Predicting Freemium Conversion in Website XYZ Using Classification

Madison Meinke, Natalia Mendoza-Orr, Cody Ortloff, Claire Tsao, Qianer Wu

7/24/2023

Required Packages and Data Importation

Data Exploration

Central Tendencies

First, we will take a look at the mean and median values for our features in the data set between the adopted class (1) and those that didn't adopt (0).

```
options(scipen = 999)

metric_avgs = music_labeled %>% group_by(adopter) %>%
  summarise(across(everything(), mean))

metric_medians = music_labeled %>% group_by(adopter) %>%
  summarise(across(everything(), median))

metric_avgs_t = transpose(metric_avgs)
colnames(metric_avgs_t) = c("Mean 0","Mean 1")
rownames(metric_avgs_t) = colnames(metric_avgs)
metric_avgs_t = round(metric_avgs_t, 2)

metric_medians_t = transpose(metric_medians)
colnames(metric_medians_t) = c("Median 0","Median 1")
rownames(metric_medians_t) = colnames(metric_medians)
metric_medians_t = round(metric_medians_t, 2)


comb = merge(metric_avgs_t, metric_medians_t, by = 0)
colnames(comb) = c("Feature", "Mean 0", "Mean 1", "Median 0", "Median 1")
comb
```

```
##                        Feature    Mean 0    Mean 1  Median 0  Median 1
## 1                      adopter      0.00      1.00      0.00      1.00
## 2                          age     23.94     25.82     23.00     24.00
## 3               avg_friend_age     24.02     25.29     23.00     24.40
## 4              avg_friend_male      0.62      0.63      0.67      0.67
## 5          delta_avg_friend_age      0.28      0.25      0.25      0.24
## 6        delta_avg_friend_male      0.00      0.00      0.00      0.00
## 7             delta_friend_cnt      0.83      2.49      0.00      0.00
## 8     delta_friend_country_cnt      0.10      0.31      0.00      0.00
## 9           delta_good_country      0.00      0.00      0.00      0.00
## 10           delta_lovedTracks      4.19     14.49      0.00      1.00
## 11             delta_playlists      0.00      0.01      0.00      0.00
## 12                 delta_posts      0.09      0.47      0.00      0.00
## 13                delta_shouts      0.82      5.53      0.00      0.00
## 14         delta_songsListened    922.84   2218.31      0.00    911.00
## 15 delta_subscriber_friend_cnt     -0.02     -0.06      0.00      0.00
## 16                  friend_cnt     18.76     36.99      7.00     16.00
## 17          friend_country_cnt      3.97      7.04      2.00      4.00
## 18                good_country      0.36      0.27      0.00      0.00
## 19                 lovedTracks     86.55    247.42     15.00    100.50
## 20                        male      0.62      0.73      1.00      1.00
## 21                   playlists      0.55      0.94      0.00      1.00
## 22                       posts      5.04     16.58      0.00      0.00
## 23                      shouts     28.86     76.85      4.00      9.00
## 24               songsListened  18098.18  31587.61   7717.00  19601.00
## 25       subscriber_friend_cnt      0.42      1.45      0.00      0.00
## 26                      tenure     44.33     45.57     45.00     47.00
## 27                     user_id 851971.13 886085.75 848537.50 915712.50
```

Proportion of Adopters in our data set

```
round(prop.table(table(music_labeled$adopter)), 3)
```

```
##
##     0     1
## 0.963 0.037
```

96.3 percent of the individuals in our data set did not adopt to the premium service, while 3.7 percent of them did. Our data set is greatly imbalanced with the non-adopters being the majority class (Class 0), while our positive class that we are interested in is the people who adopted (Class 1)

Comparing Classification Methods (K-NN vs Decision Tree)

