BCC_Prediction

Dipanta

2023-08-24

Problem Statement

We got the dataset of a U.S. bank customer for getting the information that , this particular customer will leave bank or not. Bases upon independent feature we have to predict the customer will exited or not.

Importing Libraries

```
library(tidyverse)
library(corrplot)
library(cowplot)
library(caret)
library(tibble)
library(car)
library(caTools)
library(knitr)
library(e1071)
library(randomForest)
library(xgboost)
```

Preprocessing

```
df = read.csv("Churn_Modelling.csv")
```

```
head(df)
```

```
##   RowNumber CustomerId  Surname CreditScore Geography Gender Age Tenure
## 1         1   15634602 Hargrave         619    France Female  42      2
## 2         2   15647311     Hill         608     Spain Female  41      1
## 3         3   15619304     Onio         502    France Female  42      8
## 4         4   15701354     Boni         699    France Female  39      1
## 5         5   15737888 Mitchell         850     Spain Female  43      2
## 6         6   15574012      Chu         645     Spain   Male  44      8
##     Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited
## 1      0.00             1         1              1       101348.88      1
## 2  83807.86             1         0              1       112542.58      0
```

```
## 3 159660.80             3          1              0        113931.57      1
## 4      0.00             2          0              0         93826.63      0
## 5 125510.82             1          1              1         79084.10      0
## 6 113755.78             2          1              0        149756.71      1
```

```r
summary(df)
```

```
##    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
```

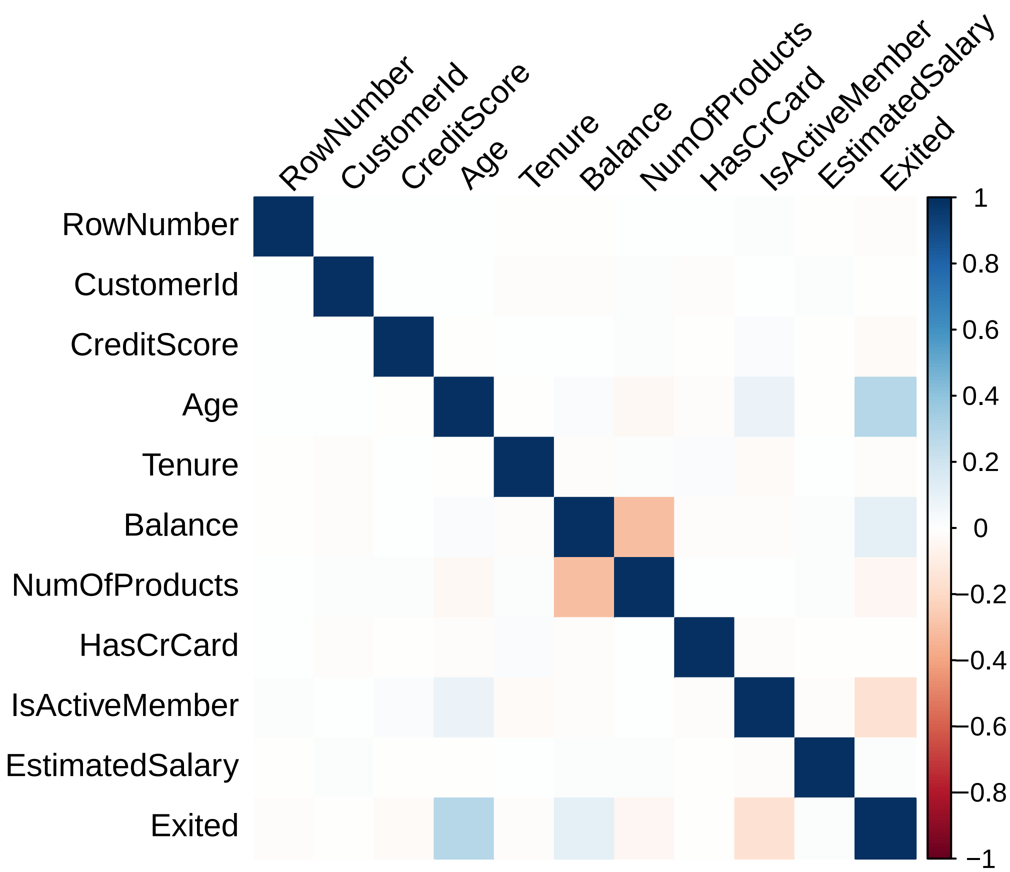
- There is no missing value.
- Exited is our dependent variable.

```r
# Calculate correlations
correlations = cor(df[, sapply(df, is.numeric) & names(df) != "Exited"], df$Exited) * 100

# Create a data frame for correlations
correlations_df = data.frame(Column = names(df)[sapply(df, is.numeric) & names(df) != "Exited"],
                             Correlation = correlations)

# Sort the data frame by correlation
correlations_df = correlations_df[order(-correlations_df$Correlation), ]

# Print the correlations table
print(correlations_df)
```

```
##                              Column Correlation
## Age                             Age  28.5323038
## Balance                     Balance  11.8532769
## EstimatedSalary EstimatedSalary   1.2096861
## CustomerId               CustomerId  -0.6247987
## HasCrCard                 HasCrCard  -0.7137766
## Tenure                       Tenure  -1.4000612
## RowNumber                 RowNumber  -1.6571371
## CreditScore             CreditScore  -2.7093540
## NumOfProducts         NumOfProducts  -4.7819865
## IsActiveMember       IsActiveMember -15.6128278
```

