XYZ Premium Convert Analysis

Rebecca Meyer, Narae Kang, Shubham Garg, Pranvi Setia, Yun-Chien Yen

2022-07-31

Introduction

Our team conducted data analysis to predict which people would be likely to convert from free users to premium subscribers in the next 6 month period, if they are targeted by our promotional campaign.

Import library

Our analysis was performed in Jupyter Lab - RStudio. The packages needed to preform the following analysis are shown below.

```
library(dplyr)
library(caret)
library(rpart)
library(class)
library(pROC)
library(e1071)
library(kknn)
library(FSelectorRcpp)
library(doParallel)
library(ROSE)
library(ggplot2)
library(GGally)
library(gridExtra)
library(rpart.plot)
```

We then uploaded the data using the read.csv() function. First, we set variables (e.g. 'male', 'good_country') as binary factor type and take out user id from the data as it represents the key in data without analytical meanings.

Data Processing

```
xyz <- read.csv("XYZData.csv")
xyz$adopter <- as.factor(xyz$adopter)
xyz <- xyz[2:27]
xyz_for_normalization <- xyz
xyz$male <- as.factor(xyz$male)
xyz$good_country <- as.factor(xyz$good_country)
```

Data Normalization and Train-test split

```
# Define function
normalize = function(x){
return ((x - min(x))/(max(x) - min(x)))}

# Data normalization
xyz_normalized = xyz_for_normalization %>% mutate_at(1:25, normalize)

# Split dataset into 70% training and 30% testing
train_rows = createDataPartition(y = xyz$adopter, p = 0.70, list = FALSE)
```

For KNN, we used normalized data

```
xyz_normalized_train = xyz_normalized[train_rows,]
xyz_normalized_test = xyz_normalized[-train_rows,]
```

For decision tree and Naive Bayes , we used the original data (without normalization)

```
xyz_train = xyz[train_rows,]
xyz_test = xyz[-train_rows,]
```

Dealing with inbalanced data - oversampling

While exploring the data, we found that for the predictor adopter, adopter = 1 has 1540 (3.7%) and adopter = 0 has 40000 (96.3%) records. We oversampled the dataset for the adopter = 1 (premium user), to balance the dataset to get better predictions for adopted = 1 class (premium users).

```
table(xyz_train$adopter)
```

```
##
##     0     1
## 28000  1078
```

```
# Creating balanced dataset for Naive Bayes and Decision Tree
xyz_train_balanced <-
  ovun.sample(adopter ~ ., data = xyz_train, method = "over",
              N = 50000, seed = 1)$data
table(xyz_train_balanced$adopter)
```

```
##
##     0     1
## 28000 22000
```

```
# Creating balanced dataset for KNN
xyz_train_balanced_normalized <- ovun.sample(adopter ~ ., data = xyz_normalized_train, method = "over",
```

Feature Selection

We used the filter approach to find the most important features. We used the information_gain() function in the FselectorRcpp package. The function outputs information gains for each feature. Then, we used the cut_attrs() function to select the top k features. We tried multiple values of K and chose K = 8, to optimize the complexity of the model and the results obtained from it.

```
# Selecting the top 8 feautures
IG = information_gain(adopter ~ ., data = xyz_train_balanced)
topK = cut_attrs(IG, k = 8)

#Examining the top 8 features
topK
```

```
## [1] "songsListened"         "delta_avg_friend_age"  "avg_friend_age"
## [4] "lovedTracks"           "delta_songsListened"   "friend_cnt"
## [7] "subscriber_friend_cnt" "delta_lovedTracks"
```

```
# Creating new training and testing datasets consisting of the top 8 features
xyz_train_top8_balanced = xyz_train_balanced %>% select(topK, adopter)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(topK)` instead of `topK` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
xyz_train_top8_balanced_normalized = xyz_train_balanced_normalized %>% select(topK, adopter)
xyz_test_top8 = xyz_test %>% select(topK, adopter)
xyz_test_top8_normalized = xyz_normalized_test %>% select(topK, adopter)
```

Modeling

For the model building, we tried three methods : (1) KNN (2) Naive Bayes (3) Decision Tree