K-NN

Determining the ideal value of for k in our k-NN

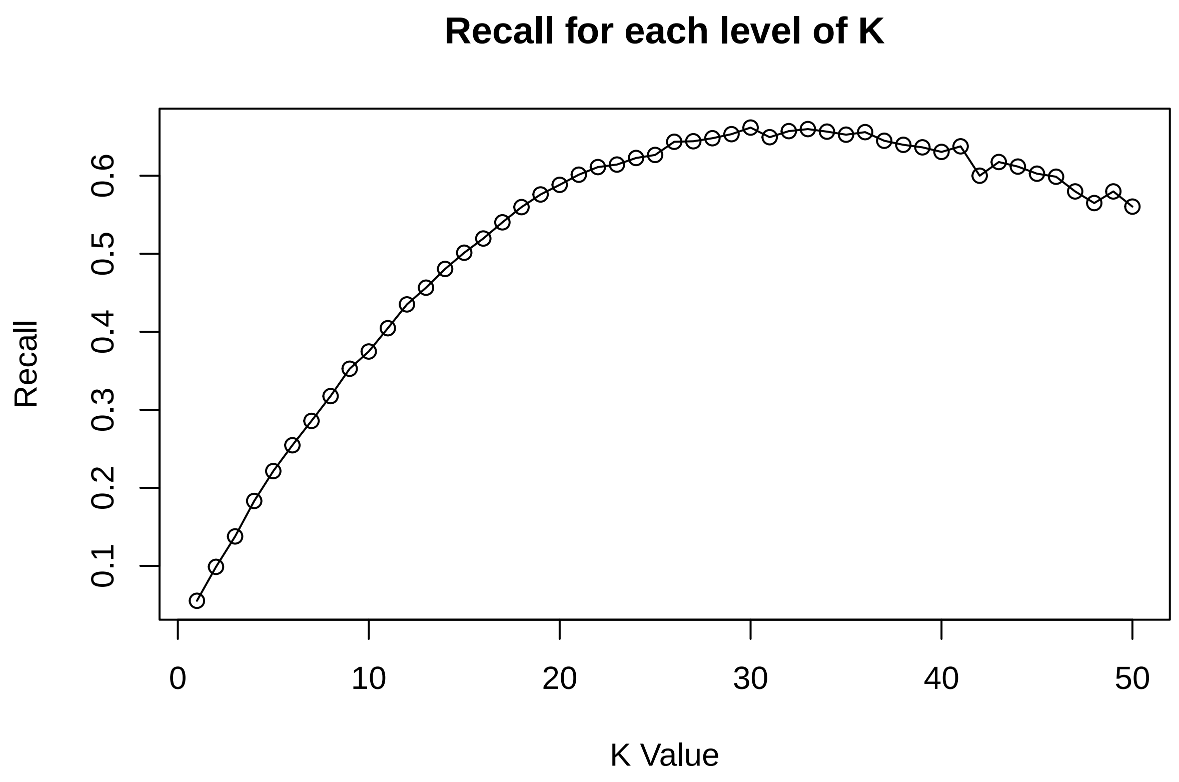
We will use cross-validation to determine the ideal number of k for our model: the number of neighbors to take into consideration when making predictions. We will do a 5 fold cross validation and test k from 1 to 50 to see which k gives the highest recall level. We are interested in looking at recall because we want to build a model that correctly classifies a large quantity of our adopters. This is of interest to us because we have so few adopters that we want to be able to capture as many as possible, without classifying every individual as an adopter.

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

# Coerce the adopter variable to a 2 level factor
music_labeled$adopter = as.factor(music_labeled$adopter)

# Normalize the Data
xyz_normalized <- music_labeled %>% mutate_at(1:26, normalize)


# 5 fold cross-validation
cv = createFolds(y = xyz_normalized$adopter, k = 5)

kval_list = c()
recall_list = c()
```

```r
for (kval in 1:50) {
  rec_cv = c()
  for (test_rows in cv) {
    # partition data into Training and Test
    xyz_norm_train = xyz_normalized[-test_rows,]
    xyz_norm_test = xyz_normalized[test_rows, ]

    # upSample training data
    xyz_norm_train = upSample(xyz_norm_train, xyz_norm_train$adopter)

    # build the k-NN model
    pred_knn = knn(train = xyz_norm_train[,2:26],
                   test = xyz_norm_test[, 2:26],
                   cl = xyz_norm_train[, 27],
                   k = kval)

    # Storing confusion Matrix
    conf <- confusionMatrix(data = pred_knn,
                            reference = as.factor(xyz_norm_test[,27]),
                            mode = 'prec_recall',
                            positive = '1')

    # Getting recall from stored confusion matrix
    recall <- conf$byClass[6]

    # Adding the recall for each fold into a list
    rec_cv = c(rec_cv, recall)
  }
  # Storing the k and average of the 5 folds recall value into lists
  kval_list = c(kval_list, kval)
  recall_list = c(recall_list,mean(rec_cv))
}
```

