Technical Document for Company XYZ Classification Model Creation

Group 16

2023-07-20

Contents

1 Intro

The following document includes all analysis our group performed in R, as well as a description of any analysis we performed using additional tools, such as Python.

Not all of the work here was explicitly used in our managerial recommendations, but all of it informed our decision making. For each section, there will be a description of our problem-solving procedure.

In some cases, we performed additional analysis beyond what is included in the code and output in this document, as some of it is repetitive. In these situations, there will be a description of the kind of analysis performed and a justification of why the code was left out of the document.

1.1 Evaluation Criteria/Performance Metric

The provided data set is significantly unbalanced, so accuracy is not a meaningful metric as it could be misleadingly high if we followed a naive classification scheme where we classified all users as part of the majority class (non-subscribers).

Other measurements can be valuable, such as precision and recall - with precision likely being more useful, as it means we are more confident in the once we classify as potential subscribers and aren't including a lot of extraneous False Positives. F1 would be a useful way to balance precision and recall.

However, the performance metric we selected for the model evaluation was the AUC-ROC. Its consideration of both the true positive rate and the false positive rate makes it a better selection for imbalanced data sets. In the business context, the higher the AUC-ROC, the better the model is at distinguishing likely subscribers from unlikely subscribers, allowing marketing to better allocate their budget.

1.2 Set-Up

To prepare the data for analysis, we imported the following libraries.

```r
# Load necessary libraries
library(caret)
library(e1071)
library(dplyr)
library(ggplot2)
library(pROC)
library(class)
library(ROSE)
library(dplyr)
library(caret)
library(rpart)
library(rpart.plot)
library(class)
```

We also defined a normalization function as described in class.

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

Next, we read the *XYZData.csv* file into memory and performed pre-processing.

```r
# Load the data
data <- read.csv("XYZData.csv")

# Preprocessing the data
data$user_id <- NULL # Removing user_id as it's not a predictive feature

# Convert categorical variables into factors
data$adopter <- as.factor(data$adopter)
data$male <- as.factor(data$male)
data$good_country <- as.factor(data$good_country)
```

Next, we split the data into training and testing sets, using the common split of 70% training to 30% testing.

```r
# Splitting the data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(data$adopter, p = .7, list = FALSE)
dataTrain <- data[ trainIndex,]
dataTest  <- data[-trainIndex,]
```

Finally, we used the ROSE package to oversample the minority class. We chose to do this as it is a common strategy for dealing with imbalanced data. Since the class we care about predicting is such a small percentage of the observations, oversampling makes the most sense.

```r
# Use the ROSE package to oversample the minority class
set.seed(123)
dataTrain <- ovun.sample(adopter ~ ., data = dataTrain,
                         method = "over", seed = 123)$data
```

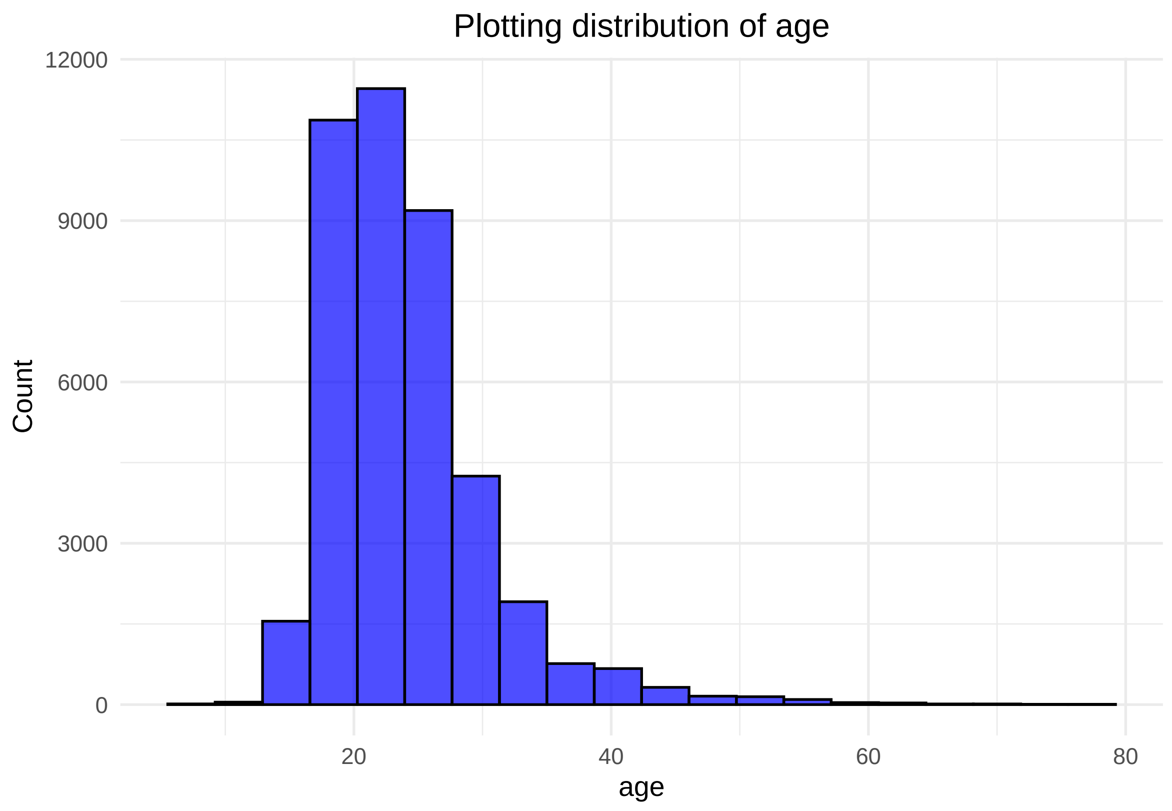
2 Descriptive Analytics

First, we performed a basic analysis, looking at the structure and summary of the data. The output has been excluded here due to its size and relative lack of importance.

Next, due to the unbalanced nature of the data, we looked at the distribution of each variable/column using histograms. The code we used to loop through each column is below, but the output has been excluded due to the number of plots and the relative lack of important of showing those in this document. An example of one of the plots is included in the subsequent code chunk.

```r
# # Iterate over each column of the data
# for(column in names(data)) {
#
#   # Check if the column is numeric (continuous)
#   if(is.numeric(data[[column]])) {
#     # Plot histogram with 10 bins
#     p <- ggplot(data, aes_string(column)) +
#       geom_histogram(bins = 20, fill = "blue", color = "black", alpha = 0.7) +
#       labs(x = column, y = "Count",
#            title = paste("Plotting distribution of", column)) +
#       theme_minimal() +
#       theme(plot.title = element_text(hjust = 0.5))
#   } else {
#     # Plot bar plot for discrete variables
#     p <- ggplot(data, aes_string(column)) +
#       geom_bar(fill = "blue", color = "black", alpha = 0.7) +
#       labs(x = column, y = "Count",
#            title = paste("Plotting distribution of", column)) +
#       theme_minimal() +
#       theme(plot.title = element_text(hjust = 0.5))
#   }
#
#   print(p)
# }
```

```r
p <- ggplot(data, aes_string('age')) +
      geom_histogram(bins = 20, fill = "blue", color = "black", alpha = 0.7) +
      labs(x = 'age', y = "Count",
           title = paste("Plotting distribution of", 'age')) +
      theme_minimal() +
      theme(plot.title = element_text(hjust = 0.5))

print(p)
```



We performed some additional analysis of the data using Microsoft Excel. This was primarily just getting used to the data structure and seeing the kind of data available to us.

3 Model Creation (No Feature Selection)

Following this descriptive analysis, we performed multiple methods of model creation without any feature selection. We considered this our baseline model, by which we would determine the performance of all the models using feature selection.

The primary concern with using a model that evaluates on all features is technical performance. Even if the overall model performance is strong from a data and prediction perspective, if it is too burdensome on the technical resources, then it's not that useful. Additionally, a model using all features likely includes a number of extraneous features that either don't contribute to the model's performance or actually detract and lead to overfitting.

