2 Training Set Models

We first use all columns to build the first model, `model1`.

```
model1 <- glm(Exited ~., family = "binomial", data = train)
```

Then we use AIC, to easily test the model's performance, and to exclude variables based on their significance and create `model2`.

```
model2 <- stepAIC(model1, trace = 0)
summary(model2)
```

```
##
## Call:
## glm(formula = Exited ~ CreditScore + GeographyGermany + GenderFemale +
##     Age + Tenure + Balance + NumOfProducts + HasCrCardNo + IsActiveMemberNo,
##     family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2942  -0.6558  -0.4602  -0.2727   2.9777
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.977e+00  2.854e-01 -17.436  < 2e-16 ***
## CreditScore      -6.593e-04  3.235e-04  -2.038   0.0416 *
## GeographyGermany  7.185e-01  7.317e-02   9.819  < 2e-16 ***
## GenderFemale      5.078e-01  6.299e-02   8.062 7.51e-16 ***
## Age               7.344e-02  2.982e-03  24.629  < 2e-16 ***
## Tenure           -2.006e-02  1.090e-02  -1.841   0.0656 .
## Balance           2.742e-06  5.925e-07   4.628 3.70e-06 ***
## NumOfProducts    -9.091e-02  5.494e-02  -1.655   0.0980 .
## HasCrCardNo       1.065e-01  6.800e-02   1.566   0.1174
## IsActiveMemberNo  1.004e+00  6.627e-02  15.157  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7540.2  on 7471  degrees of freedom
## Residual deviance: 6407.2  on 7462  degrees of freedom
## AIC: 6427.2
##
## Number of Fisher Scoring iterations: 5
```

The Variance Inflation Factor (VIF) is used to detect the presence of multicollinearity. Variance inflation factors (VIF) measure how much the variance of the estimated regression coefficients are inflated as compared to when the predictor variables are not linearly related. Hence we use the VIF function to check for multicollinearity:

```
vif(model2)
```

```
##      CreditScore GeographyGermany     GenderFemale              Age
```

```
##         1.001635         1.209144         1.003995         1.071438
##           Tenure          Balance    NumOfProducts      HasCrCardNo
##         1.004385         1.290699         1.080591         1.002288
## IsActiveMemberNo
##         1.069649
```

We see that all VIF values of `model2` are lesser than 2, but the p-value for `HasCrCardNo` is still relatively high, so we remove it to create `model3`:

```
model3 <-
  glm(formula = Exited ~  CreditScore + GeographyGermany + GenderFemale
      + Age + Tenure + Balance + NumOfProducts + IsActiveMemberNo,
      family = "binomial", data = train)
summary(model3)
```

```
##
## Call:
## glm(formula = Exited ~ CreditScore + GeographyGermany + GenderFemale +
##     Age + Tenure + Balance + NumOfProducts + IsActiveMemberNo,
##     family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3070  -0.6562  -0.4604  -0.2731   2.9674
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.943e+00  2.845e-01 -17.378  < 2e-16 ***
## CreditScore      -6.572e-04  3.235e-04  -2.032   0.0422 *
## GeographyGermany  7.163e-01  7.314e-02   9.793  < 2e-16 ***
## GenderFemale      5.077e-01  6.298e-02   8.061 7.54e-16 ***
## Age               7.347e-02  2.981e-03  24.645  < 2e-16 ***
## Tenure           -2.062e-02  1.089e-02  -1.893   0.0583 .
## Balance           2.756e-06  5.922e-07   4.654 3.25e-06 ***
## NumOfProducts    -9.113e-02  5.492e-02  -1.659   0.0970 .
## IsActiveMemberNo  1.002e+00  6.624e-02  15.135  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7540.2  on 7471  degrees of freedom
## Residual deviance: 6409.6  on 7463  degrees of freedom
## AIC: 6427.6
##
## Number of Fisher Scoring iterations: 5
```

Since this model does not seem to have any apparent discrepancies or issues, we use this as our final validation model to predict the churn rate on our training and test sets.

2.2.3 Cross Validation

We set the default threshold value as 0.5.

```
Lmodel <- model3
train_prob <- predict(Lmodel, data = train, type = "response")
test_prob <- predict(Lmodel, newdata = test, type = "response")
```

```r
train_pred <- factor(ifelse(train_prob >= 0.5, "Yes", "No"))
train_actual <- factor(ifelse(train$Exited == 1, "Yes", "No"))
test_pred <- factor(ifelse(test_prob >= 0.5, "Yes", "No"))
test_actual <- factor(ifelse(test$Exited == 1, "Yes", "No"))
```

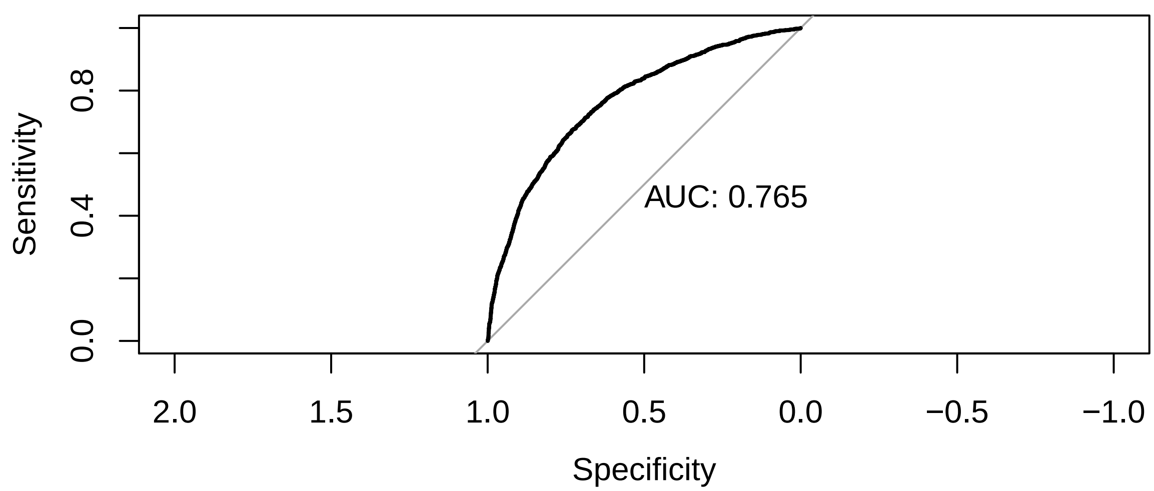
Now, we compute the confusion matrix and ROC for both training and test sets.