- IsActiveMember is has deep -ve correlation with a customer leaving (obvious). ie. Active/Regular customers are highily unlikely to leave.
- Age has a mild correlation with Exited. People with more age are likely to leave.
- Mild +ve correlation is observed for Balanced as well.

```r
# Calculate correlations
cor_matrix = cor(df[, sapply(df, is.numeric)])

# Create a correlation heatmap
corrplot(cor_matrix, method = "color", tl.col = "black", tl.srt = 45)
```



```r
head(df)
```

```
##   RowNumber CustomerId  Surname CreditScore Geography Gender Age Tenure
## 1         1   15634602 Hargrave         619    France Female  42      2
## 2         2   15647311     Hill         608     Spain Female  41      1
## 3         3   15619304     Onio         502    France Female  42      8
## 4         4   15701354     Boni         699    France Female  39      1
## 5         5   15737888 Mitchell         850     Spain Female  43      2
## 6         6   15574012      Chu         645     Spain   Male  44      8
##     Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited
## 1      0.00             1         1              1       101348.88      1
## 2  83807.86             1         0              1       112542.58      0
## 3 159660.80             3         1              0       113931.57      1
## 4      0.00             2         0              0        93826.63      0
## 5 125510.82             1         1              1        79084.10      0
## 6 113755.78             2         1              0       149756.71      1
```

Checking data imbalance

```r
# Group by Gender and calculate max credit score
max_credit_by_gender <- df %>%
  group_by(Gender) %>%
  summarize(max_credit = max(CreditScore))

print(max_credit_by_gender)
```

```
## # A tibble: 2 x 2
##   Gender max_credit
##   <chr>       <int>
## 1 Female        850
## 2 Male          850
```