KNN

First, we looped through k = 3 to k= 20 and found the best k clusters.

```
AUC_curve <- c()
prec <- c()
rec <- c()
acc <- c()
F1 <- c()

for (n in 3:20) {
  # knn model
  pred_knn = knn(train = xyz_train_top8_balanced_normalized[,1:8],
                 test = xyz_test_top8_normalized[,1:8],
                 cl = xyz_train_top8_balanced_normalized$adopter,
                 k = n,
                 prob = TRUE)

 cm = confusionMatrix(pred_knn,
```

```
                  xyz_test_top8_normalized$adopter,
                  positive = "1",
                  mode = "prec_recall")

 prec[n-2] = cm[["byClass"]][["Precision"]]
 rec[n-2] = cm[["byClass"]][["Recall"]]
 acc[n-2] = cm[["overall"]][["Accuracy"]]
 F1[n-2] = cm[["byClass"]][["F1"]]

 model_knn = kknn(adopter ~ .,
                  train = xyz_train_top8_balanced_normalized,
                  test = xyz_test_top8_normalized,
                  k = n,
                  distance = 2)

 pred_prob_knn = model_knn$prob
  # caculate roc curve for each k
  roc_curve = roc(response=
                    ifelse(xyz_test_top8_normalized$adopter==
                             '1', 1, 0),
                  predictor = pred_prob_knn[,"1"])

  auc(roc_curve)

  AUC_curve[n-2] = auc(roc_curve)
}
```

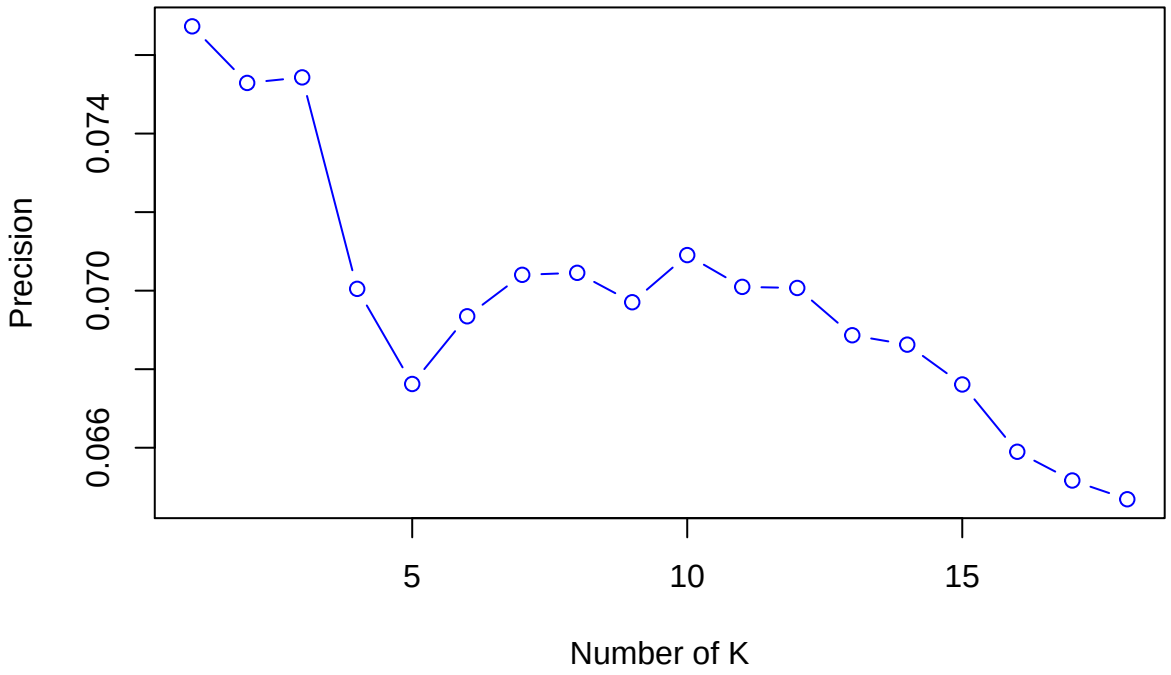
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
# Examine and plot the precision values for K = 3 to K = 20 for KNN..
prec
```