The Training Set

```r
confusionMatrix(data = train_pred, reference = train_actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  5756 1189
##        Yes  199  328
##
##                Accuracy : 0.8142
##                  95% CI : (0.8052, 0.823)
##     No Information Rate : 0.797
##     P-Value [Acc > NIR] : 9.451e-05
##
##                   Kappa : 0.2415
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9666
##             Specificity : 0.2162
##          Pos Pred Value : 0.8288
##          Neg Pred Value : 0.6224
##              Prevalence : 0.7970
##          Detection Rate : 0.7703
##    Detection Prevalence : 0.9295
##       Balanced Accuracy : 0.5914
##
##        'Positive' Class : No
##
```

```r
roc <- roc(train$Exited, train_prob, plot= TRUE, print.auc=TRUE)
```

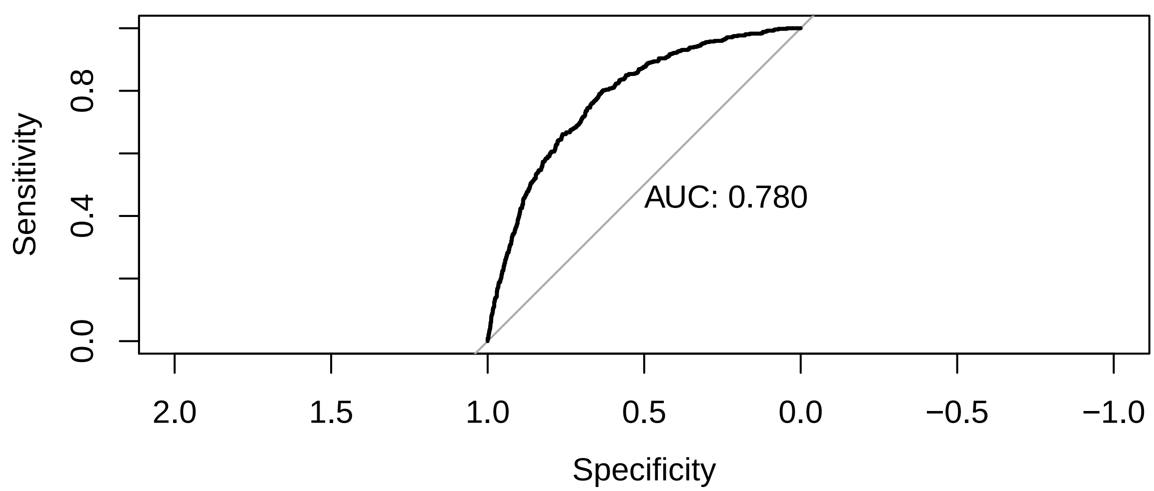


The Test Set

```
confusionMatrix(data = test_pred, reference = test_actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1937  425
##        Yes   71   95
##
##                Accuracy : 0.8038
##                  95% CI : (0.7878, 0.8191)
##     No Information Rate : 0.7943
##     P-Value [Acc > NIR] : 0.1234
##
##                   Kappa : 0.197
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9646
##             Specificity : 0.1827
##          Pos Pred Value : 0.8201
##          Neg Pred Value : 0.5723
##              Prevalence : 0.7943
##          Detection Rate : 0.7662
##    Detection Prevalence : 0.9343
##       Balanced Accuracy : 0.5737
##
##        'Positive' Class : No
##
```

```
roc <- roc(test$Exited, test_prob, plot= TRUE, print.auc=TRUE)
```



Therefore we get the following table of results:

	Training Set	Test Set
Accuracy	0.8142	0.8038
Specificity	0.2162	0.1827
Sensitivity	0.9666	0.9646
AUC Value	0.7650	0.7800

We then proceed to find the optimal threshold point that maximises the specificity and sensitivity.