```r
# Turning kval_list and recall values into a Data Frame for easy visualization
df = data.frame(kval_list, recall_list)
colnames(df) = c("K", "Recall")
```

```
plot(df$K, df$Recall, main = "Recall for each level of K", xlab = "K Value", ylab = "Recall",
     type = "o")
```



```
df %>% filter(Recall == max(Recall))
```

```
##    K    Recall
## 1 30 0.6616883
```

By looking at our plot of K values and Recall level, we can see that our Recall is the highest when k = 30 (0.6616883). We will now explore feature selection using a k-NN model with k = 30 to see if we can maximize the recall level even further.

Feature Selection

Based on the previous output we will run a Wrapper's Approach: Backward Selection on the k-NN model when k = 30 to determine which features can be removed and increase performance of our model. Due to the high computational cost of running a k-NN model, we will not be performing cross validation while running our feature selection.

```
train_rows <- createDataPartition(y = music_labeled$adopter, p = 0.75, list = FALSE)

xyz_normalized <- music_labeled %>% mutate_at(1:26, normalize)
xyz_norm_train = xyz_normalized[train_rows, ]
```

```r
xyz_norm_test = xyz_normalized[-train_rows, ]

table(xyz_norm_train$adopter)
```

```
##
##     0     1
## 30000  1155
```

```r
up_xyz_norm_train <- upSample(xyz_norm_train, xyz_norm_train$adopter)

table(up_xyz_norm_train$adopter)
```

```
##
##     0     1
## 30000 30000
```

```r
# Full model
pred_knn = knn(train = up_xyz_norm_train[,2:26],
               test = xyz_norm_test[, 2:26],
               cl = up_xyz_norm_train[, 27],
               k = 30)

# Storing confusion Matrix
conf <- confusionMatrix(data = pred_knn,
                        reference = xyz_norm_test$adopter,
                        mode = 'prec_recall',
                        positive = '1')

# Getting recall from stored confusion matrix
best_recall = conf$byClass[6]

# Total features that we will cycle through
selected_features = 2:26
```

```r
while (TRUE) {
  feature_to_drop = -1
  for (i in selected_features) {
    # Partition the data into training and testing
    train = up_xyz_norm_train %>% select(setdiff(selected_features, i), adopter)
    test = xyz_norm_test %>% select(setdiff(selected_features, i), adopter)

    tr = train %>% select(-adopter)
    te = test %>% select(-adopter)

    # Building the model
    pred_knn = knn(train = tr,
                   test = te,
                   cl = train$adopter,
                   k = 30)

    # Storing the confusion Matrix
    conf <- confusionMatrix(data = pred_knn,
                            reference = test$adopter,
                            mode = 'prec_recall',
                            positive = '1')


    # Getting recall from stored confusion matrix
    recall <- conf$byClass[6]

    # If the recall for this new model with one feature taken out is higher than we want
    # to make the new recall our best recall and drop the feature taken out for next round
    if (recall > best_recall){
      best_recall = recall
      feature_to_drop = i
    }
  }

  # Drop the features and print out a list of the features that remain
  if (feature_to_drop != -1){
    selected_features = setdiff(selected_features, feature_to_drop)
    print(selected_features)
    print(best_recall)
  }
  else break
}
```