3.1 K-nn Analysis

First, we performed a K-nn analysis. Initially we used a value of k = 5.

```
set.seed(123)
# identify numeric columns
```

```r
numeric_cols <- sapply(dataTrain, is.numeric)

# Standardize the numeric variables
dataTrain_s <- dataTrain %>% mutate_at(c(3:23), normalize)
dataTest_s <- dataTest %>% mutate_at(c(3:23), normalize)

# Perform k-NN
knn_pred <- knn(train = dataTrain_s,
                test = dataTest_s,
                cl = dataTrain$adopter, k = 5)

# Confusion Matrix
cm <- confusionMatrix(table(pred = knn_pred,
                            true = dataTest$adopter),
                      positive = "1")

cm
```

```
## Confusion Matrix and Statistics
##
##       true
## pred     0     1
##    0 11414   412
##    1   586    50
##
##                Accuracy : 0.9199
##                  95% CI : (0.915, 0.9246)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0503
##
##  Mcnemar's Test P-Value : 4.345e-08
##
##             Sensitivity : 0.108225
##             Specificity : 0.951167
##          Pos Pred Value : 0.078616
##          Neg Pred Value : 0.965162
##              Prevalence : 0.037073
##          Detection Rate : 0.004012
##    Detection Prevalence : 0.051035
##       Balanced Accuracy : 0.529696
```

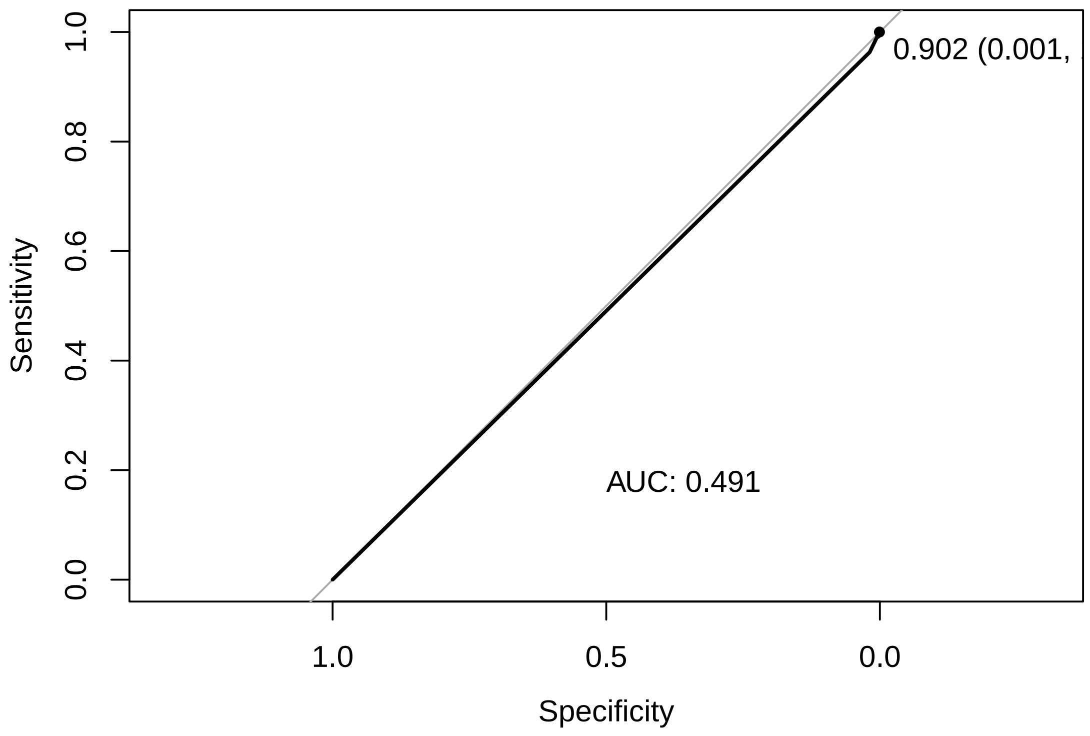
```
##
##        'Positive' Class : 1
##
```

Using code obtained from https://stackoverflow.com/questions/49915843/obtaining-an-auc-value-from-the-knn-function, we evaluated the performance of the model using the AUC and ROC curve.

```
set.seed(123)
mod <- class::knn(cl = dataTrain_s$adopter,
                  test = dataTest_s[,1:25],
                  train = dataTrain_s[,1:25],
                  k = 5,
                  prob = TRUE)

roc(dataTest_s$adopter, attributes(mod)$prob)
```

```
##
## Call:
## roc.default(response = dataTest_s$adopter, predictor = attributes(mod)$prob)
##
## Data: attributes(mod)$prob in 12000 controls (dataTest_s$adopter 0) < 462 cases (dataTest_s$adopte
## Area under the curve: 0.4909
```

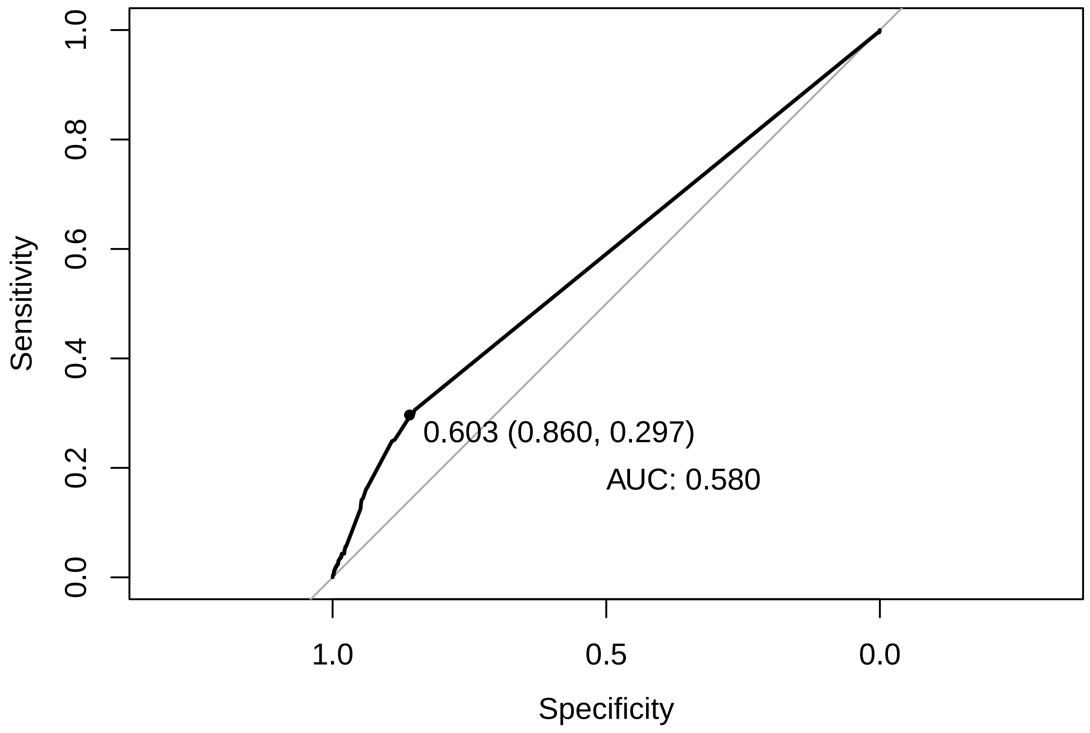
```
plot(roc(dataTest_s$adopter, attributes(mod)$prob),
     print.thres = T,
     print.auc = T,
     print.auc.y = 0.2)
```



The low AUC value suggests low overall model performance that is actually inferior to random guessing. Even adjusting for different K values, we obtained similarly poor results. For example, see the results for k = 50 below. We did find that higher values of k in later analyses did result in better model performance, with values from 100-150 seeming to have the best performance in general.

```
set.seed(123)
# Perform k-NN
knn_pred <- class::knn(cl = dataTrain_s$adopter,
                  test = dataTest_s[,1:25],
                  train = dataTrain_s[,1:25],
                  k = 50,
                  prob = TRUE)

# Confusion Matrix
cm <- confusionMatrix(table(pred = knn_pred,
                            true = dataTest$adopter),
                      positive = "1")

mod <- roc(dataTest_s$adopter,
           attributes(mod)$prob)
```

```r
plot(roc(dataTest_s$adopter,
         attributes(knn_pred)$prob),
     print.thres = T,
     print.auc = T,
     print.auc.y = 0.2)
```