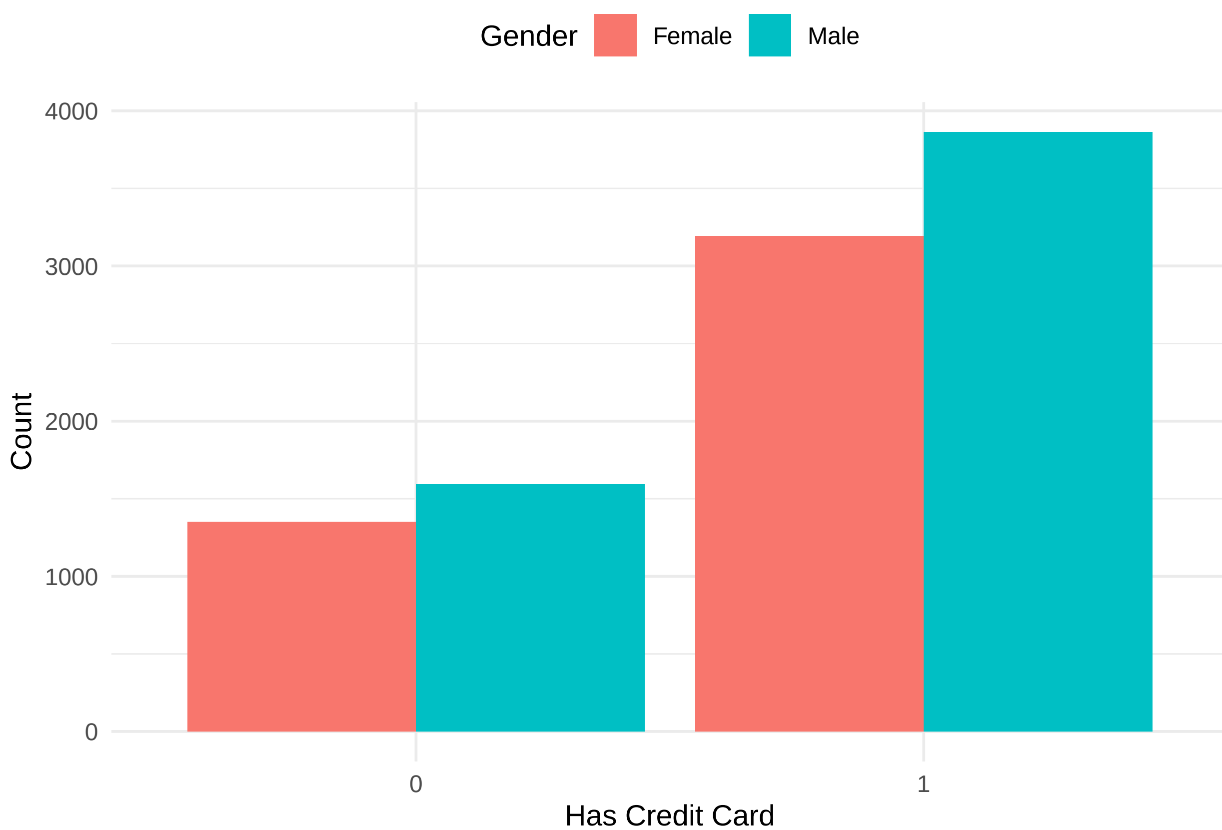
```r
# Group by Gender and HasCrCard, then count the occurrences
card_counts_by_gender <- df %>%
  group_by(Gender, HasCrCard) %>%
  summarize(count = n())
```

```
## 'summarise()' has grouped output by 'Gender'. You can override using the
## '.groups' argument.
```