```
df1
```

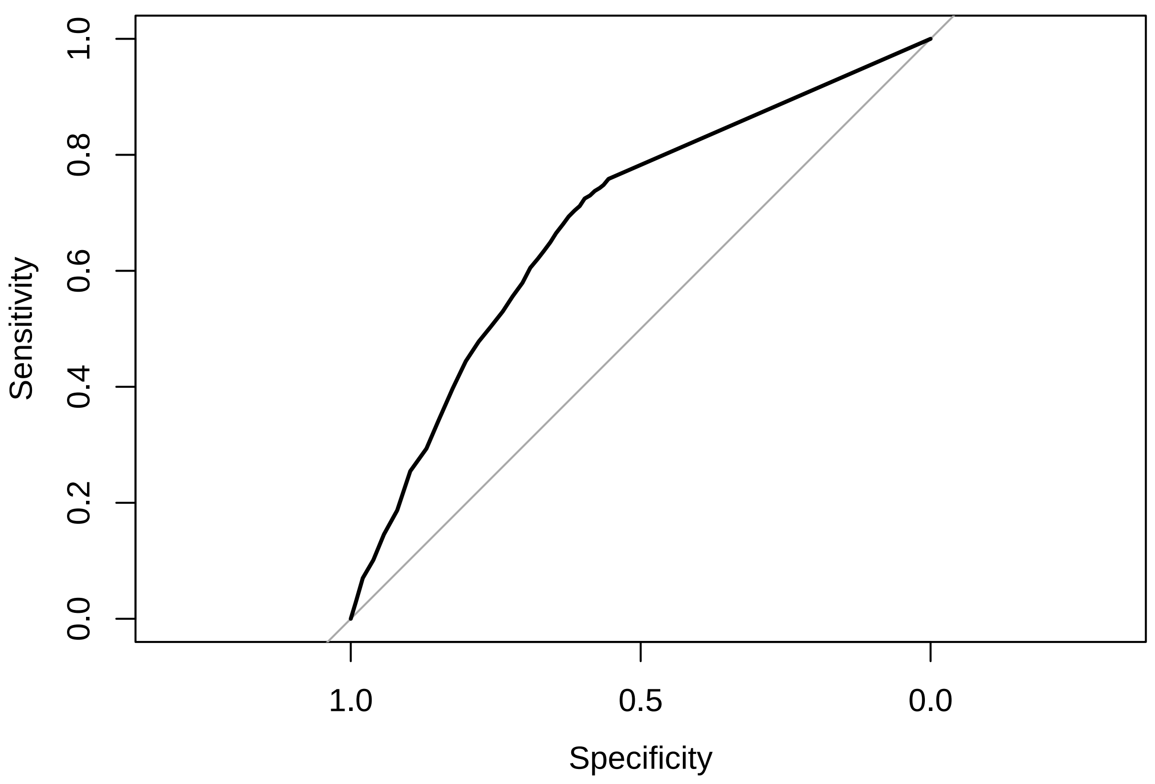
```
##                                            Feature Recall After Eliminated
## 1    First Feature Elimated       avg_friend_male                0.6597403
## 2 Second Feature Eliminated                tenure                0.7012987
## 3  Third Feature Eliminated delta_avg_friend_male                0.7246753
```

With the avg_friend_male, tenure, and delta_avg_friend_male features removed from the k-NN model when k = 30 we can achieve a max Recall of 0.7246753. Of all the individuals who adopted to the premium service we have correctly classified 72.46% of them using our model.

ROC Curve and AUC

Now, we will examine the ROC Curve and Area under the Curve (AUC) for the k-NN model when k = 30 with the insignificant features removed.

```
plot(roc_curve)
```



```
auc(roc_curve)
```

```
## Area under the curve: 0.6854
```

Findings of k-NN model

For our k-NN model we found the Recall gets maximized when k = 30 (0.6616883). By performing a Wrapper's Approach: Backward Elimination algorithm of the k = 30 k-NN model, we are able to remove 3 insignificant features (avg_friend_male, tenure, and delta_avg_friend_male) and maximize our recall at 0.7246753. , this means that of all the individuals who adopted to the premium service we have correctly classified 72.46% of them using our model.

The Area under the ROC Curve for our model is 0.6854. This means that for our model when a randomly chosen individual in our data set that is actually an adopter to the premium service and a randomly chosen individual in our data set that is actually not an adopter to the premium service is selected, then the probability that the randomly chosen individual that is actually an adopter to the premium service has a higher predicted probability of belonging to the adopted class than the individual that is not an actual adopter to the premium service is 0.6854.

Decision Tree

By their nature, decision tree models identify the features of highest importance. Therefore we will focus our attention on pruning our tree based on information gain. Within R's decision tree package, the parameter cp (complexity parameter) controls the information gain needed to create an additional split. The default value is 0.01, and we were interested in whether or not, this was the correct value for our model. We created a cross-validation model with 10 folds inside a nested loop to test a range of cp values. We were able to evaluate the summarized findings (cp value, precision, recall, and F1 score) and choose our preferred value of cp based.

Due to the unbalanced nature of the data, over sampling of the positive class (1) will be needed. During the cross validation process, the 9 folds that were to be used as the training data would be over sampled in order to help build our model. This model would in turn be tested on the remaining fold with over sampling the positive class nor under sampling the negative class (0).

Evaluation of cp

```r
# Create a nested loop that utilizes cross validation over a range of cp values that
# collects and reports the precision, recall, F1 score for evaluation.