3.2 Naive Bayes

Next, we did a similar analysis using the Naive Bayes approach.

```r
set.seed(123)
NB_model = naiveBayes(adopter ~ ., data = dataTrain)

# Make predictions

pred_nb = predict(NB_model, dataTest)
prob_pred_nb = predict(NB_model, dataTest, type = "raw")

confusionMatrix(data = pred_nb,
                reference = dataTest$adopter,
                mode = "prec_recall",
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 11319   376
##          1   681    86
##
##                Accuracy : 0.9152
##                  95% CI : (0.9102, 0.92)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0982
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.112125
##                  Recall : 0.186147
##                      F1 : 0.139951
##              Prevalence : 0.037073
##          Detection Rate : 0.006901
##    Detection Prevalence : 0.061547
##       Balanced Accuracy : 0.564699
##
##        'Positive' Class : 1
##
```

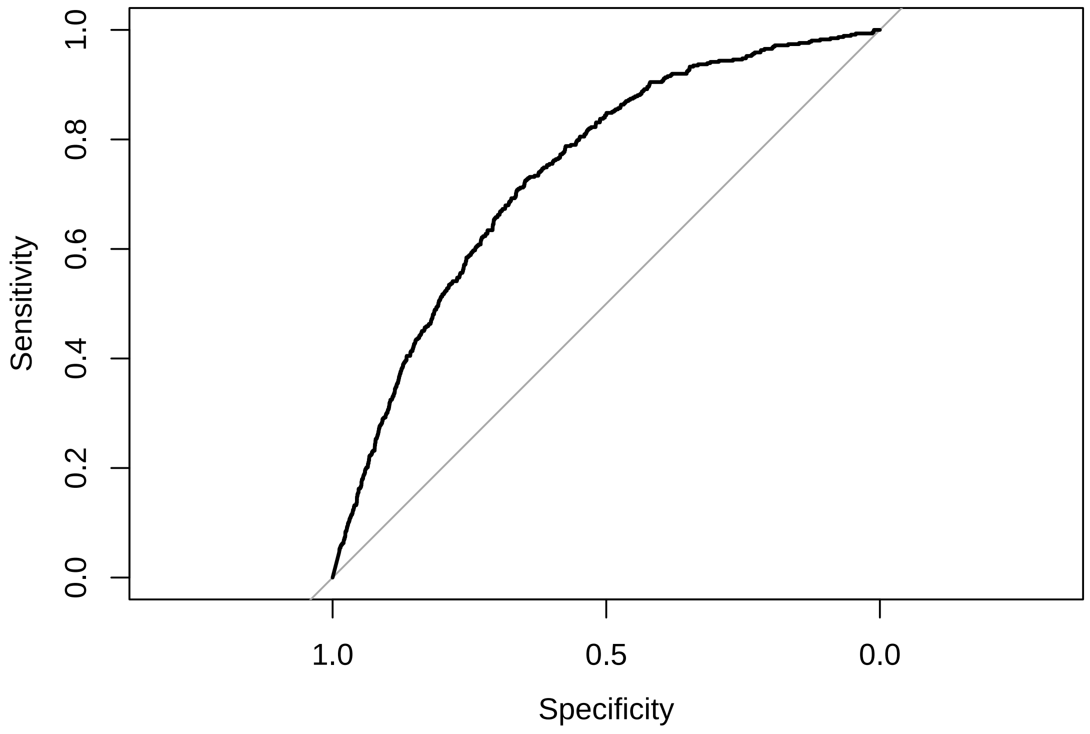
The AUC-ROC performance is below.

```
dataTest_roc = dataTest %>%
  mutate(prob = prob_pred_nb[,"1"]) %>%
  arrange(desc(prob)) %>%
  mutate(adopter_yes = ifelse(adopter=="1",1,0)) %>%
# the following two lines make the roc curve
  mutate(TPR = cumsum(adopter_yes)/sum(adopter_yes),
         FPR = cumsum(1-adopter_yes)/sum(1-adopter_yes))

roc_nb = roc(response = dataTest_roc$adopter_yes,
  predictor = dataTest_roc$prob)
```

```r
# This package also allows you to calculate the AUC.
auc(roc_nb)
```

```
## Area under the curve: 0.7398
```

```r
plot(roc_nb)
```



These results were significantly more encouraging, with a much higher AUC value of 0.7398.

3.3 Decision Tree

We also followed a decision tree process.

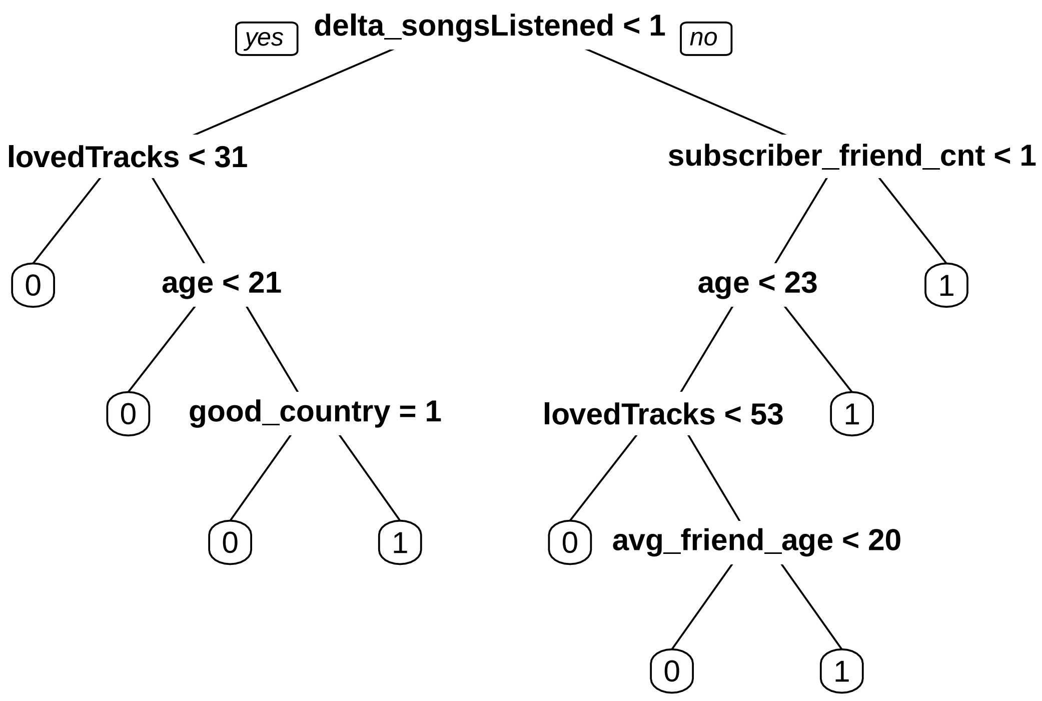
```r
set.seed(2)
library(rpart)
tree <- rpart(adopter ~ ., data = dataTrain,
              method = "class",
              parms = list(split = "information"))

pred <- predict(tree, dataTest, type = "class")
dataTest$adopter <- factor(dataTest$adopter, levels = levels(dataTrain$adopter))
```

```
confusionMatrix(pred,
                reference = dataTest$adopter,
                mode = "prec_recall",
                positive = "1")
```

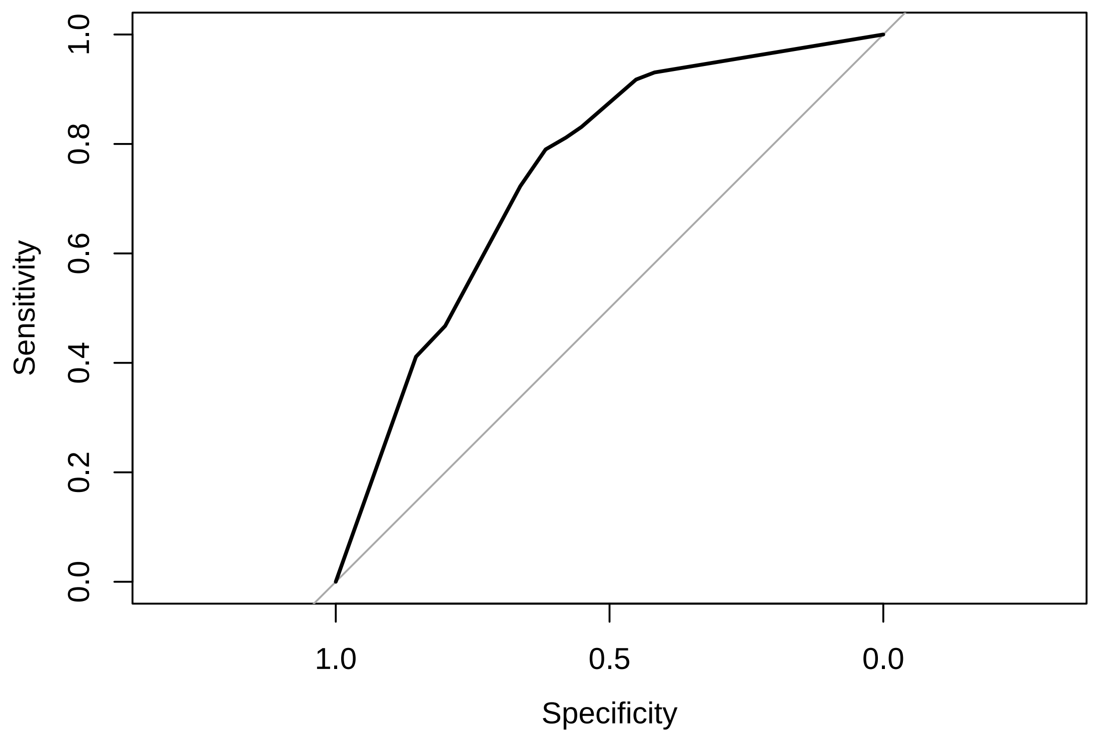
```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    0    1
##          0 7401   97
##          1 4599  365
##
##                Accuracy : 0.6232
##                  95% CI : (0.6146, 0.6317)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0716
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.07353
##                  Recall : 0.79004
##                      F1 : 0.13454
##              Prevalence : 0.03707
##          Detection Rate : 0.02929
##    Detection Prevalence : 0.39833
##       Balanced Accuracy : 0.70340
##
##        'Positive' Class : 1
##
```