```
##  [1] 0.07673061 0.07528958 0.07543103 0.07004608 0.06762295 0.06934710
##  [7] 0.07040328 0.07045382 0.06970509 0.07090602 0.07009595 0.07006855
## [13] 0.06886447 0.06862515 0.06761078 0.06589749 0.06516562 0.06468830
```

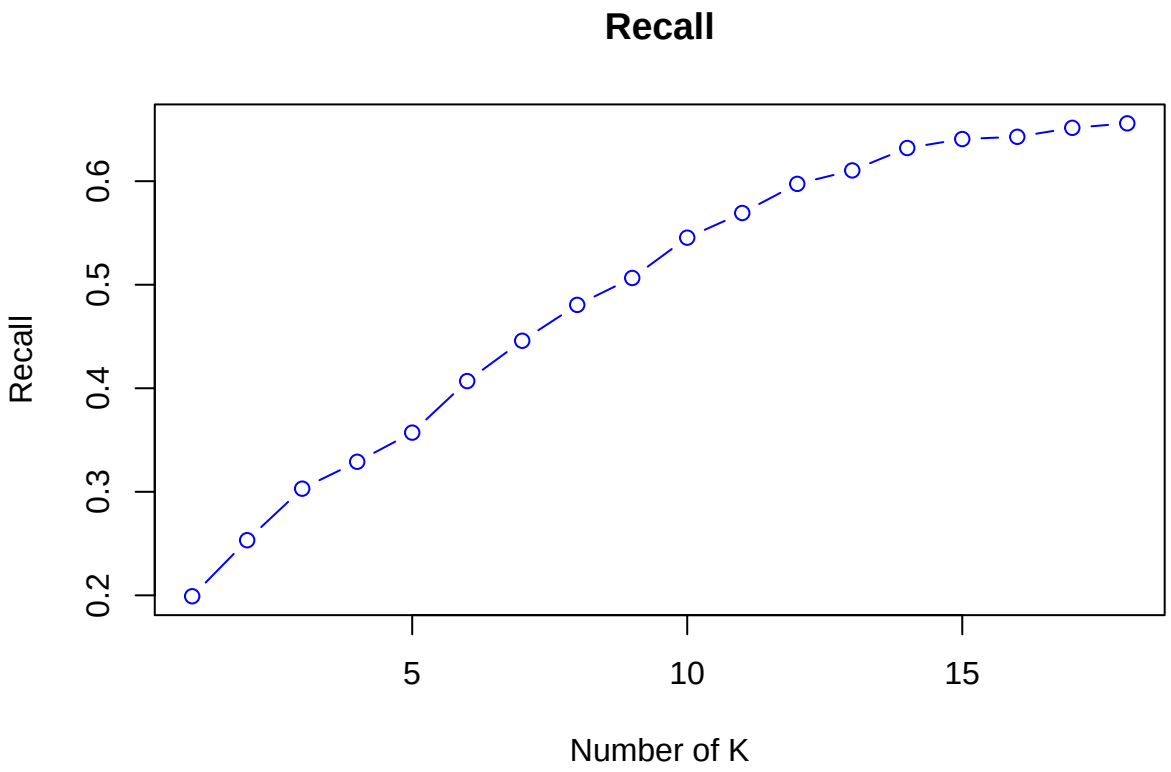
```
plot(1:18,
     prec,
     type = "b",
     main = "Precision",
     xlab = "Number of K", ylab = "Precision",
     col = 'blue')
```