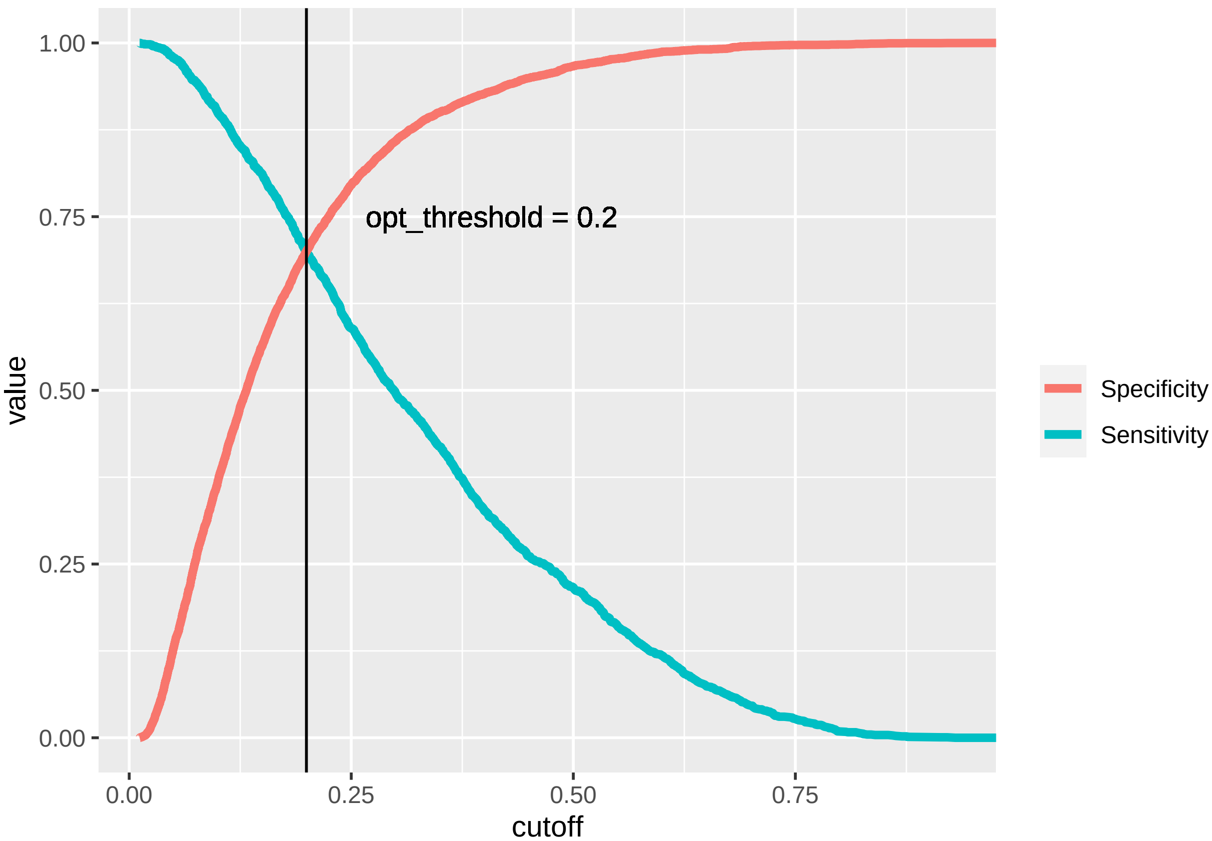
2.2.4 Finding the optimal cutoff

```
pred <- prediction(train_prob, train_actual)
perf <- performance(pred, "spec", "sens")

thres <- data.frame(threshold=perf@alpha.values[[1]], specificity=perf@x.values[[1]],
                    sensitivity= perf@y.values[[1]])

opt_thres <- thres[which.min(abs(thres$specificity-thres$sensitivity)),]
opt_thres %>% knitr::kable()
```

	threshold	specificity	sensitivity
2848	0.1995233	0.7000659	0.700084



The optimal cutoff is 0.2. So I use it as the threshold to predict churn on training and test sets.

Prediction on training set with threshold = 0.2:

```
train_pred_c <- factor(ifelse(train_prob >= 0.2, "Yes", "No"))
confusionMatrix(data = train_pred_c, reference = train_actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  4176  457
##        Yes 1779 1060
##
```

```
##                Accuracy : 0.7007
##                  95% CI : (0.6902, 0.7111)
##     No Information Rate : 0.797
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.302
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7013
##             Specificity : 0.6987
##          Pos Pred Value : 0.9014
##          Neg Pred Value : 0.3734
##              Prevalence : 0.7970
##          Detection Rate : 0.5589
##    Detection Prevalence : 0.6200
##       Balanced Accuracy : 0.7000
##
##        'Positive' Class : No
##
```

Prediction on test set with threshold = 0.2:

```
test_prob <- predict(Lmodel, newdata = test, type = "response")
test_pred_c <- factor(ifelse(test_prob >= 0.2, "Yes", "No"))
confusionMatrix(data = test_pred_c, reference = test_actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1413  156
##        Yes  595  364
##
##                Accuracy : 0.7029
##                  95% CI : (0.6847, 0.7207)
##     No Information Rate : 0.7943
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3075
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7037
##             Specificity : 0.7000
##          Pos Pred Value : 0.9006
##          Neg Pred Value : 0.3796
##              Prevalence : 0.7943
##          Detection Rate : 0.5589
##    Detection Prevalence : 0.6206
##       Balanced Accuracy : 0.7018
##
##        'Positive' Class : No
##
```

For the training set, the Accuracy is 0.70, and the Sensitivity and Specificity are both about 0.70. For the test set, the Accuracy is 0.70, and the Sensitivity and Specificity are both about 0.70 as well! Overall, this model with adjusted cutoff works well.

2.2.5 Summary for Logistic Regression Model

The final Logistic Regression Model (with threshold = 0.5) has Accuracy of 0.70 and the AUC is 0.78. Based on the P values for variables, `GeographyGermany`, `Tenure` and `NumOfProducts` have more significant influence on predicting churn.

2.3 Classification Tree Model

Decision Trees are a class of very powerful Machine Learning model cable of achieving high accuracy in many tasks while being highly interpretable. What makes decision trees special in the realm of ML models is really their clarity of information representation. The "knowledge" learned by a Classification Tree through training is directly formulated into a hierarchical structure. This structure holds and displays the knowledge in such a way that it can easily be understood.

2.3.1 Data Preparation

Classification Tree models can handle categorical variables without one-hot encoding them, and one-hot encoding will degrade tree-model performance. Thus, we re-prepare the data for Classification Tree and random forest models. We kept the "bank_clean" data before we do logistic regression and change the character variables to factors. Here's the final dataset we use for training classification tree models.

```
banktree <- bank_clean
banktree <- banktree %>%
  mutate_if(is.character, as.factor)
str(banktree)
```

```
## tibble [10,000 x 12] (S3: tbl_df/tbl/data.frame)
##  $ RowNumber      : num [1:10000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ CreditScore    : num [1:10000] 619 608 502 699 850 645 822 376 501 684 ...
##  $ Geography      : Factor w/ 3 levels "France","Germany",..: 1 3 1 1 3 3 1 2 1 1 ...
##  $ Gender         : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 2 2 1 2 2 ...
##  $ Age            : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:10000] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
##  $ HasCrCard      : num [1:10000] 1 0 1 0 1 1 1 1 0 1 ...
##  $ IsActiveMember : num [1:10000] 1 1 0 0 1 0 1 0 1 1 ...
##  $ EstimatedSalary: num [1:10000] 101349 112543 113932 93827 79084 ...
##  $ Exited         : num [1:10000] 1 0 1 0 0 1 0 1 0 0 ...
```