We then evaluated the performance.

```
set.seed(2)
library(rpart.plot)
prp(tree, varlen = 0)
```



```
set.seed(2)
# Make ROC and calculate AUC
pred_tree_roc = predict(tree, dataTest, type = "prob")
roc_tree = roc(response = dataTest$adopter,
              predictor = pred_tree_roc[,"1"])
plot(roc_tree)
```



```
auc(roc_tree)
```

```
## Area under the curve: 0.7449
```

This process also offered encouraging results, with a higher AUC of 0.7449 and an intuitive, easy to explain decision tree model. However, we knew that feature selection would get us better model performance that would also be easier to scale, as there would be less dimensions required in the data.

4 Model Creation (Feature Selection)

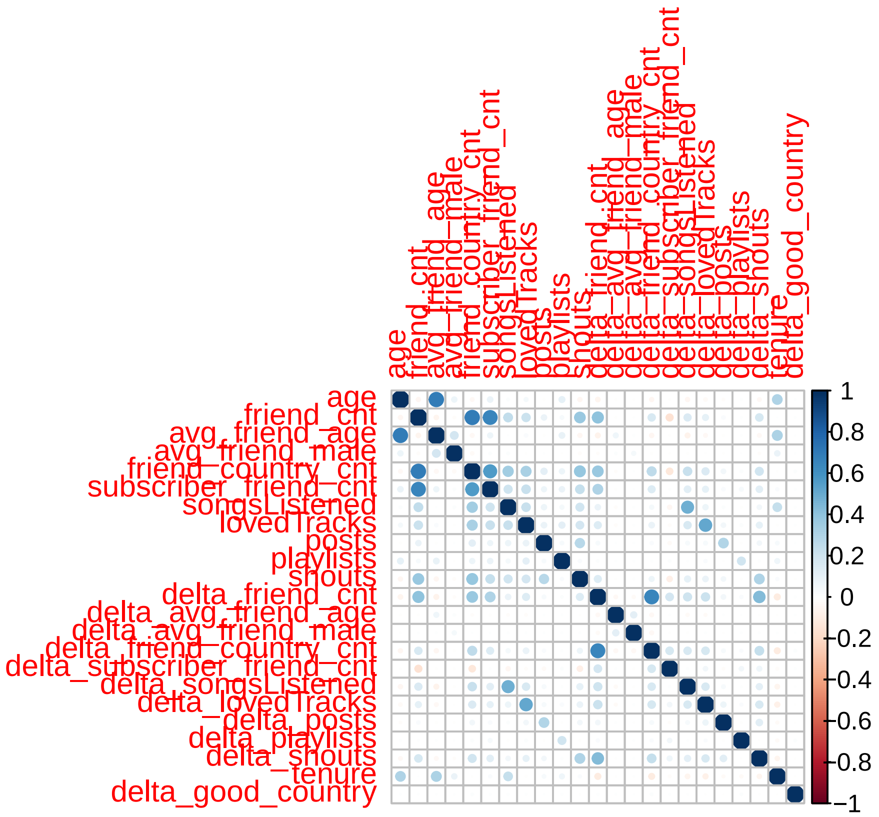
4.1 Correlation Matrix

To see what columns/variables might correlate with each other, and thereby, which columns we might be able to exclude from the analysis, we plotted a correlation matrix. However, we ultimately did not use any of this in our final analysis. ChatGPT assisted by offering some sample code that we could modify.

```
# Load the necessary library
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.3.1
## corrplot 0.92 loaded
```

```
# Select only the numeric columns
numeric_cols <- sapply(data, is.numeric)
numeric_data <- data[numeric_cols]

# Compute the correlation matrix
correlation_matrix <- cor(numeric_data, use = "pairwise.complete.obs")

# Visualize the correlation matrix
corrplot(correlation_matrix, method = "circle")
```



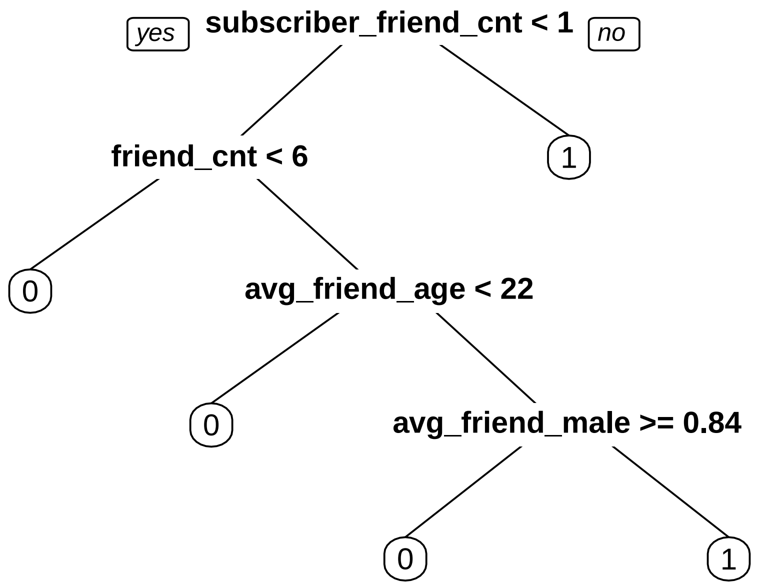
4.2 Theorized Feature Selection

Before doing anything programatically, we picked some sets of features that we thought might provide some predictive power and bucketed them into three groups - demographic information, friend characteristics/social engagement, and listening habits. These features in these groups are described in the code below.