```
# Examine and plot the recall values
rec
```

```
##  [1] 0.1991342 0.2532468 0.3030303 0.3290043 0.3571429 0.4069264 0.4458874
##  [8] 0.4805195 0.5064935 0.5454545 0.5692641 0.5974026 0.6103896 0.6320346
## [15] 0.6406926 0.6428571 0.6515152 0.6558442
```

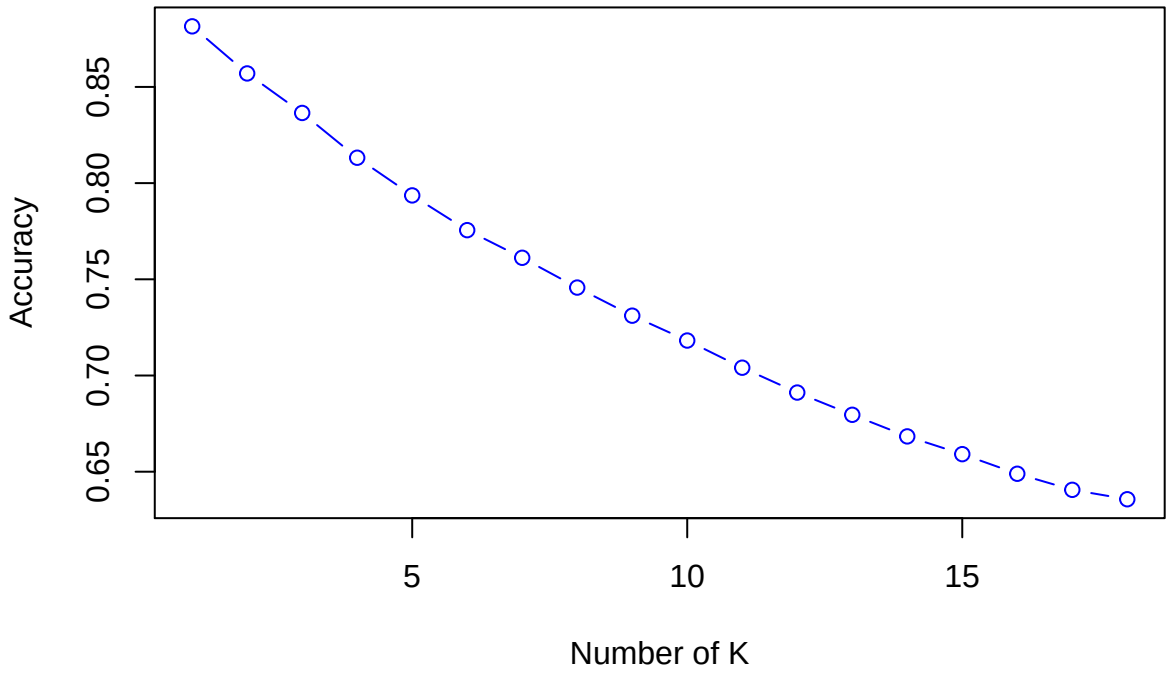
```
plot(1:18,
     rec,
     type = "b",
     main = "Recall",
     xlab = "Number of K", ylab = "Recall",
     col = 'blue')
```




```
# Examine and plot the accuracy values
acc
```

```
##  [1] 0.8814797 0.8570053 0.8364628 0.8131921 0.7936126 0.7755577 0.7611940
##  [8] 0.7457069 0.7311026 0.7181833 0.7040603 0.6911411 0.6795859 0.6683518
## [15] 0.6591237 0.6489328 0.6405874 0.6356925
```

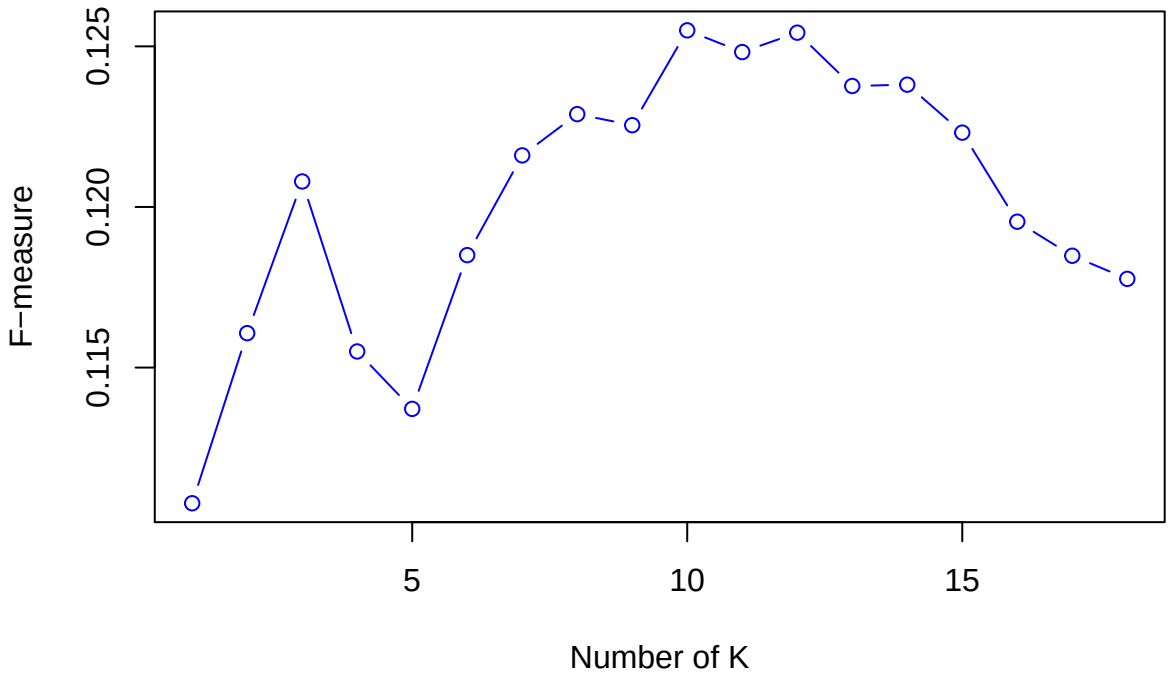
```
plot(1:18,
     acc,
     type = "b",
     main = "Accuracy",
     xlab = "Number of K", ylab = "Accuracy",
     col = 'blue')
```