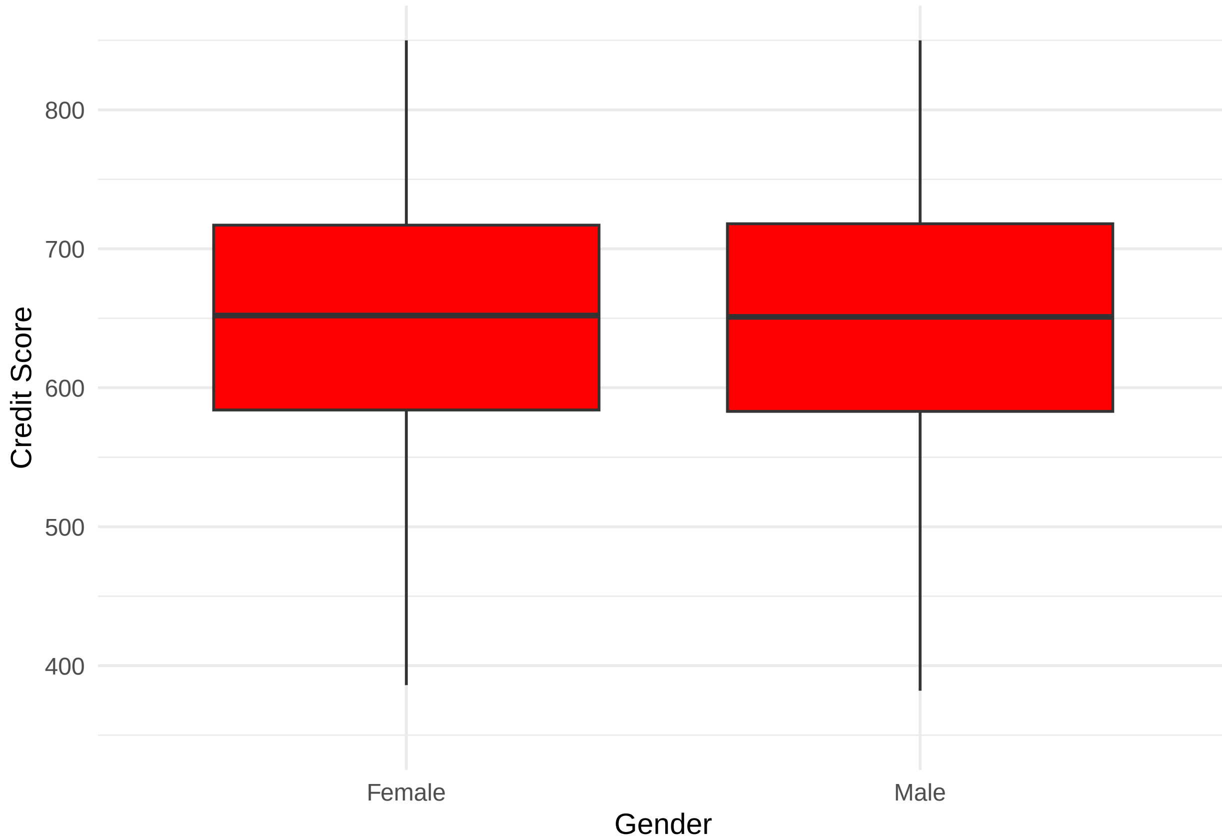
```r
print(card_counts_by_gender)
```

```
## # A tibble: 4 x 3
## # Groups:   Gender [2]
##   Gender HasCrCard count
##   <chr>      <int> <int>
## 1 Female         0  1351
## 2 Female         1  3192
## 3 Male           0  1594
## 4 Male           1  3863
```