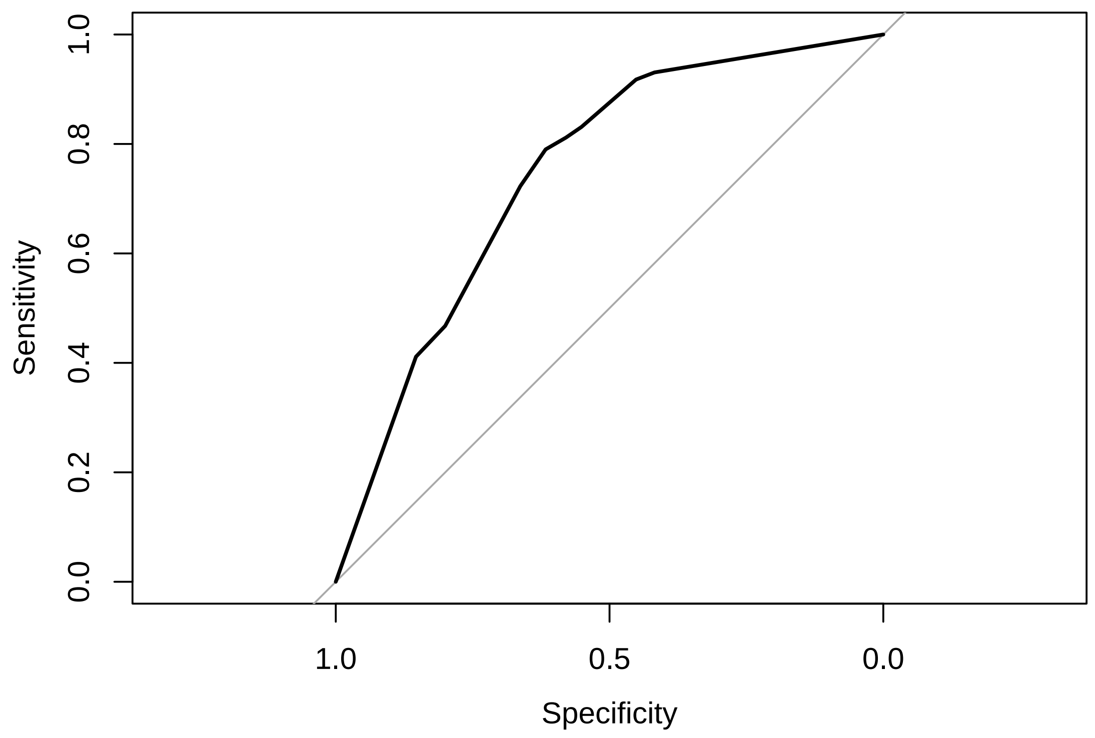
```
demographic <- c("age", "male", "good_country", "tenure")
friend_social <- c("friend_cnt", "avg_friend_age", "avg_friend_male",
                   "subscriber_friend_cnt", "shouts")
listening_habits <- c("songsListened", "lovedTracks",
                      "posts", "playlists", "tenure")
```

The friend characteristics performed the best, and since the code is similar between the three groups, only the results for this group are included here. Additionally, we have only included the decision tree code, since it had the best performance and, other than the selected columns being different, the code for the K-nn and Naive Bayes models were the same.

```
set.seed(123)

# Filter the datasets to include only the specified columns
selected_cols <- c("friend_cnt", "avg_friend_age", "avg_friend_male",
                   "subscriber_friend_cnt", "shouts", "adopter")

dataTrain_filtered <- dataTrain[, selected_cols]
dataTest_filtered <- dataTest[, selected_cols]

tree_f <- rpart(adopter ~ ., data = dataTrain_filtered,
              method = "class",
              parms = list(split = "information"))

pred <- predict(tree_f, dataTest_filtered, type = "class")
dataTest_filtered$adopter <- factor(dataTest_filtered$adopter,
                                    levels = levels(dataTrain_filtered$adopter))

prp(tree_f, varlen = 0)
```



```r
set.seed(2)
# Make ROC and calculate AUC
pred_tree_roc_f = predict(tree_f, dataTest_filtered, type = "prob")
roc_tree_f = roc(response = dataTest_filtered$adopter,
              predictor = pred_tree_roc_f[,"1"])
plot(roc_tree)
```



```r
auc(roc_tree)
```

```
## Area under the curve: 0.7449
```

4.3 Random Forest

Next, we used the *randomForest* package to determine what features might be the most important, and to work that into our model. This was determined after asking ChatGPT some of the best ways to perform feature selection and consulting the documentation at https://cran.r-project.org/web/packages/randomForest/randomForest.pdf. However, we did not otherwise use the Random Forest package to develop a model using the Random Forest method due to lack of experience.

```r
library(randomForest)
set.seed(123)
rf <- randomForest(adopter ~ ., data = data, importance = TRUE)
```

```r
importance_order <- order(rf$importance[, "MeanDecreaseAccuracy"], decreasing = TRUE)
top_features_rf <- rownames(rf$importance)[importance_order][1:10]
top_features_rf
```

```
##  [1] "delta_songsListened" "friend_cnt"          "songsListened"
##  [4] "delta_lovedTracks"    "avg_friend_age"      "friend_country_cnt"
##  [7] "age"                  "shouts"              "lovedTracks"
## [10] "tenure"
```

The 5 most important features according to this method are “delta_songsListened”, “friend_cnt”, songsListened”, “delta_lovedTracks”, and “avg_friend_age”.

4.4 Cosine Similarity

We also performed a similar analysis using the Cosine Similarity method using the *lsa* package.

```r
library(lsa)
xyz <- data
xyz$adopter <- as.factor(xyz$adopter)
xyz_normalized = xyz %>% mutate_at(c(1,3:23), normalize)

matrix = cbind(xyz_normalized$adopter,
               xyz_normalized$friend_cnt,
               xyz_normalized$avg_friend_age,
               xyz_normalized$avg_friend_male,
               xyz_normalized$friend_country_cnt,
               xyz_normalized$subscriber_friend_cnt,
               xyz_normalized$songsListened,
               xyz_normalized$lovedTracks,
               xyz_normalized$posts,
               xyz_normalized$playlists,
               xyz_normalized$shouts,
               xyz_normalized$delta_avg_friend_age,
               xyz_normalized$delta_friend_cnt,
               xyz_normalized$delta_avg_friend_male,
               xyz_normalized$delta_friend_country_cnt,
               xyz_normalized$delta_songsListened,
               xyz_normalized$delta_subscriber_friend_cnt,
               xyz_normalized$delta_lovedTracks,
               xyz_normalized$delta_posts,
               xyz_normalized$delta_shouts,
               xyz_normalized$delta_songsListened,
               xyz_normalized$delta_subscriber_friend_cnt)

# Compute cosine similarity
cosine_matrix <- cosine(matrix)
```

```
# Assign column names to the cosine matrix
col_names <- c("adopter",
               "friend_cnt",
               "avg_friend_age",
               "avg_friend_male",
               "friend_country_cnt",
               "subscriber_friend_cnt",
               "songsListened",
               "lovedTracks",
               "posts",
               "playlists",
               "shouts",
               "delta_avg_friend_age",
               "delta_friend_cnt",
               "delta_avg_friend_male",
               "delta_friend_country_cnt",
               "delta_songsListened",
               "delta_subscriber_friend_cnt",
               "delta_lovedTracks",
               "delta_posts",
               "delta_shouts",
               "delta_songsListened",
               "delta_subscriber_friend_cnt")

colnames(cosine_matrix) <- col_names

row_index <- 1  # Specify the row index for filtering
threshold <- 0.8  # Specify the cosine similarity threshold

# Filter the cosine values greater than the threshold for the specified row
filtered_values <- cosine_matrix[row_index, ] > threshold

# Get the column names of the filtered values
filtered_columns <- col_names[filtered_values]

# Print the filtered column names
print(filtered_columns)
```

```
## [1] "adopter"                  "avg_friend_age"
## [3] "avg_friend_male"          "delta_avg_friend_age"
## [5] "delta_friend_cnt"         "delta_avg_friend_male"
## [7] "delta_friend_country_cnt" "delta_songsListened"
##  [9] "delta_subscriber_friend_cnt" "delta_lovedTracks"
## [11] "delta_shouts"                "delta_songsListened"
## [13] "delta_subscriber_friend_cnt"
```

This process found that some of the most important features were “avg_friend_age”, “avg_friend_male”, and most of the “delta” features.

4.5 Information Gain Filter Approach

Since this is a classification task, the filter approach is a good way to determine important features as it emphasizes information gain, which is more meaningful in this situation.

```r
# Feature Selection
library(FSelectorRcpp)
IG = information_gain(adopter ~ ., data = dataTrain)
# e.g., select top 5
topK = cut_attrs(IG, k = 10)
topK
```

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

This process found that “songsListened”, “delta_avg_friend_age”, “delta_songsListened”, “avg_friend_age”, and “lovedTracks” were the most important features.

4.6 Model Training, Testing, and Selection

We performed a number of repetitive model generations like the ones previously described in this document with different permutations of the features, getting results similar or equal to those obtained using all features. Some of these tests used the following features.

```r
top_7_features <- c('songsListened', 'delta_avg_friend_age',
                    'delta_songsListened', 'avg_friend_age',
                    'lovedTracks', 'delta_avg_friend_male',
                    'avg_friend_male')
top_5_features <- c('songsListened', 'delta_avg_friend_age',
                    'delta_songsListened', 'avg_friend_age',
                    'lovedTracks')
top_3_features <- c('songsListened', 'delta_avg_friend_age',
                    'delta_songsListened')
```

However,, since their performance and structure was so similar, the code and results have been excluded here. The feature set we ultimately decided on was “lovedTracks”, “songsListened”, “avg_friend_age”, and “delta_songsListened” due to their high placement in the feature selection methods we explored. This obtained the best model performance, with an auc higher than the model with no feature selection. The code and results are below.