```
# Examine and plot the F-measure values
F1
```

```
##  [1] 0.1107766 0.1160714 0.1207938 0.1155015 0.1137147 0.1184998 0.1216057
##  [8] 0.1228896 0.1225452 0.1254980 0.1248220 0.1254260 0.1237656 0.1238075
## [15] 0.1223140 0.1195412 0.1184806 0.1177614
```

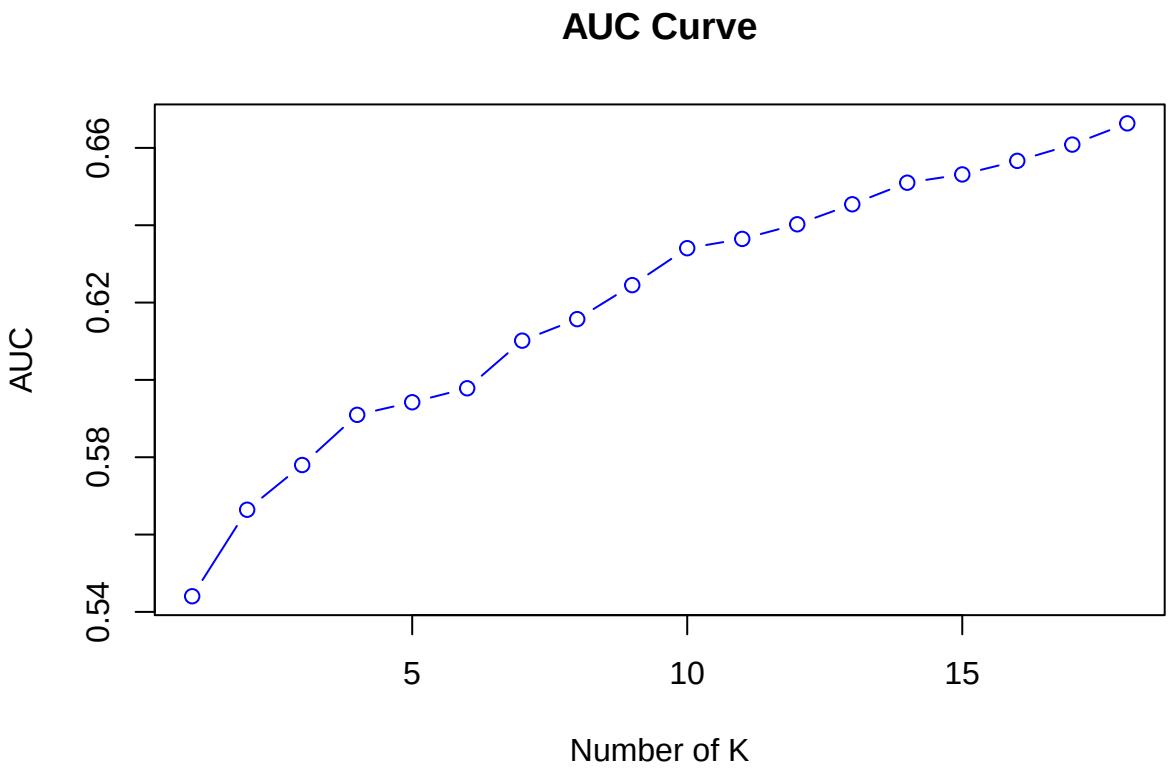
```
plot(1:18,
     F1,
     type = "b",
     main = "F-measure",
     xlab = "Number of K", ylab = "F-measure",
     col = 'blue')
```