Split the data into training and test sets.

```
set.seed(818)
tree <- sample(0:1, size= nrow(banktree), prob = c(0.75,0.25), replace = TRUE)
traintree <- banktree[tree == 0, ]
testtree <- banktree[tree == 1, ]
```

2.3.2 Train Model1

First, we use all variables to build the model_tree1.

```
model_tree1 <- rpart(formula = Exited ~., data = traintree,
                     method = "class", parms = list(split = "gini"))
```

2.3.3 Cross Validation

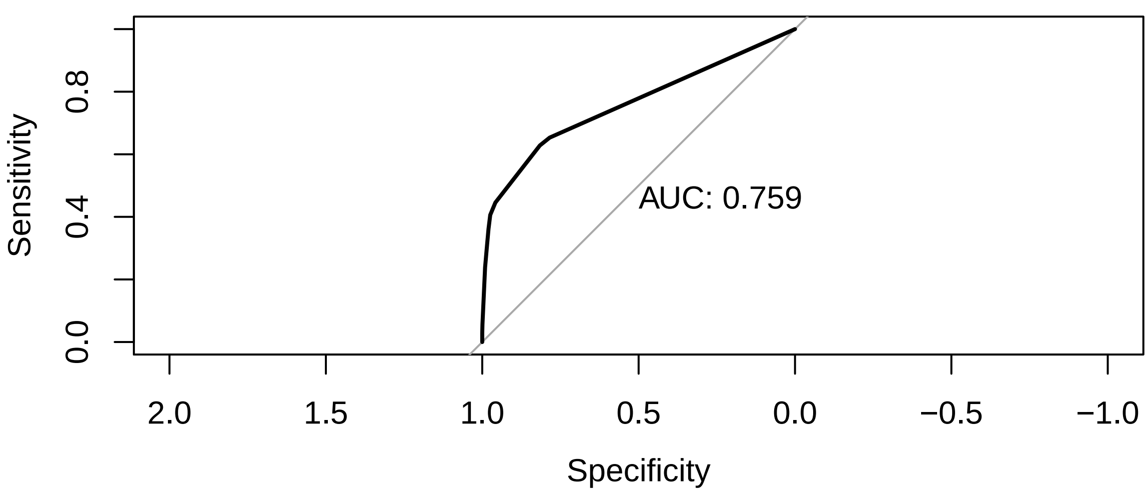
```r
traintree_pred1 <- predict(model_tree1, data = traintree, type = "class")
traintree_prob1 <- predict(model_tree1, data = traintree, type = "prob")
testtree_pred1 <- predict(model_tree1, newdata= testtree, type = "class")
testtree_prob1 <- predict(model_tree1, newdata = testtree, type = "prob")
```

For the Training Set

```r
confusionMatrix(data = as.factor(traintree_pred1), reference = as.factor(traintree$Exited))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5803  901
##          1  152  616
##
##                Accuracy : 0.8591
##                  95% CI : (0.851, 0.8669)
##     No Information Rate : 0.797
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4663
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9745
##             Specificity : 0.4061
##          Pos Pred Value : 0.8656
##          Neg Pred Value : 0.8021
##              Prevalence : 0.7970
##          Detection Rate : 0.7766
##    Detection Prevalence : 0.8972
##       Balanced Accuracy : 0.6903
##
##        'Positive' Class : 0
##
```

```r
traintree_actual <- ifelse(traintree$Exited==1,1,0)
roc <- roc(traintree_actual, traintree_prob1[,2], plot= TRUE, print.auc=TRUE)
```

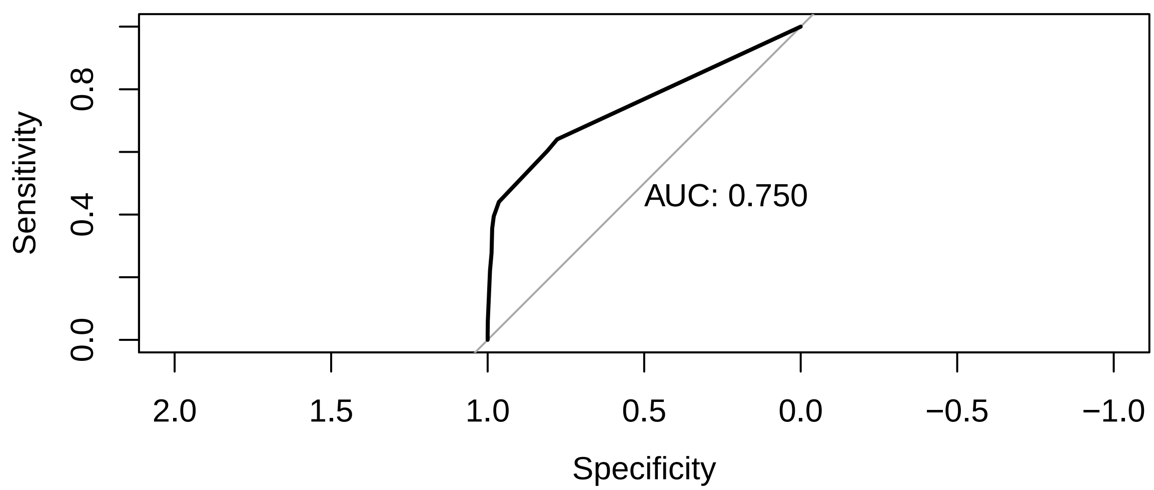


For the Test Set

```
confusionMatrix(data = as.factor(testtree_pred1), reference = as.factor(testtree$Exited))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1969  315
##          1   39  205
##
##                Accuracy : 0.86
##                  95% CI : (0.8458, 0.8733)
##     No Information Rate : 0.7943
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4666
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9806
##             Specificity : 0.3942
##          Pos Pred Value : 0.8621
##          Neg Pred Value : 0.8402
##              Prevalence : 0.7943
##          Detection Rate : 0.7789
##    Detection Prevalence : 0.9035
##       Balanced Accuracy : 0.6874
##
##        'Positive' Class : 0
##
```

```
testtree_actual <- ifelse(testtree$Exited == 1, 1,0)
roc <- roc(testtree_actual, testtree_prob1[,2], plot = TRUE, print.auc = TRUE)
```