First, we processed the data.

```r
set.seed(2)

# Import the dataset
xyz <- read.csv("XYZData.csv", stringsAsFactors = TRUE)
xyz$user_id <- NULL

# assign factor
xyz$adopter <- as.factor(xyz$adopter)
```

Then we split the data and oversampled it.

```r
set.seed(2)
# Split the dataset into 70% training and 30% testing
train_rows <- createDataPartition(y = xyz$adopter, p = 0.7, list = FALSE)
xyz_train <- xyz[train_rows,]
xyz_test <- xyz[-train_rows,]

data.balanced.over <- ovun.sample(adopter ~ ., data = xyz_train, seed = 2,
                                  method = "over")$data
```

Then we filtered the data to the desired features.

```r
library(magrittr)
set.seed(2)
selected_cols <- c('lovedTracks', 'songsListened',
                   'avg_friend_age', 'delta_songsListened',
                   'adopter')

XYZ_train_filtered_c <- xyz_train[, selected_cols]
XYZ_balanced_filtered_c <- data.balanced.over[, selected_cols]
XYZ_test_filtered_c <- xyz_test[, selected_cols]
```

Firstly, we developed the decision tree model.

```
set.seed(2)
tree_f = rpart(adopter ~ ., data = XYZ_balanced_filtered_c,
            method = "class",
            parms = list(split = "information"),
                         control = rpart.control(cp = 0.0005, minsplit = 13,
                                                 maxdepth = 5))
```

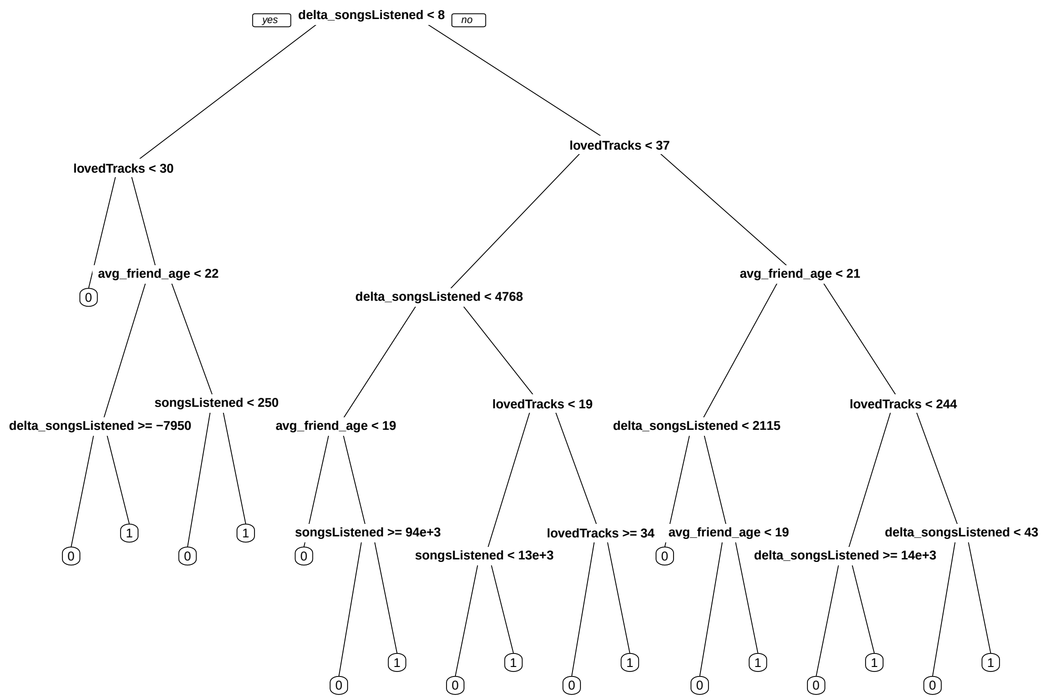
Then we evaluated the performance.

```
set.seed(2)
pred_tree_f = predict(tree_f, XYZ_test_filtered_c, type = "class")
adp_f <- as.factor(XYZ_test_filtered_c$adopter)
```

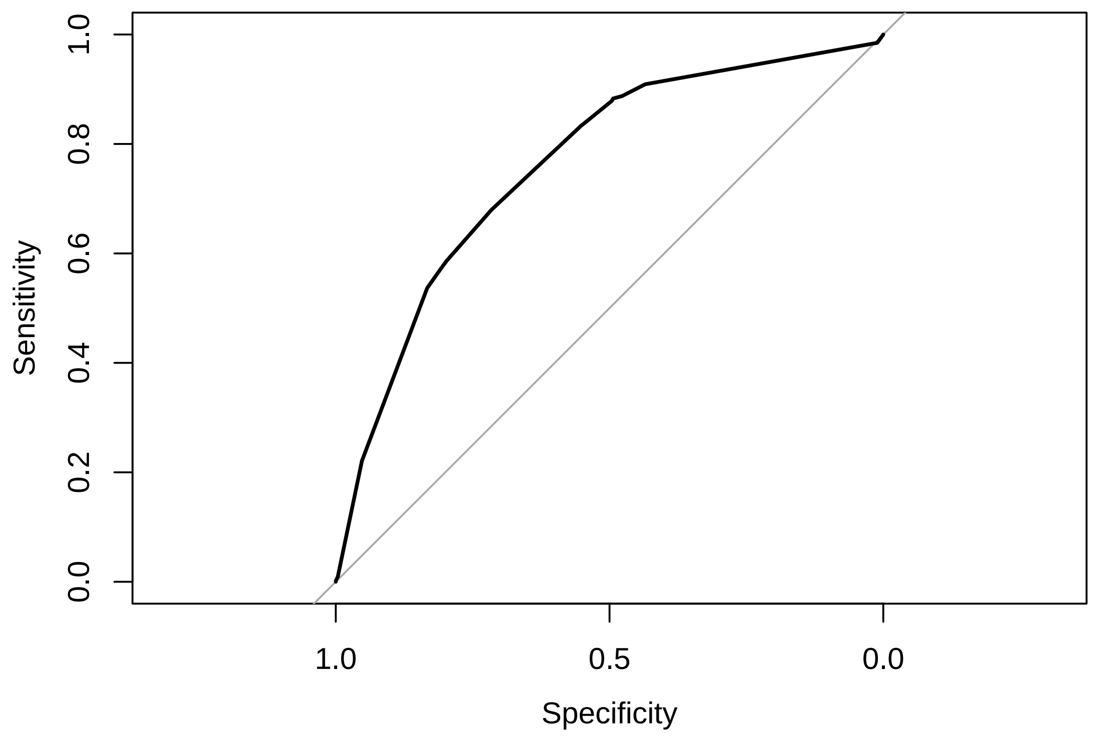
```
set.seed(2)
confusionMatrix(data = pred_tree_f,
                reference = adp_f,
                mode = "prec_recall",
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6622   77
##          1 5378  385
##
##                Accuracy : 0.5623
##                  95% CI : (0.5535, 0.571)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0591
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.06681
##                  Recall : 0.83333
##                      F1 : 0.12369
##              Prevalence : 0.03707
##          Detection Rate : 0.03089
##    Detection Prevalence : 0.46245
##       Balanced Accuracy : 0.69258
```

```
##
##        'Positive' Class : 1
##
```



```
set.seed(2)
# Make ROC and calculate AUC
pred_tree_roc_f = predict(tree_f, XYZ_test_filtered_c, type = "prob")
roc_tree_f = roc(response = XYZ_test_filtered_c$adopter,
              predictor = pred_tree_roc_f[,"1"])
plot(roc_tree_f)
```



```r
# This package also allows you to calculate the AUC.
auc(roc_tree_f)
```

```
## Area under the curve: 0.7607
```

```r
selected_tree <- tree_f
roc_selected_tree <- roc_tree_f
pred_selected_tree_roc <- pred_tree_roc_f
```

This decision tree model obtained us an auc of .7607, the highest obtained so far. However, to be thorough, we also performed the analysis using Naive Bayes and K-nn with the code below.