```
# examine and plot the AUC values.
AUC_curve
```

```
##  [1] 0.5440528 0.5664193 0.5779864 0.5909675 0.5942238 0.5978321 0.6101501
##  [8] 0.6157181 0.6245124 0.6340637 0.6364628 0.6402459 0.6454217 0.6509989
## [15] 0.6531358 0.6566442 0.6608547 0.6663478
```

```
plot(1:18,
     AUC_curve,
     type = "b",
     main = "AUC Curve",
     xlab = "Number of K", ylab = "AUC",
     col = 'blue')
```



Results for KNN

After running multiple iterations of KNN, using K = 3 to K = 20 values for the nearest neighbors, we concluded that the KNN model with K = 9 performs the best which optimizes the precision and recall for the model.We noticed that the recall keeps increasing as we increase the value of k, but the precision keeps decreasing after K = 9. Hence, we have chosen this model as the best K-NN model.

Naive Bayes

Next, we used the Naive bayes classifier to train the model and evaluate its performance.

Naive Bayes for top 8 features

```
NB_model_balanced = naiveBayes(adopter ~ ., data = xyz_train_top8_balanced)
pred_nb_balanced = predict(NB_model_balanced, xyz_test_top8)
prob_pred_nb_balanced = predict(NB_model_balanced, xyz_test_top8, type = "raw")
```

Confusion matrix and metrics

```
confusionMatrix(data = pred_nb_balanced,
                reference = xyz_test_top8$adopter,
                positive = "1",
                mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 11001   333
##          1   999   129
##
##                Accuracy : 0.8931
##                  95% CI : (0.8876, 0.8985)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1158
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.11436
##                  Recall : 0.27922
##                      F1 : 0.16226
##              Prevalence : 0.03707
##          Detection Rate : 0.01035
##    Detection Prevalence : 0.09052
##       Balanced Accuracy : 0.59799
##
##        'Positive' Class : 1
##
```

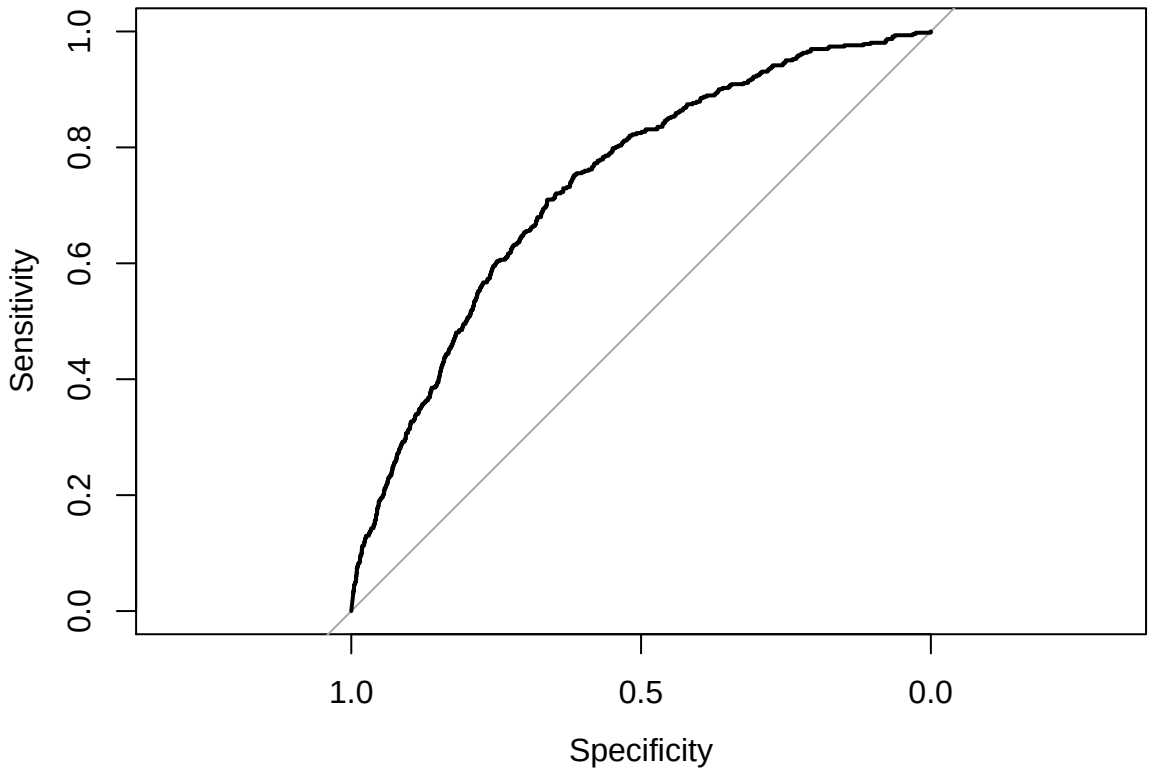
ROC curve