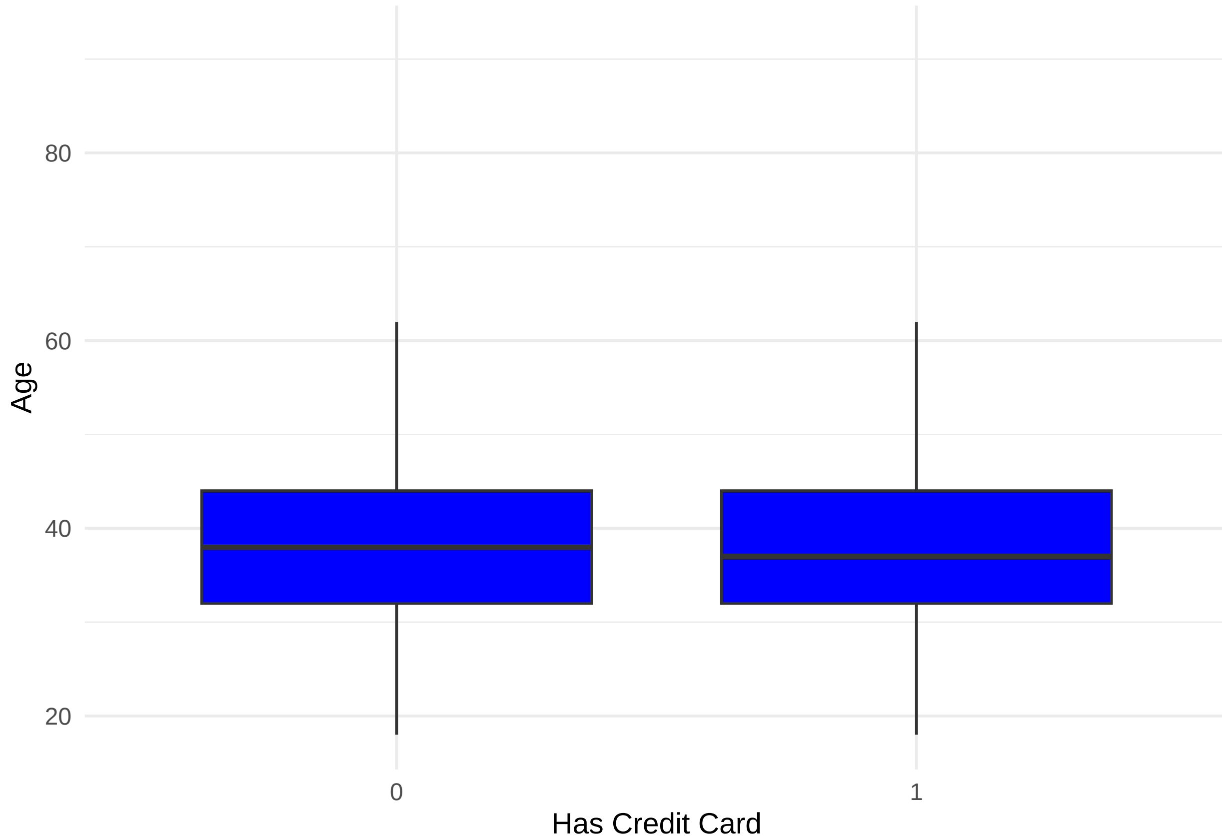
```r
# Create a bar plot
ggplot(card_counts_by_gender, aes(x = factor(HasCrCard), y = count, fill = factor(Gender))) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Has Credit Card", y = "Count", fill = "Gender") +
  scale_fill_discrete(name = "Gender") +
  theme_minimal() +
  theme(legend.position = "top")
```




```r
# Gender Vs Credit_score
ggplot(df, aes(x = factor(Gender), y = CreditScore)) +
  geom_boxplot(fill = "red", outlier.shape = NA) +
  labs(x = "Gender", y = "Credit Score") +
  theme_minimal()
```




```r
# Has_credit vs Age
ggplot(df, aes(x = factor(HasCrCard), y = Age)) +
  geom_boxplot(fill = "blue", outlier.shape = NA) +
  labs(x = "Has Credit Card", y = "Age") +
  theme_minimal()
```



Label Encoding

```
# Create label encoding dictionaries
gender_labels = c("Female" = 0, "Male" = 1)
geography_labels = c("France" = 0, "Germany" = 1, "Spain" = 2)

# Apply label encoding to the Gender column
df$Gender = gender_labels[df$Gender]

# Apply label encoding to the Geography column
df$Geography = geography_labels[df$Geography]
```

Feature selection

```
# Remove specified columns
columns_to_remove = c("RowNumber", "CustomerId", "Surname")
df = df[, !(names(df) %in% columns_to_remove)]
```

Checking VIF

```r
selected_vars <- c("CreditScore", "Gender", "Age", "Tenure", "Balance", "HasCrCard", "IsActiveMember",
model <- lm(df$Exited ~ ., data = df[, selected_vars])


# Calculate VIF
vif_values <- vif(model)

print(vif_values)
```

```
##     CreditScore          Gender             Age          Tenure         Balance
##        1.000841        1.001927        1.009790        1.001823        1.006468
##       HasCrCard  IsActiveMember EstimatedSalary       Geography
##        1.001169        1.009901        1.000570        1.005469
```