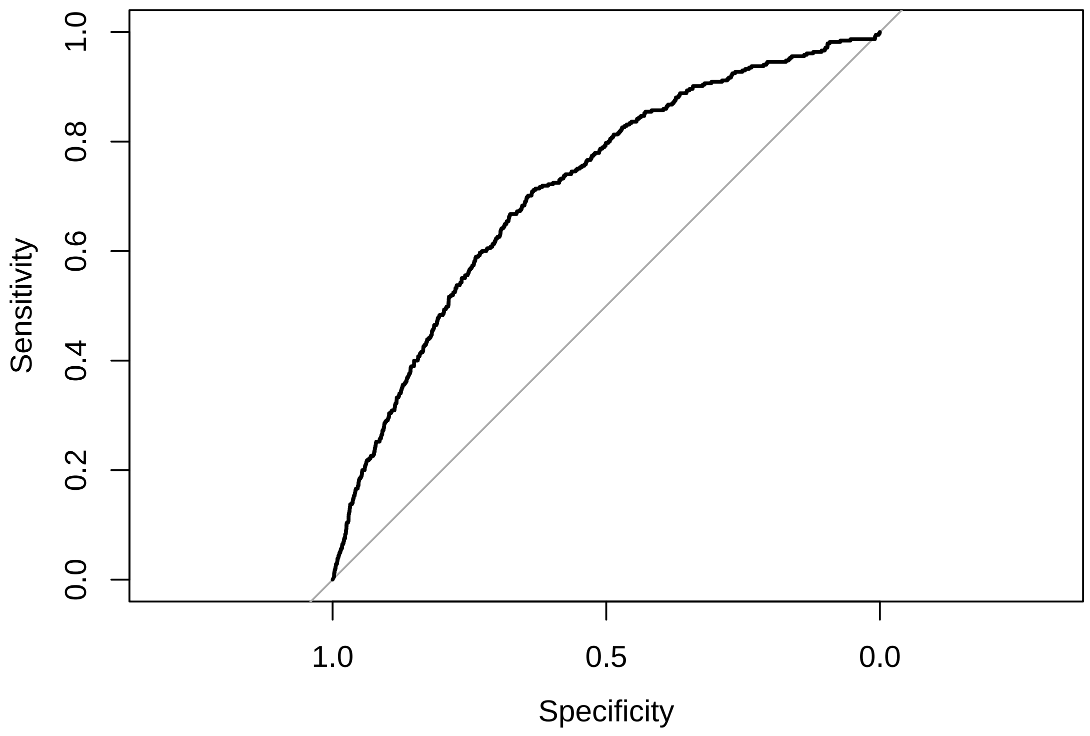
4.6.1 Naive Bayes:

```r
library(e1071)
set.seed(2)

bayes_train_rows <- createDataPartition(y = xyz$adopter, p = 0.75, list = FALSE)
bayes_xyz_train <- xyz[bayes_train_rows,]
bayes_xyz_test <- xyz[-bayes_train_rows,]
```

```r
bayes_data.balanced.over <- ovun.sample(adopter ~ ., data = bayes_xyz_train, seed = 2,
                                  method = "over")$data

set.seed(2)
#model selection base on feature selection
selected_cols <- c('songsListened', 'delta_avg_friend_age',
                   'delta_songsListened', 'avg_friend_age',
                   'lovedTracks', 'adopter')

XYZ_train_filtered_b <- bayes_data.balanced.over[, selected_cols]
XYZ_balanced_filtered_b <- bayes_data.balanced.over[, selected_cols]
XYZ_test_filtered_b <- bayes_data.balanced.over[, selected_cols]

# Train the Naive Bayes model with oversampled data
nb_model <- naiveBayes(adopter ~ ., data = XYZ_train_filtered_b)

# Make predictions on the test set
pred <- predict(nb_model, bayes_xyz_test)

# Calculate AUC
all_levels <- unique(c(levels(pred), levels(XYZ_train_filtered_b$adopter)))

pred <- factor(pred, levels = all_levels)
bayes_xyz_test$adopter <- factor(bayes_xyz_test$adopter, levels = all_levels)

pred_prob <- predict(nb_model, bayes_xyz_test, type = "raw")
roc_obj <- roc(response = bayes_xyz_test$adopter,
               predictor = pred_prob[, "1"],
               levels = all_levels)

auc_value <- auc(roc_obj)
auc_value
```

```
## Area under the curve: 0.715
```

```r
plot(roc_obj)
```



4.6.2 K-nn:

Based on tests with a number of k values, we opted for a k value of 300, as model performance appeared to degrade at higher values and performance was poorer at lower values.

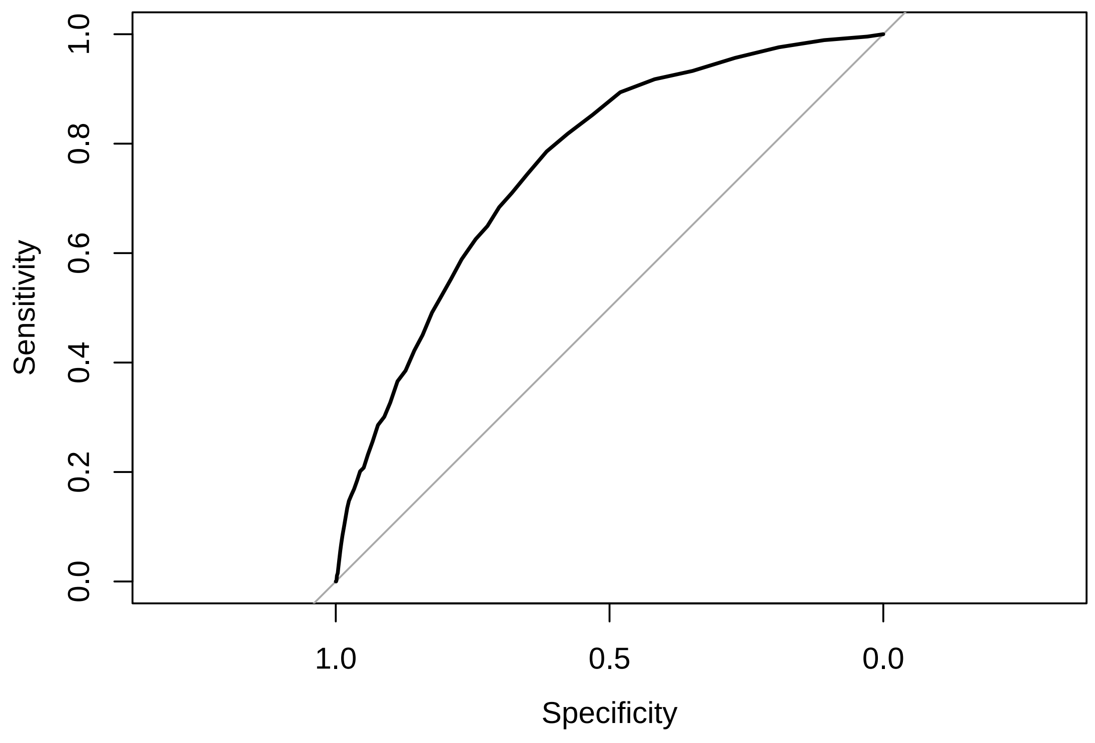
```
set.seed(2)
train_rows = createDataPartition(y = xyz$adopter, p = 0.70, list = FALSE)

selected_cols <- c('songsListened', 'delta_avg_friend_age',
                   'delta_songsListened', 'avg_friend_age',
                   'lovedTracks', 'adopter')

xyz$adopter <- as.factor(xyz$adopter)
xyz_normalized = xyz %>% mutate_at(1:25, normalize)
xyz_normalized_train = xyz_normalized[train_rows,selected_cols]
xyz_normalized_test = xyz_normalized[-train_rows,selected_cols]

library(kknn)
```

```
##
## Attaching package: 'kknn'
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
set.seed(2)
model_knn = kknn(adopter ~ .,
                 train = xyz_normalized_train,
                 test = xyz_normalized_test,
                 k = 300,
                 distance = 5,
                 kernel = "rectangular")
pred_prob_knn = model_knn$prob


knn_roc_curve = roc(response = ifelse(xyz_normalized_test$adopter == "1", 1, 0),
predictor = pred_prob_knn[,"1"])
```