```
roc = roc(response = xyz_test_top8$adopter,
          predictor = prob_pred_nb_balanced[,"1"])
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc)
```




```
auc(roc)
```

```
## Area under the curve: 0.7351
```

Results for Naive Bayes

The Naive Bayes algorithm performs slightly better than KNN, having a higher F-measure = 15.29%, precision, accuracy and AUC but worse recall.

Decision Tree

Next, we used the decision tree algorithm to construct a classifier. We noticed that the feature susbcriber friend count was causing a lot of variation in the results because of outliers, so we decided to drop it and use top 7 features to build the decision tree model.

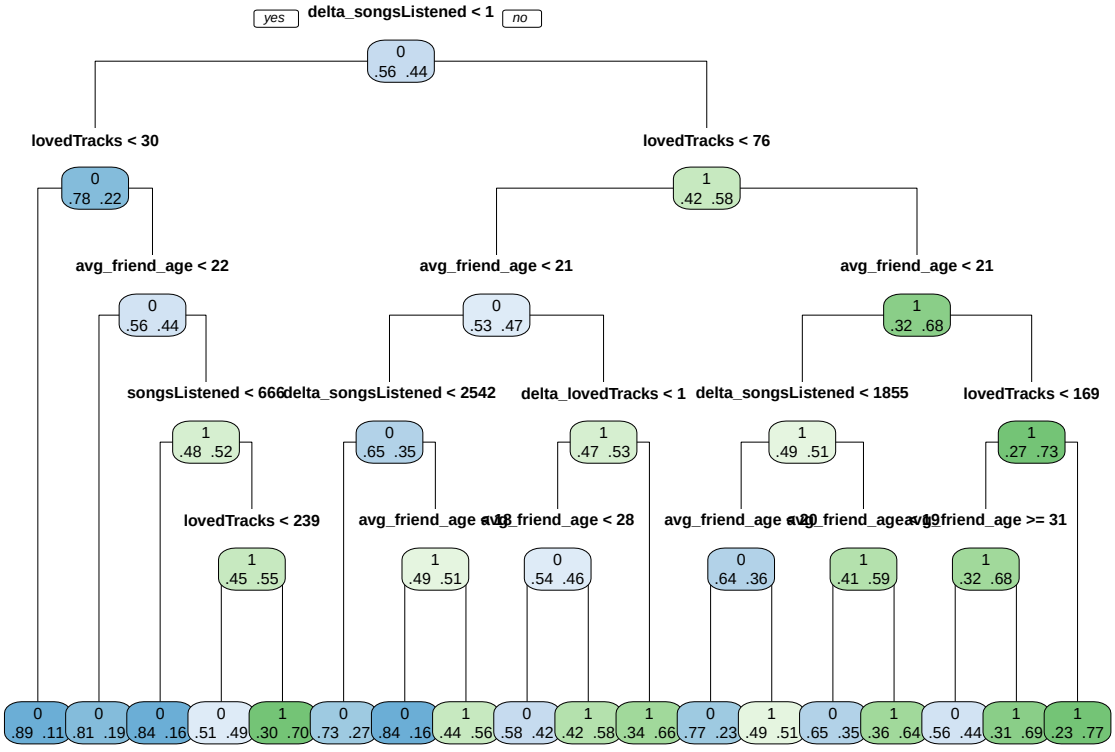
```
xyz_train_dt <- xyz_train_top8_balanced %>% select(-subscriber_friend_cnt)
xyz_test_dt <- xyz_test_top8 %>% select(-subscriber_friend_cnt)