Split The Data set

```r
set.seed(123)

split = sample.split(df$Exited, SplitRatio = 0.8)

training_set = df[split, ]
test_set = df[!split, ]
```

```r
# Feature Scaling
training_set[-11] = scale(training_set[-11])
test_set[-11] = scale(test_set[-11])
```

Model Building

Logistic Regression

```r
# Fitting Logistic Regression to the Training set
classifier1 = glm(formula = Exited ~ .,
                  family = binomial,
                  data = training_set)
```

```r
# Predicting the Test set results
prob_pred = predict(classifier1, type = 'response', newdata = test_set[-11])
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

```r
# Making the Confusion Matrix
conf_matrix1 = table(test_set[, 11], y_pred > 0.5)
print(conf_matrix1)
```

```
##
##     FALSE TRUE
##   0  1547   46
##   1   326   81
```

Evalution matrics

```r
# Calculate evaluation metrics
accuracy <- sum(diag(conf_matrix1)) / sum(conf_matrix1)
precision <- conf_matrix1[2, 2] / sum(conf_matrix1[, 2])
recall <- conf_matrix1[2, 2] / sum(conf_matrix1[2, ])
f1_score <- 2 * (precision * recall) / (precision + recall)


# Create a data frame for the metrics
metrics_df <- data.frame(
  Metric = c("Precision", "Recall", "F1 Score", "Accuracy"),
  Value = c(precision, recall, f1_score, accuracy)
)

# Print the metrics table
kable(metrics_df, format = "html", caption = "Evaluation Metrics for log_reg")
```

Evaluation Metrics for log_reg

Metric

Value

Precision

0.6377953

Recall

0.1990172

F1 Score

0.3033708

Accuracy

0.8140000

SVM

```r
# Fitting SVM to the Training set
classifier2 = svm(formula = Exited ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')

# Predicting the Test set results
y_pred = predict(classifier2, newdata = test_set[-11])

# Making the Confusion Matrix
conf_matrix2 = table(test_set[, 11], y_pred)
print(conf_matrix2)
```

```
##    y_pred
##        0    1
##   0 1593    0
##   1  407    0
```

Naive_bayes

```r
# Fitting SVM to the Training set

classifier3 = naiveBayes(x = training_set[-11],
                        y = training_set$Exited)

# Predicting the Test set results
y_pred = predict(classifier3, newdata = test_set[-11])

# Making the Confusion Matrix
conf_matrix3 = table(test_set[, 11], y_pred)
print(conf_matrix3)
```

```
##    y_pred
##        0    1
##   0 1567   26
##   1  305  102
```

Evalution matrics

```r
# Calculate evaluation metrics
accuracy <- sum(diag(conf_matrix3)) / sum(conf_matrix3)
precision <- conf_matrix3[2, 2] / (conf_matrix3[2, 2] + conf_matrix3[1, 2])
recall <- conf_matrix3[2, 2] / sum(conf_matrix3[2, ])
f1_score <- 2 * (precision * recall) / (precision + recall)

# Create a data frame for the metrics
metrics_df <- data.frame(
  Metric = c("Precision", "Recall", "F1 Score", "Accuracy"),
  Value = c(precision, recall, f1_score, accuracy)
)

# Print the metrics table
kable(metrics_df, format = "html", caption = "Evaluation Metrics for Naive_Baiyes")
```

Evaluation Metrics for Naive_Baiyes

Metric

Value

Precision

0.7968750

Recall

0.2506143

F1 Score

0.3813084

Accuracy

0.8345000

XGBoost

```r
# Fitting XGBoost to the Training set
classifier5 = xgboost(data = as.matrix(training_set[-11]), label = training_set$Exited, nrounds = 10)
```