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

## Setting direction: controls < cases
```

```
plot(knn_roc_curve)
```

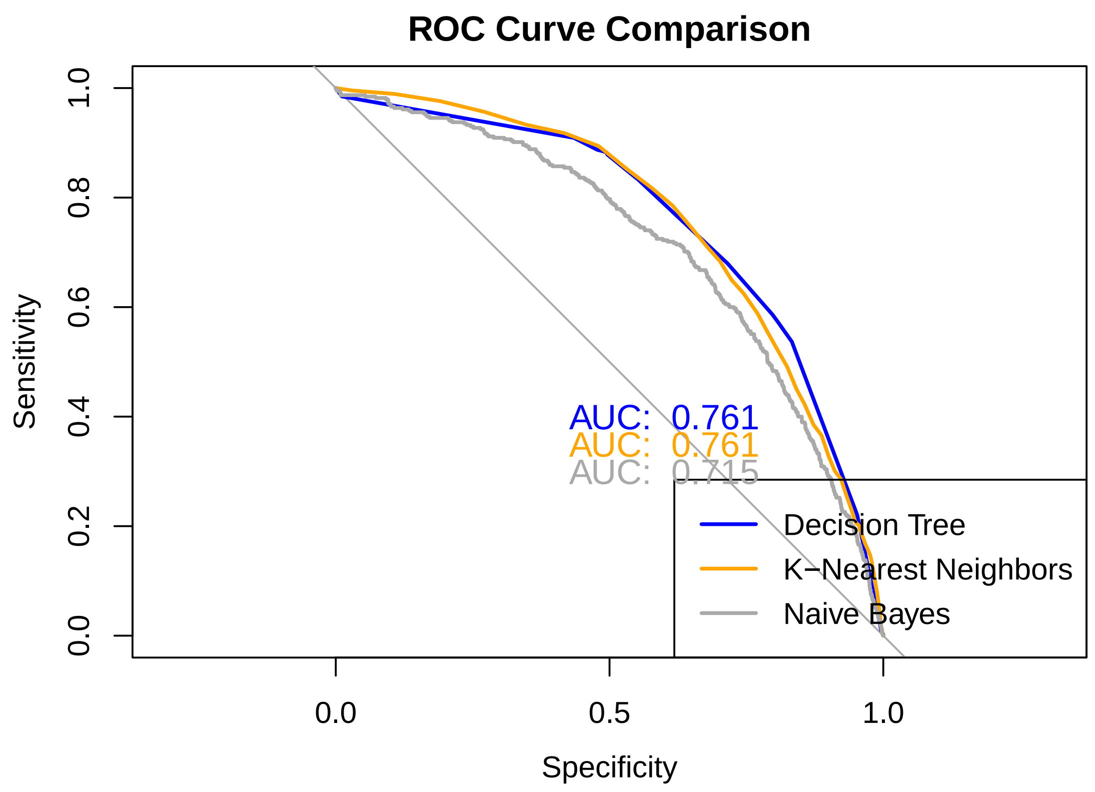


```r
auc(knn_roc_curve)
```

```
## Area under the curve: 0.7609
```

Finally, we plotted the performance of each model using these features against each other.

```r
plot(roc_tree_f,
     col = "blue",
     main = "ROC Curve Comparison",
     lwd = 2,
     xlim = c(0, 1),
     ylim = c(0, 1))
lines(knn_roc_curve, col = "orange", lwd = 2)
lines(roc_obj, col = "darkgrey", lwd = 2)
legend("bottomright",
       legend = c("Decision Tree",
                  "K-Nearest Neighbors",
                  "Naive Bayes"),
       col = c("blue",  "orange","darkgrey"),
       lwd = 2)

# Compute and display AUC values
auc_tree_f <- auc(roc_tree_f)
auc_obj <- auc(roc_obj)
auc_knn <- auc(knn_roc_curve)
# Round AUC values to 3 decimal places
auc_text <- paste("AUC: ", round(c(auc_tree_f, auc_obj, auc_knn), 3))

# Add AUC numbers to the plot
text(0.6, 0.4, auc_text[1], col = "blue", cex = 1.2)
text(0.6, 0.35, auc_text[3], col = "orange", cex = 1.2)
text(0.6, 0.3, auc_text[2], col = "darkgrey", cex = 1.2)
```



4.7 Cross Validation

To gain confidence in our results, we performed some cross validation on this particular method and this set of features.

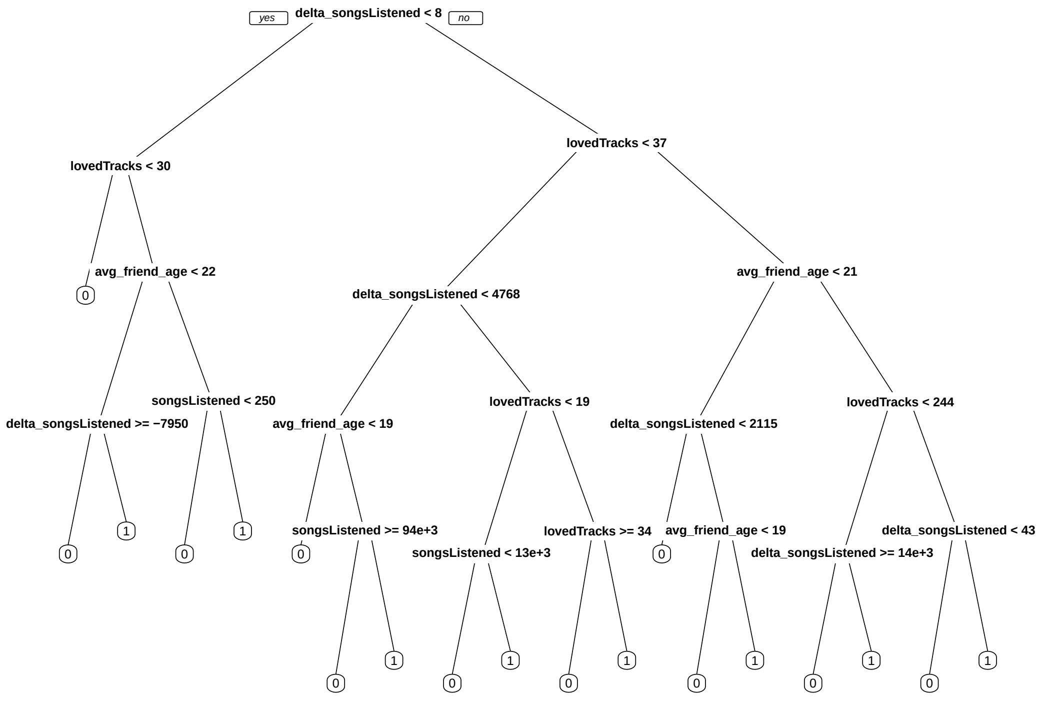
```r
# Make predictions
set.seed(2)
selected_cols <- c('lovedTracks', 'songsListened', 'avg_friend_age',
                   'delta_songsListened', 'adopter')

AUC_cv = c()
cv = createFolds(y = data$adopter, k = 10)
for (test_rows in cv) {

  dataTrain = data[-test_rows,]
  dataTest = data[test_rows,]
  dataTrain <- ovun.sample(adopter ~ ., data = dataTrain,
                           method = "over", seed = 123)$data
  dataTrain_filtered <- dataTrain[, selected_cols]
  dataTest_filtered <- dataTest[, selected_cols]
  tree_f = rpart(adopter ~ ., data = dataTrain_filtered,
           method = "class",
           parms = list(split = "information"),
                        control = rpart.control(cp = 0.0005, minsplit = 13,
                                                maxdepth = 5))



  pred_tree_f = predict(tree_f, dataTest_filtered, type = "class")
  adp_f <- as.factor(dataTest_filtered$adopter)

  pred_tree_roc_f = predict(tree_f, dataTest_filtered, type = "prob")
  roc_tree_f = roc(response = dataTest_filtered$adopter,
               predictor = pred_tree_roc_f[,"1"])

  # This package also allows you to calculate the AUC.
  AUC_cv = c(AUC_cv, auc(roc_tree_f))
}
mean(AUC_cv)
```

```
## [1] 0.7555583
```

As the obtained AUC value was similar to our prior results, we gained confidence in the validity of the model on new data.

As previously shown, our final selected model is below

```
prp(selected_tree, varlen = 0)
```



5 Analysis Performed Outside of R

5.1 Excel

We used Microsoft Excel for some initial data exploration, including some graphing and tabular analysis.

5.2 Python

We used Python to add confidence in our feature selection by performing a similar analysis using standard python methods and common packages. The code will not be replicated here, but a brief description of some of the procedures are as follows:

- Used Pandas and Numpy to analyze and manipulate the data.
- Plotted correlations.
- Developed a decision tree model using sklearn and matplotlib (for selected features and for all features).