Hence, we get the following table of results:

	Training Set	Test Set
Accuracy	0.859	0.860
Specificity	0.406	0.394
Sensitivity	0.975	0.981
AUC Value	0.759	0.750

Since each of the variables have negligible correlation, it is unlikely to affect the performance of the Classification Tree model. So we keep the first model as our final model.

2.3.4 Summary for Classification Tree Model

The final Classification Tree model has Accuracy of **0.86** and AUC of **0.75** for the test set. It performs better than the logistic regression model, which had an Accuracy of **0.70** and AUC of **0.78** for the test set.

2.4 Random Forest

Random forest is an ensemble tool which takes a subset of observations and a subset of variables to build a decision trees. It builds multiple such Classification Tree and amalgamate them together to get a more accurate and stable prediction. This is direct consequence of the fact that by maximum voting from a panel of independent judges, we get the final prediction better than the best judge.

2.4.1 Data Preparation

We use the same data prepared for Classification Tree models.

2.4.2 Train Model

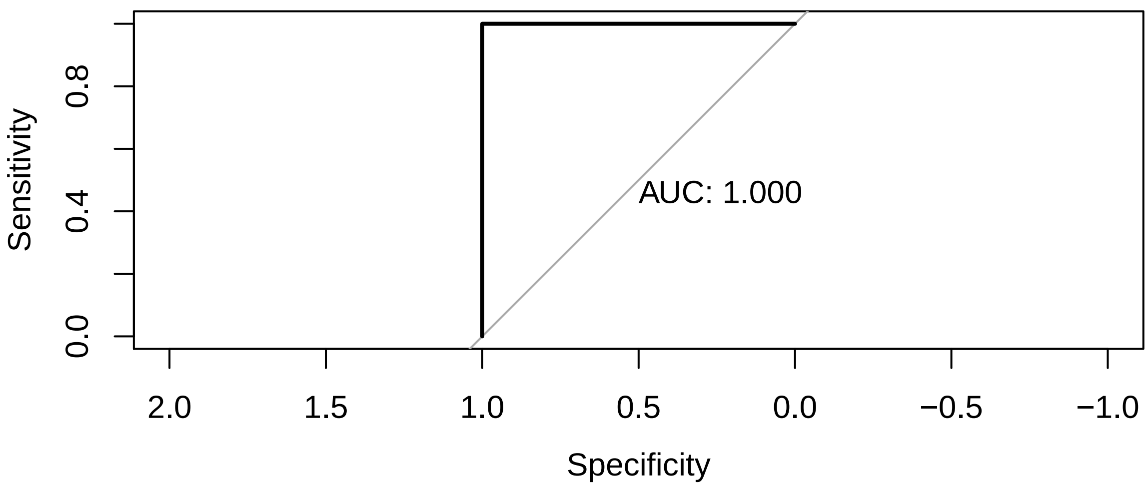
```
##
## Call:
##  randomForest(formula = as.factor(Exited) ~ ., data = traintree)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.57%
## Confusion matrix:
##      0   1 class.error
## 0 5750 205  0.03442485
## 1  809 708  0.53328939
```

2.4.3 Cross Validation

For the Training Set:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5955    0
##          1    0 1517
##
##                Accuracy : 1
##                  95% CI : (0.9995, 1)
##     No Information Rate : 0.797
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

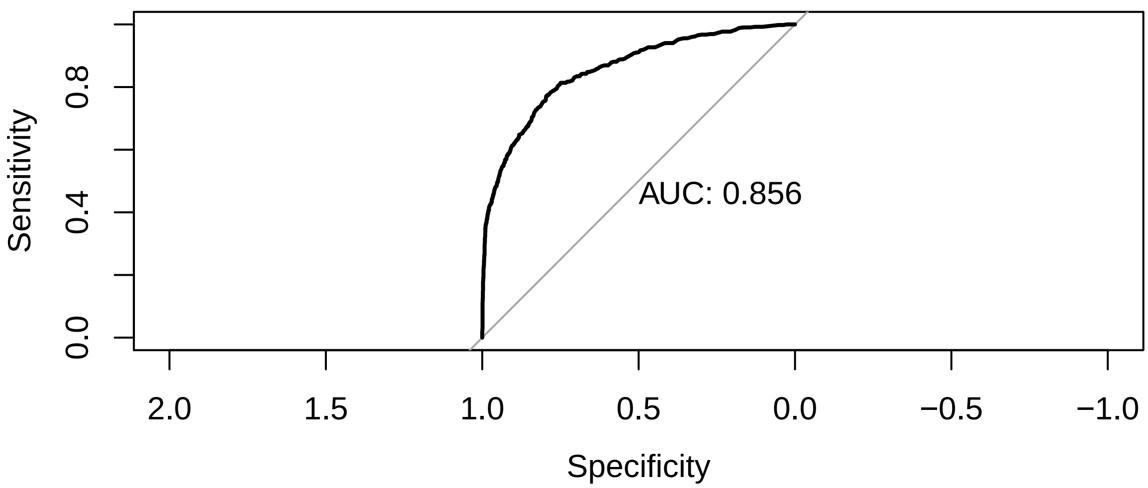
```
##                  Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##            Sensitivity : 1.000
##            Specificity : 1.000
##         Pos Pred Value : 1.000
##         Neg Pred Value : 1.000
##             Prevalence : 0.797
##         Detection Rate : 0.797
##   Detection Prevalence : 0.797
##      Balanced Accuracy : 1.000
##
##       'Positive' Class : 0
##
```