tree = rpart(adopter ~ .,
             data = xyz_train_dt, method = "class",
             parms = list(split = "information"),
             control = list(minsplit = 200,
                            maxdepth = 5,
                            cp = 0))
```

Plotting

```
tree$variable.importance
```

```
##  delta_songsListened          lovedTracks         songsListened
##            3678.13748           2975.18836           1656.60484
##    delta_lovedTracks           friend_cnt       avg_friend_age
##            1401.56277           1393.33523           1161.76923
## delta_avg_friend_age
##              28.73381
```

```
rpart.plot(tree, type = 1, extra =4, cex=0.5)
```



model evaluation

```
pred = predict(tree, xyz_test_dt, type = "class")
confusionMatrix(pred, as.factor(xyz_test_dt$adopter),
                mode = "prec_recall", positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9009  160
##          1 2991  302
##
##                Accuracy : 0.7472
##                  95% CI : (0.7394, 0.7548)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1025
```

```
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.09171
##                  Recall : 0.65368
##                      F1 : 0.16085
##              Prevalence : 0.03707
##          Detection Rate : 0.02423
##    Detection Prevalence : 0.26424
##       Balanced Accuracy : 0.70221
##
##        'Positive' Class : 1
##
```

Result for Decision Tree

We changed the parameters, minsplit and maxdepth, to find a decision tree with good precision, recall and F-measures and chose this decision tree.

Conclusion

We chose decision tree as the best classifier for our case, because our main aim is to correctly classify the premium adopters. The decision tree has the highest recall, which represents the amount of premium adopters that were correctly classified out of all the actual premium adopters. While having a good recall rate, the decision tree classifier also doesn't degrade as much on precision as K-NN and Naive Bayes. It also has the best F-measure and a fairly good accuracy.

Appendix for variable plottings

```
b1 = ggplot(data=xyz, aes(x=age)) +
  geom_histogram(fill="steelblue", color="black") +
  xlim(0, 100) + ggtitle("Age Dist")
b2 = ggplot(data=xyz, aes(x=friend_cnt)) +
  geom_histogram(fill="steelblue", color="black") + xlim(0, 2000) + ylim(0, 10000)
ggtitle("friend_cnt Dist")
```

```
## $title
## [1] "friend_cnt Dist"
##
## attr(,"class")
## [1] "labels"
```

```
b3 = ggplot(data=xyz, aes(x=songsListened)) + xlim(0, 25000) +
  geom_histogram(fill="steelblue", color="black") +
  ggtitle("songsListened Dist")
b4 = ggplot(data=xyz, aes(x=shouts)) +  xlim(0, 1000) + ylim(0, 7500) +
  geom_histogram(fill="steelblue", color="black") +
  ggtitle("shouts Dist")

grid.arrange(b1, b2, b3, b4)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 2 rows containing missing values (geom_bar).
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 6 rows containing non-finite values (stat_bin).

## Warning: Removed 2 rows containing missing values (geom_bar).
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 10082 rows containing non-finite values (stat_bin).

## Warning: Removed 2 rows containing missing values (geom_bar).
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 122 rows containing non-finite values (stat_bin).

## Warning: Removed 2 rows containing missing values (geom_bar).
```