# Create a 10-fold cross-validation split
cv = createFolds(y=music_labeled$adopter, k=10)

# The range of cp values to be evaluated
seqn = seq(0.001, 0.02, 0.001)

# Empty lists to store our summarized statistics
prec_all = c()
re_call_all = c()
f1_all = c()

for (c_p in seqn) {
  prec = c()
  re_call = c()
  f1 = c()

  for (test_rows in cv) {
    music_train = music_labeled[-test_rows,2:27]
    music_test = music_labeled[test_rows,2:27]
    # over sampling the positive class, which is in the minority within the data set
    minority_data = subset(music_train, adopter == 1)
    majority_data = subset(music_train, adopter == 0)
    miority_size = nrow(minority_data)
    ovun_train = ovun.sample(adopter ~ ., data=music_train,
                         method="both", N=miority_size, seed=3001)

    tree = rpart(adopter ~ ., data = ovun_train$data,
                 method = "class",
                 parms = list(split = "information"),
                 control = list(maxdepth =4,
                               cp = c_p))

    pred_tree_prune = predict(tree, music_test, type = "class")
    conf = confusionMatrix(data = pred_tree_prune,
                           reference = music_test$adopter,
                           mode = "prec_recall",
                           positive = "1")
    # for each confusion matrix generated, we want to capture the precision, recall, and
    # F-1 score
    prec_val = conf$byClass[5]
    re_call_val = conf$byClass[6]
    f1_val = conf$byClass[7]

    prec = c(prec, prec_val)
    re_call = c(re_call, re_call_val)
    f1 = c(f1, f1_val)

  }

  # Average the performance metrics for each series of folds to understand the
  # performance of each cross-validation set
  prec_all = c(prec_all, mean(prec))
  re_call_all = c(re_call_all, mean(re_call))
  f1_all = c(f1_all, mean(f1))
}
```

```
# print out a summary of the loop
summary_df <- data.frame(seqn, prec_all, recall_all, f1_all)
summary_df
```

```
##     seqn   prec_all recall_all    f1_all
## 1  0.001 0.07976756  0.7136364 0.1430542
## 2  0.002 0.07973957  0.7103896 0.1429653
## 3  0.003 0.07973957  0.7103896 0.1429653
## 4  0.004 0.07980525  0.7097403 0.1430775
## 5  0.005 0.08014537  0.7246753 0.1439741
## 6  0.006 0.08014537  0.7246753 0.1439741
## 7  0.007 0.07866665  0.7441558 0.1419954
## 8  0.008 0.07868972  0.7435065 0.1420269
## 9  0.009 0.07868972  0.7435065 0.1420269
## 10 0.010 0.07853976  0.7409091 0.1417403
## 11 0.011 0.07872465  0.7396104 0.1419739
## 12 0.012 0.07927109  0.7292208 0.1427271
## 13 0.013 0.07859002  0.7298701 0.1416291
## 14 0.014 0.07872899  0.7214286 0.1417104
## 15 0.015 0.07762237  0.7344156 0.1402049
## 16 0.016 0.07762237  0.7344156 0.1402049
## 17 0.017 0.07762237  0.7344156 0.1402049
## 18 0.018 0.07762237  0.7344156 0.1402049
## 19 0.019 0.07778493  0.7279221 0.1403447
## 20 0.020 0.07736172  0.7214286 0.1395478
```

Ultimately we chose a value of cp = 0.008. This value, along with our max depth of 4, provided us with reasonably high value for recall (0.7435) and a balanced F-1 score (0.142) for this data set. Neither were highest achieved, but we are cognizant of the potential of choosing a cp value that is too small. The best recall we achieved was 0.7441 when cp = 0.007. A smaller cp value will lead to a more intricate tree and much higher potential of over fitting our model to this particular data set.