For the Test Set:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1939  285
##          1   69  235
##
##                Accuracy : 0.86
##                  95% CI : (0.8458, 0.8733)
##     No Information Rate : 0.7943
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4935
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9656
##             Specificity : 0.4519
##          Pos Pred Value : 0.8719
##          Neg Pred Value : 0.7730
##              Prevalence : 0.7943
```

```
##          Detection Rate : 0.7670
##    Detection Prevalence : 0.8797
##       Balanced Accuracy : 0.7088
##
##        'Positive' Class : 0
##
```



Hence, we get the following table of results:

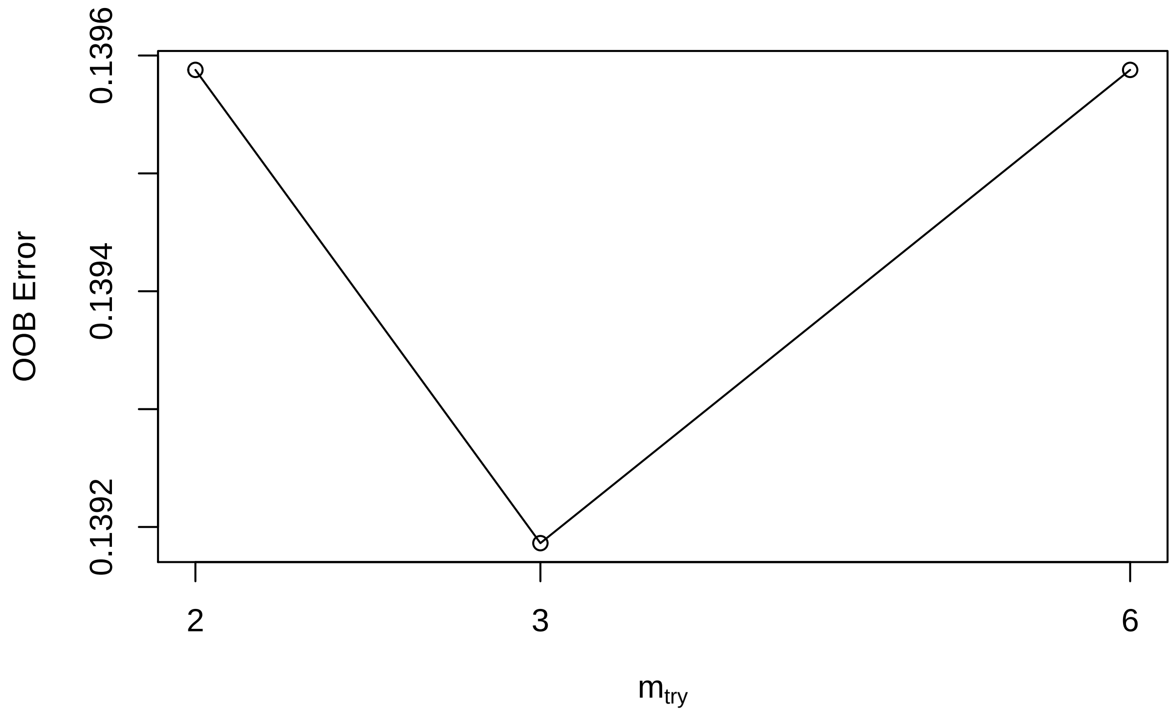
	Training Set	Test Set
Accuracy	1	0.860
Specificity	1	0.452
Sensitivity	1	0.966
AUC Value	1	0.856

2.4.4 Tuning

2.4.4.1 Tuning mtry with tuneRF

```
set.seed(818)
modelrf2 <- tuneRF(x = subset(traintree, select = -Exited), y = as.factor(traintree$Exited), ntreeTry =
```

```
## mtry = 3  OOB error = 13.92%
## Searching left ...
## mtry = 2     OOB error = 13.96%
## -0.002884615 0.05
## Searching right ...
## mtry = 6     OOB error = 13.96%
## -0.002884615 0.05
```




```
print(modelrf2)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1])
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.53%
## Confusion matrix:
##      0   1 class.error
## 0 5749 206  0.03459278
## 1  805 712  0.53065260
```

When mtry = 3, OOB decreases from 13.73% to 13.70%; when mtry = 6, OOB then increases to 13.88%.

2.4.4.2 Grid Search based on OOB error

We first establish a list of possible values for mtry, nodesize and sampsize.

```
mtry <- seq(2, ncol(traintree) * 0.8, 2)
nodesize <- seq(3, 8, 2)
sampsize <- nrow(traintree) * c(0.7, 0.8)
hyper_grid <- expand.grid(mtry = mtry, nodesize = nodesize, sampsize = sampsize)
```

Then, we create a loop to find the combination with the optimal 'oob err'.

```
oob_err <- c()
for (i in 1:nrow(hyper_grid)) {
  model <- randomForest(formula = as.factor(Exited) ~ .,
                        data = traintree,
                        mtry = hyper_grid$mtry[i],
                        nodesize = hyper_grid$nodesize[i],
```

```r
                        sampsize = hyper_grid$sampsize[i])
  oob_err[i] <- model$err.rate[nrow(model$err.rate), "OOB"]
}

opt_i <- which.min(oob_err)
print(hyper_grid[opt_i,])
```

```
##    mtry nodesize sampsize
## 22    4        7   5977.6
```

The optimal hyperparameters are mtry = 4, nodesize = 5, sampsize = 5230.4.

2.4.5 Train model 2 with optimal hyperparameters.