```
## [1]  train-rmse:0.419544
## [2]  train-rmse:0.372652
## [3]  train-rmse:0.344824
## [4]  train-rmse:0.329253
## [5]  train-rmse:0.318175
## [6]  train-rmse:0.311981
## [7]  train-rmse:0.308010
## [8]  train-rmse:0.305611
## [9]  train-rmse:0.303503
## [10] train-rmse:0.301551
```

```r
# Predicting the Test set results
y_pred = predict(classifier5, newdata = as.matrix(test_set[-11]))
y_pred = (y_pred >= 0.5)

# Making the Confusion Matrix
conf_matrix5 = table(test_set[, 11], y_pred)
print(conf_matrix5)
```

```
##    y_pred
##     FALSE TRUE
##   0  1531   62
##   1   208  199
```

Evalution matrics

```r
# Calculate evaluation metrics
accuracy <- sum(diag(conf_matrix5)) / sum(conf_matrix5)
precision <- conf_matrix5[2, 2] / (conf_matrix5[2, 2] + conf_matrix5[1, 2])
recall <- conf_matrix5[2, 2] / sum(conf_matrix5[2, ])
f1_score <- 2 * (precision * recall) / (precision + recall)

# Create a data frame for the metrics
metrics_df <- data.frame(
  Metric = c("Precision", "Recall", "F1 Score", "Accuracy"),
  Value = c(precision, recall, f1_score, accuracy)
)

# Print the metrics table
kable(metrics_df, format = "html", caption = "Evaluation Metrics for Random forest")
```

Evaluation Metrics for Random forest

Metric

Value

Precision

0.7624521

Recall

0.4889435

F1 Score

0.5958084

Accuracy

0.8650000

```
# Applying k-Fold Cross Validation

folds = createFolds(df$Exited, k = 10)
cv = lapply(folds, function(x) {
  training_fold = df[-x, ]
  test_fold = df[x, ]
  classifier = xgboost(data = as.matrix(training_fold[-11]), label = training_fold$Exited, nrounds = 10
  y_pred = predict(classifier, newdata = as.matrix(test_fold[-11]))
  y_pred = (y_pred >= 0.5)
  cm = table(test_fold[, 11], y_pred)
  accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])
  return(accuracy)
})
```

```
## [1]  train-rmse:0.418302
## [2]  train-rmse:0.370219
## [3]  train-rmse:0.342841
## [4]  train-rmse:0.327062
## [5]  train-rmse:0.317398
## [6]  train-rmse:0.311022
## [7]  train-rmse:0.306601
## [8]  train-rmse:0.304373
## [9]  train-rmse:0.301942
## [10] train-rmse:0.298806
## [1]  train-rmse:0.418300
## [2]  train-rmse:0.370277
## [3]  train-rmse:0.342184
## [4]  train-rmse:0.326322
## [5]  train-rmse:0.317118
## [6]  train-rmse:0.310666
## [7]  train-rmse:0.306873
## [8]  train-rmse:0.303867
## [9]  train-rmse:0.300304
## [10] train-rmse:0.298492
## [1]  train-rmse:0.419538
## [2]  train-rmse:0.372041
## [3]  train-rmse:0.345068
## [4]  train-rmse:0.329329
## [5]  train-rmse:0.320171
## [6]  train-rmse:0.314364
## [7]  train-rmse:0.309129
```