```r
summary_df %>% arrange(desc(recall_all)) %>% slice_head(n = 2)
```

```
##    seqn   prec_all recall_all    f1_all
## 1 0.007 0.07866665  0.7441558 0.1419954
## 2 0.008 0.07868972  0.7435065 0.1420269
```

Build Decision Tree

Using cp = 0.008 we will build our Decision Tree

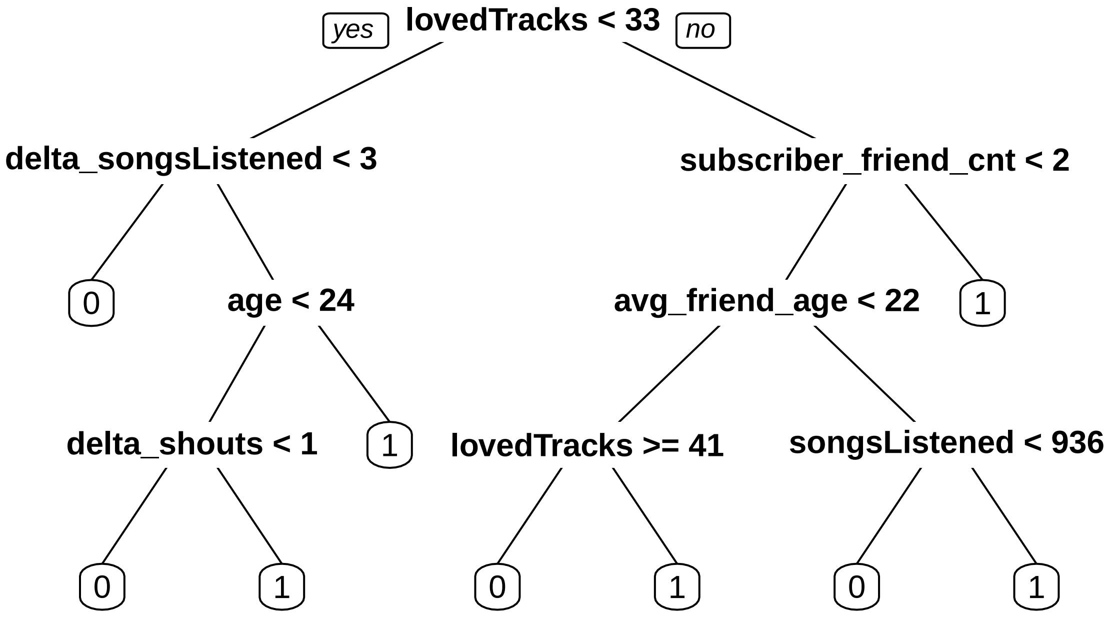
```r
# Build model with our chosen parameters and evaluate its performance on
# an over sampled test/train set (80/20 split)
# cp = 0.008

music_labeled$adopter = as.factor(music_labeled$adopter)
set.seed(123)
train_rows = createDataPartition(y = music_labeled$adopter, p = 0.8, list = FALSE)
music_train = music_labeled[train_rows,2:27]
music_test = music_labeled[-train_rows,2:27]
minority_data = subset(music_train, adopter == 1)
majority_data = subset(music_train, adopter == 0)
miority_size = nrow(minority_data)
ovun_train = ovun.sample(adopter ~ ., data=music_train, method="both", N=miority_size,
                         seed=3001)

tree = rpart(adopter ~ ., data = ovun_train$data,
             method = "class",
             parms = list(split = "information"),
             control = list(maxdepth = 4,
                            cp = 0.008))

pred_tree_prune = predict(tree, music_test, type = "class")
conf = confusionMatrix(data = pred_tree_prune,
                       reference = music_test$adopter,
                       mode = "prec_recall",
                       positive = "1")
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5290   86
##          1 2710  222
##
##                Accuracy : 0.6635
##                  95% CI : (0.6532, 0.6736)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.075
##
```