```r
set.seed(802)
modelrf3 <- randomForest(formula = as.factor(Exited) ~., data = traintree, mtry = 4, nodesize = 5, samps
print(modelrf3)
```

```
##
## Call:
##  randomForest(formula = as.factor(Exited) ~ ., data = traintree,      mtry = 4, nodesize = 5, sampsi
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 13.6%
## Confusion matrix:
##      0   1 class.error
## 0 5746 209  0.03509656
## 1  807 710  0.53197100
```

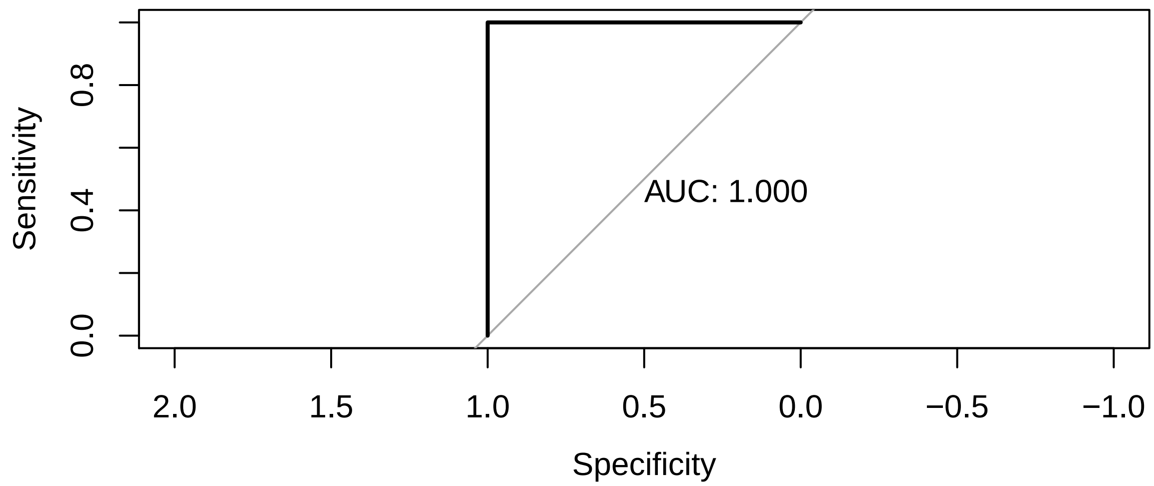
OOB of modelrf3 decreases a little bit to 13.6% with the optimal combination. The OOB of modelrf2 is 13.78%. So we will use modelrf3 as the final random forest model.

```r
trainrf_pred2 <- predict(modelrf2, traintree, type = "class")
trainrf_prob2 <- predict(modelrf2, traintree, type = "prob")
testrf_pred2 <- predict(modelrf2, newdata = testtree, type = "class")
testrf_prob2 <- predict(modelrf2, newdata = testtree, type = "prob")
```

For the Training Set:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5955    0
##          1    0 1517
##
##                Accuracy : 1
##                  95% CI : (0.9995, 1)
##     No Information Rate : 0.797
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.000
```

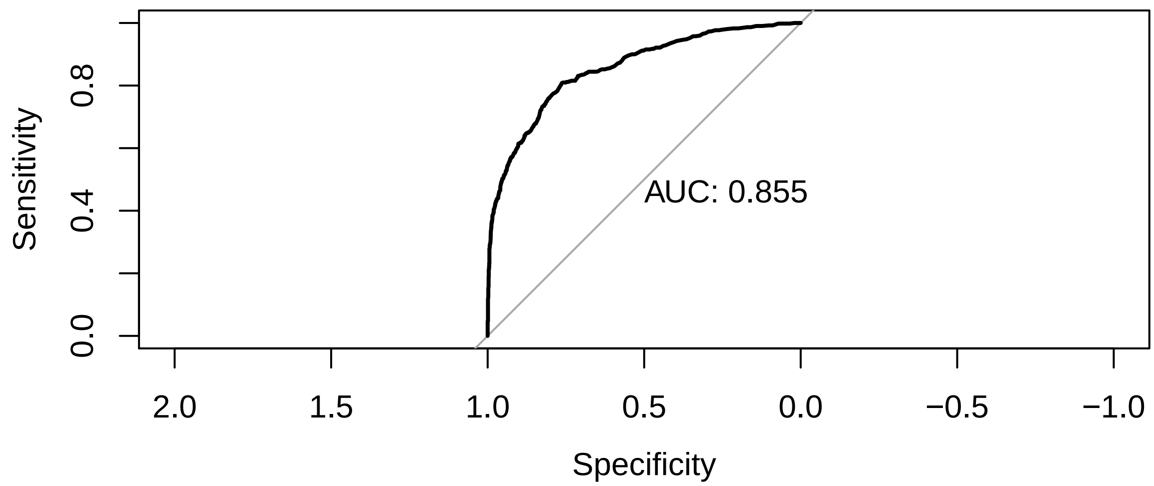
```
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 1.000
##              Prevalence : 0.797
##          Detection Rate : 0.797
##    Detection Prevalence : 0.797
##       Balanced Accuracy : 1.000
##
##        'Positive' Class : 0
##
```