```
## [8]  train-rmse:0.305668
## [9]  train-rmse:0.303183
## [10] train-rmse:0.300869
## [1]  train-rmse:0.418897
## [2]  train-rmse:0.371622
## [3]  train-rmse:0.344170
## [4]  train-rmse:0.329286
## [5]  train-rmse:0.320223
## [6]  train-rmse:0.313573
## [7]  train-rmse:0.309450
## [8]  train-rmse:0.306447
## [9]  train-rmse:0.304177
## [10] train-rmse:0.302097
## [1]  train-rmse:0.418626
## [2]  train-rmse:0.371110
## [3]  train-rmse:0.343417
## [4]  train-rmse:0.327903
## [5]  train-rmse:0.318374
## [6]  train-rmse:0.311983
## [7]  train-rmse:0.308026
## [8]  train-rmse:0.305182
## [9]  train-rmse:0.302209
## [10] train-rmse:0.299338
## [1]  train-rmse:0.418375
## [2]  train-rmse:0.370270
## [3]  train-rmse:0.343276
## [4]  train-rmse:0.328045
## [5]  train-rmse:0.317545
## [6]  train-rmse:0.311344
## [7]  train-rmse:0.307834
## [8]  train-rmse:0.305095
## [9]  train-rmse:0.302895
## [10] train-rmse:0.301189
## [1]  train-rmse:0.418001
## [2]  train-rmse:0.370064
## [3]  train-rmse:0.342488
## [4]  train-rmse:0.327542
## [5]  train-rmse:0.317576
## [6]  train-rmse:0.311102
## [7]  train-rmse:0.306406
## [8]  train-rmse:0.301882
## [9]  train-rmse:0.300026
## [10] train-rmse:0.298625
## [1]  train-rmse:0.419506
## [2]  train-rmse:0.372092
## [3]  train-rmse:0.345222
## [4]  train-rmse:0.327999
## [5]  train-rmse:0.318880
## [6]  train-rmse:0.312934
## [7]  train-rmse:0.307536
## [8]  train-rmse:0.304414
## [9]  train-rmse:0.302761
## [10] train-rmse:0.300425
## [1]  train-rmse:0.418878
```

```
## [2]  train-rmse:0.371200
## [3]  train-rmse:0.343088
## [4]  train-rmse:0.327924
## [5]  train-rmse:0.317874
## [6]  train-rmse:0.312930
## [7]  train-rmse:0.307719
## [8]  train-rmse:0.305396
## [9]  train-rmse:0.301979
## [10] train-rmse:0.300018
## [1]  train-rmse:0.419283
## [2]  train-rmse:0.371922
## [3]  train-rmse:0.345234
## [4]  train-rmse:0.330544
## [5]  train-rmse:0.319466
## [6]  train-rmse:0.312206
## [7]  train-rmse:0.308465
## [8]  train-rmse:0.304685
## [9]  train-rmse:0.302070
## [10] train-rmse:0.300406
```

```r
accuracy = mean(as.numeric(cv))
print(accuracy)
```

```
## [1] 0.8592
```

Random Forest

```r
training_set$Exited <- factor(training_set$Exited, levels = c(0, 1))
test_set$Exited <- factor(test_set$Exited, levels = c(0, 1))


# Fitting Random Forest Classification to the Training set
set.seed(123)
classifier4 = randomForest(x = training_set[-11],
                          y = training_set$Exited,
                          ntree = 50)

# Predicting the Test set results
y_pred = predict(classifier4, newdata = test_set[-11])
```

```r
# Making the Confusion Matrix
conf_matrix4 = table(test_set[, 11], y_pred)
print(conf_matrix4)
```

```
##    y_pred
##        0    1
##   0 1538   55
##   1  207  200
```

Evalution matrics

```r
# Calculate evaluation metrics
accuracy <- sum(diag(conf_matrix4)) / sum(conf_matrix4)
precision <- conf_matrix4[2, 2] / (conf_matrix4[2, 2] + conf_matrix4[1, 2])
recall <- conf_matrix4[2, 2] / sum(conf_matrix4[2, ])
f1_score <- 2 * (precision * recall) / (precision + recall)

# Create a data frame for the metrics
metrics_df <- data.frame(
  Metric = c("Precision", "Recall", "F1 Score", "Accuracy"),
  Value = c(precision, recall, f1_score, accuracy)
)

# Print the metrics table
kable(metrics_df, format = "html", caption = "Evaluation Metrics for Random forest")
```

Evaluation Metrics for Random forest

Metric

Value

Precision

0.7843137

Recall

0.4914005

F1 Score

0.6042296

Accuracy

0.8690000

Conclusion

- After using the models we conclude that *XGBoost* and *RandomForest* are the best model for the problem.