```
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##               Precision : 0.07572
##                  Recall : 0.72078
##                      F1 : 0.13704
##              Prevalence : 0.03707
##          Detection Rate : 0.02672
##    Detection Prevalence : 0.35291
##       Balanced Accuracy : 0.69101
##
##        'Positive' Class : 1
##
```

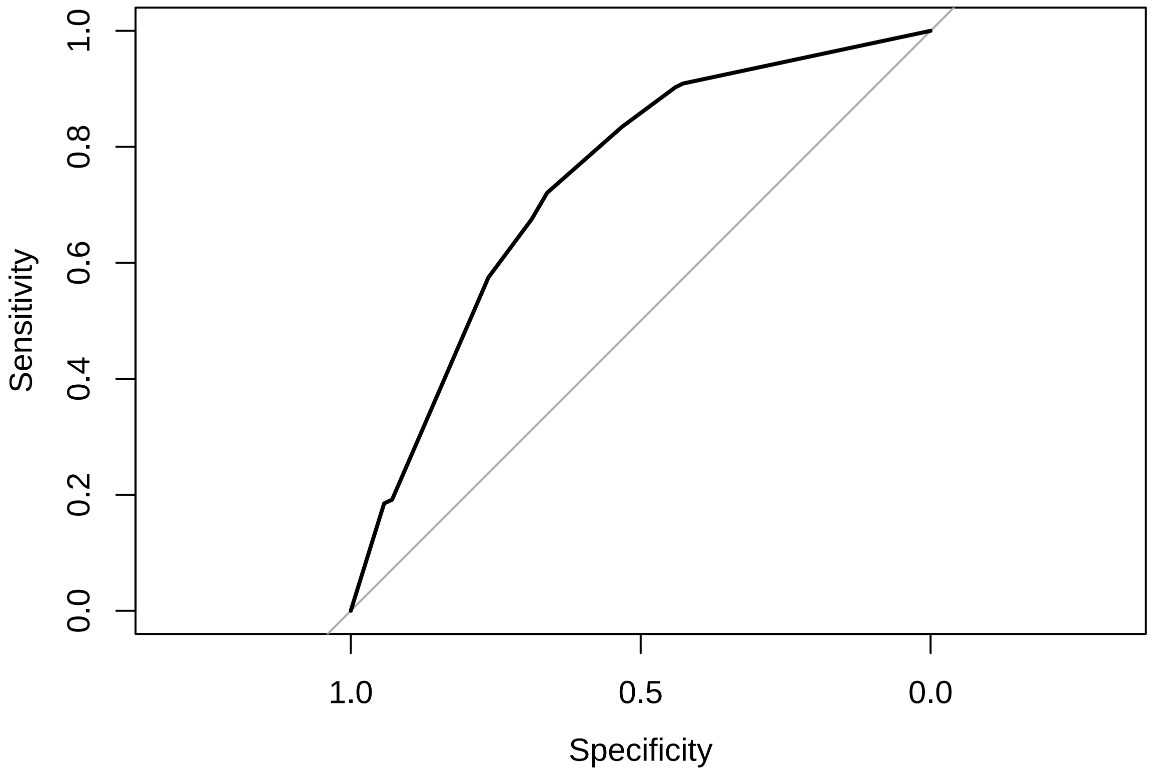
Visualization of Decision Tree