For the Test Set:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1937  284
##          1   71  236
##
##                Accuracy : 0.8596
##                  95% CI : (0.8454, 0.8729)
##     No Information Rate : 0.7943
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4934
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9646
##             Specificity : 0.4538
##          Pos Pred Value : 0.8721
##          Neg Pred Value : 0.7687
##              Prevalence : 0.7943
##          Detection Rate : 0.7662
##    Detection Prevalence : 0.8786
##       Balanced Accuracy : 0.7092
##
##        'Positive' Class : 0
```

##



Hence, we get the following table of results:

	Training Set	Test Set
Accuracy	1	0.859
Specificity	1	0.452
Sensitivity	1	0.965
AUC Value	1	0.855

2.4.6 Summary for Random Forest Model

The final random forest model has the Accuracy of 0.859 and AUC of 0.855 for the test set; the accuracy is higher than the Logistic Regression Model but really close to the Classification Tree Model. However, the Random Forest model has the highest AUC value among all 3 models.

3 Results

We can summarise the results obtained through all the models using the following table of results and the comparison of ROC and AUC for Logistic Regression, Classification Tree and Random Forest Models.

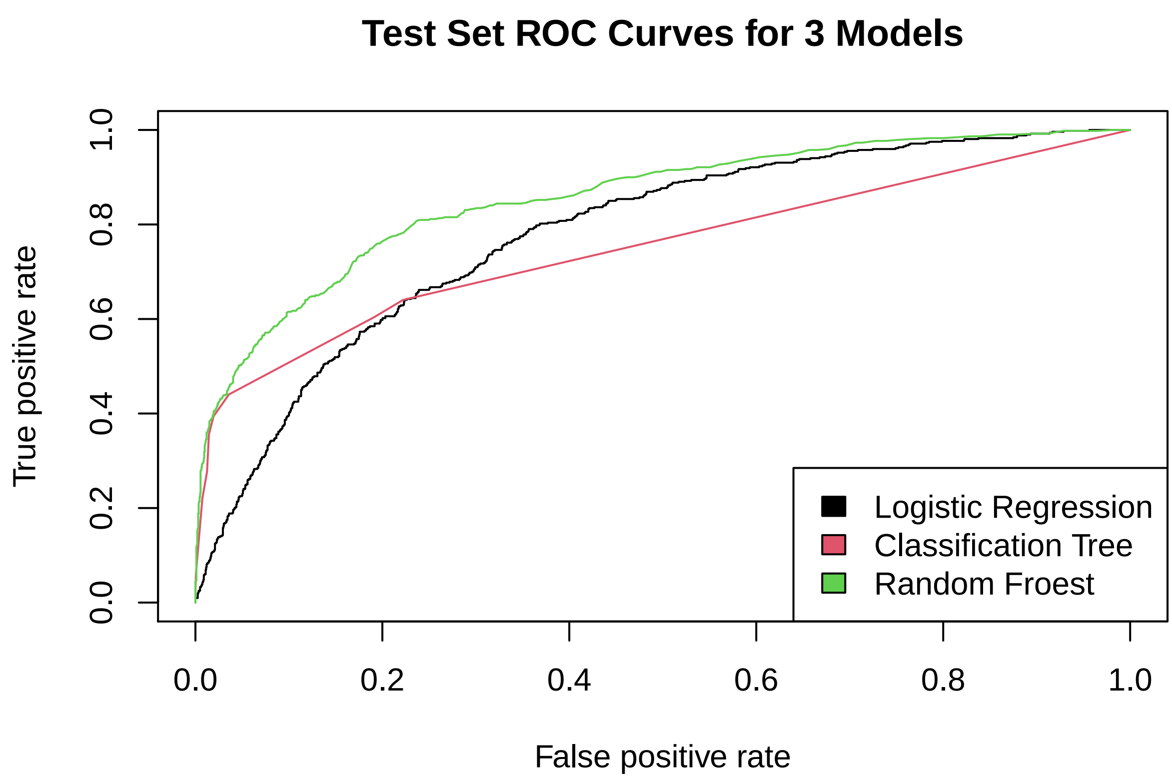
3.1 Table of Results

	Logistic Regression	Classification Tree	Random Forest
Accuracy	0.703	0.860	0.859
Specificity	0.700	0.394	0.452
Sensitivity	0.704	0.981	0.965
AUC Value	0.780	0.750	0.855

3.2 Comparison of ROC and AUC for Logistic Regression, Classification Tree and Random Forest models

```
preds_list <- list(test_prob, testtree_prob1[,2],testrf_prob2[,2])
m <- length(preds_list)
actuals_list <- rep(list(testtree$Exited), m)

pred <- prediction(preds_list, actuals_list)
rocs <- performance(pred, "tpr", "fpr")
plot(rocs, col = as.list(1:m), main = "Test Set ROC Curves for 3 Models")
legend(x = "bottomright",
       legend = c("Logistic Regression", "Classification Tree", "Random Froest"),
       fill = 1:m)
```



4 Discussion

We will now describe the metrics that we will compare in this section.

Accuracy is our starting point. It is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage.

Sensitivity is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Recall or the True Positive Rate. Sensitivity can be thought of as a measure of a classifiers completeness. A low sensitivity indicates many False Negatives.

Specificity (also called the true negative rate) measures the proportion of negatives which are correctly identified as such, and is complementary to the false positive rate. Specificity is also the number of true negatives divided by the sum of true negatives and false positives.

ROC (Receiver Operator Characteristic Curve) can help in deciding the best threshold value. It is generated by plotting the True Positive Rate against the False Positive Rate.

AUC stands for Area under the curve. AUC gives the rate of successful classification by the logistic model. The AUC makes it easy to compare the ROC curve of one model to another.

From the summary of results in the previous section it is clear that the Classification Tree Model has the greatest accuracy (0.860), followed by Logistic Regression having the greatest specificity (0.700), Classification Tree with the highest sensitivity (0.981) and Random Forest having the greatest AUC value (0.855).

5 Conclusion

This paper treats the Bank Customer Churn Analysis as a user classification problem. In this report we investigated several machine learning model and we selected the optimal model by selecting a high accuracy level combinated with a low rate of false-negatives (high sensitivity).

The Random Forest model had the optimal results for Accuracy (0.859), Sensitivity (0.965) and AUC value (0.855).

The analysis can also be further extended by exploring into other possible algorithms and models such as the Naive Bayes Model, KNN Model and Neural Networks.

6 Appendix - Environment

```
## [1] "Operating System:"

##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.0
## year           2020
## month          04
## day            24
## svn rev        78286
## language       R
## version.string R version 4.0.0 (2020-04-24)
## nickname       Arbor Day
```