```
# Plot decision tree
prp(tree, varlen = 0)
```



ROC Curve and AUC

```
plot(tree_roc)
```

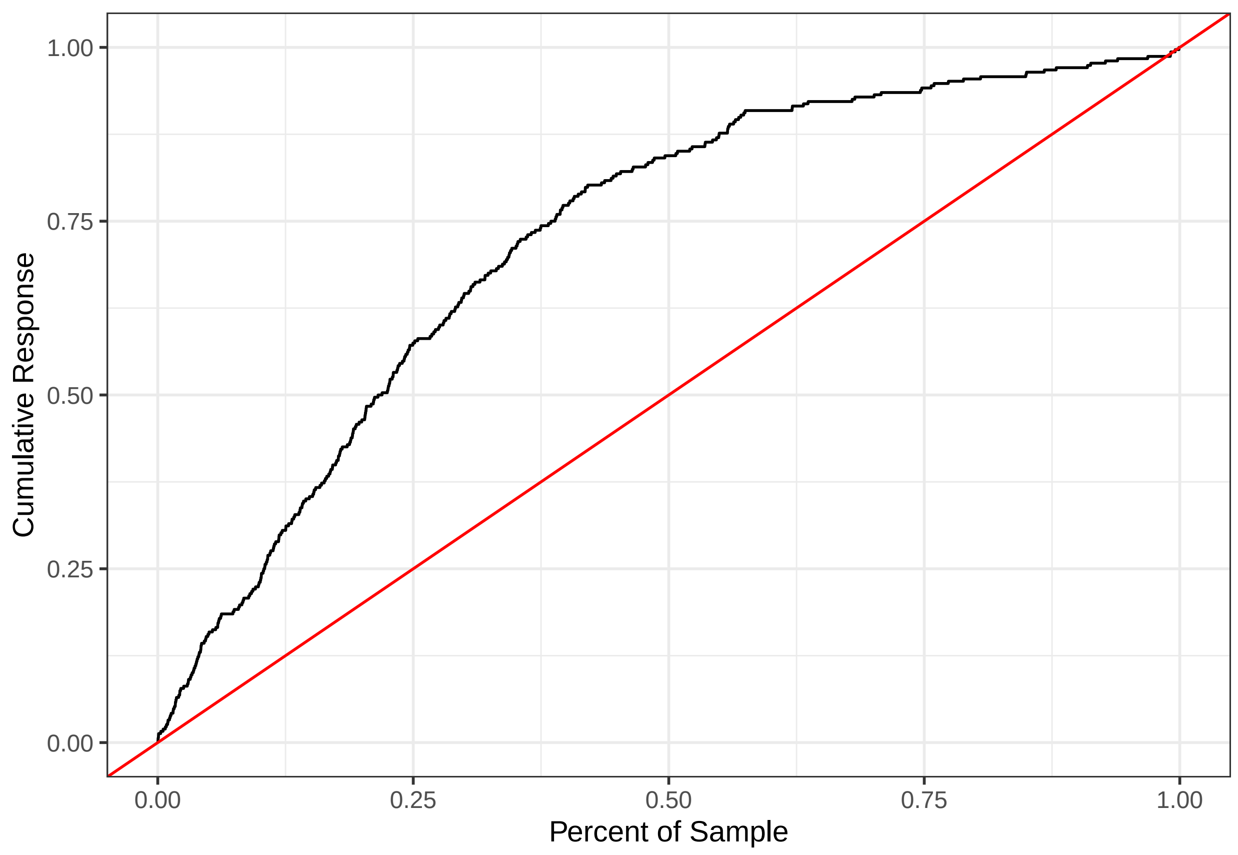


```
# calculate AUC
auc(tree_roc)
```

```
## Area under the curve: 0.7367
```

Cummulative Response Curve

```
# plot Cumulative Response Curve
music_test_cr = music_test %>%
    mutate(prob = pred_tree_prob[,"1"]) %>%
    arrange(desc(prob)) %>%
    mutate(yes_1 = ifelse(adopter=="1",1,0)) %>%
    # the following two lines make the cumulative response curve
    mutate(y = cumsum(yes_1)/sum(yes_1),
           x = row_number()/nrow(music_test))

crc <- ggplot(data = music_test_cr, aes(x = x, y = y)) +
  xlab("Percent of Sample") +
  ylab("Cumulative Response") +
  geom_line() +
  theme_bw()

crc + geom_abline(intercept = 0, slope = 1, color = 'red')
```



Using the XYZ's company to interpret this in business context: By using our model, in a marketing campaign of 41,540 individuals company XYZ can use our model to only target the top 10,362 individuals as predicted by our model. Of these individuals they are likely to obtain 847 adopters, or 55% of total adopters. Without implementing our model, company XYZ would only have gotten 385 adopters from the first 10,362 individuals. They are able to get 2.2 times as many adopters in the first 25% of the sample, which they may be interested in if cost for the marketing campaign was a concern for the company.

Descion Tree findings

For our Decision Tree model we have a maximum of 4 splits to classify an individual as an adopter of the premium service or not. Our tree uses the features of lovedTracks, delta_songsListended, subscriber_friend_cnt, age, avg_friend_age, delta_shouts, and songsListened to determine our prediction, and is able to achieve a Recall of 0.72078, using cp = 0.008. Again, this means that of all the individuals who adopted to the premium service we have correctly classified 72.08% of them using our model.

The Area under the ROC Curve for our model is 0.7367. This means that for our model when a randomly chosen individual in our data set that is actually an adopter to the premium service and a randomly chosen individual in our data set that is actually not an adopter to the premium service is selected, then the probability that the randomly chosen individual that is actually an adopter to the premium service has a higher predicted probability of belonging to the adopted class than the individual that is not an actual adopter to the premium service is 0.7367.

Final Thoughts

Our recall level for our k-NN model when k = 30 with feature selection (0.7246753) and our recall level for our Decision Tree model when cp = 0.008 (0.72078) are very close together. However, our Decision Tree model has a slightly higher AUC than our k-NN model ($0.7367 > 0.6854$) so we will choose to use our Decision Tree model rather than our k-NN model to make prediction for the XYZ music company. If given more information about the budget of the XYZ company's marketing campaign we could make even deeper recommendations using the Cumulative Response curve as well.