Freemium to Premium —— Devising An Optimized Marketing Campaign Driven by Predictive Models

2023-07-23

```r
library(rpart)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(class)
library(rpart.plot)
library(ggplot2)
library(ROSE)
```

```
## Warning: package 'ROSE' was built under R version 4.3.1

## Loaded ROSE 0.0-4
```

```r
#read the data and get a data-frame
XYZData = read.csv("XYZData.csv")
```

Decision Tree Model

In the given business context of marketing campaign with 3.7% conversion rate, we chose recall to be the parameter to be maximized as recall captures potential subscribers who are our targets.

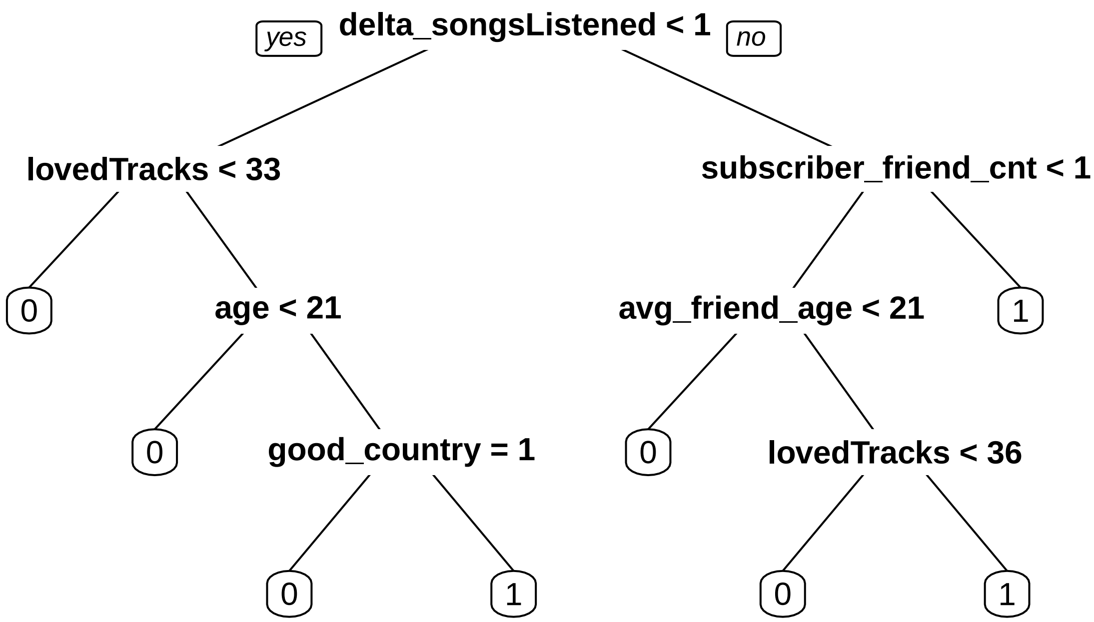
We tested different models - Decision Tree, KNN, and Naive Bayes before deciding on Decision Tree and we decided to use the decision tree since it is delivers better results of recall and having better control over the model output.

```r
#split data into training and testing sets
music_data = XYZData
train_rows = createDataPartition(music_data$adopter, p = 0.80, list = FALSE)
music_data_train = music_data[train_rows,]
music_data_test = music_data[-train_rows,]

#oversampling data since the original dataset is highly imbalanced
#setting p as 0.45 to increase the proportion of the minority class
#to be 45% in the whole dataset
oversampled_data <- ovun.sample(adopter ~., data = music_data_train,
                    method = "over", p=0.45, seed = 123)
oversampled_data_train = oversampled_data$data

#fitting the classification decision tree model
tree = rpart(adopter ~ ., data = oversampled_data_train[,2:27],
             method = "class",
             parms = list(split = "information"))

#post-processing and evaluating the performance of the tree model
prp(tree, varlen = 0)
```




```r
#uses the trained decision tree model to predict class probabilities
#for the test dataset (music_data_test), containing predictor
#variables in columns 2 to 27
pred_tree = predict(tree, music_data_test[,2:27], type = "prob")
```

```r
#uses the trained decision tree model to predict the class labels for the test dataset
pred_tree2 = predict(tree, music_data_test[,2:27], type = "class")

#If the probability of the positive class (class 1) is greater than 0.35,
#it is assigned a value of 1; otherwise, it is assigned a value of 0.
pred = ifelse(pred_tree[,2] > 0.35, 1, 0)

#setting confusion matrix to evaluate the proformance of decision tree model
confusionMatrix(data = as.factor(pred),
                reference = as.factor(music_data_test$adopter),
                mode = "prec_recall",
                positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4535   45
##          1 3501  227
##
##                Accuracy : 0.5732
##                  95% CI : (0.5625, 0.5839)
##     No Information Rate : 0.9673
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0559
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.06089
##                  Recall : 0.83456
##                      F1 : 0.11350
##              Prevalence : 0.03274
##          Detection Rate : 0.02732
##    Detection Prevalence : 0.44872
##       Balanced Accuracy : 0.69945
##
##        'Positive' Class : 1
##
```

Cross-Validation

We used the cross-validation method to evaluate the performance of the decision model because we believe the cross-validation method provide more robust and stable result by mitigating the influences of overfitting. Performing cross-validation for the same, we get a mean recall of 0.81 and a mean precision of 0.07

```r
#create cross-validation folds for the target variable 'adopter'
#k=5 sets the number of folds to be 5, meaning the whole dataset
#will be divided into 5 subsets
cv = createFolds(y = music_data$adopter, k = 5)


recall_cv = c()
```

```r
precision_cv = c()

#set up the for loop
for (test_rows in cv) {
  music_data_train = music_data[-test_rows,]
  music_data_test = music_data[test_rows,]
  oversampled_data <- ovun.sample(adopter ~., data = music_data_train,
                    method = "over", p=0.45, seed = 123)
  oversampled_data_train = oversampled_data$data
  tree = rpart(adopter ~ ., data = oversampled_data_train[,2:27],
             method = "class",
             parms = list(split = "information"))
  pred_tree = predict(tree, music_data_test[,2:27], type = "prob")
  pred = ifelse(pred_tree[,2] > 0.35, 1, 0)
  confusion_mat <- confusionMatrix(data = as.factor(pred),
                reference = as.factor(music_data_test$adopter),
                mode = "prec_recall",
                positive = '1')
  recall_cv = c(recall_cv, confusion_mat$byClass['Recall'])
  precision_cv = c(precision_cv, confusion_mat$byClass['Precision'])
}
mean(recall_cv)
```

```
## [1] 0.8101002
```

```r
mean(precision_cv)
```

```
## [1] 0.06584123
```

Lift-Curve Plot

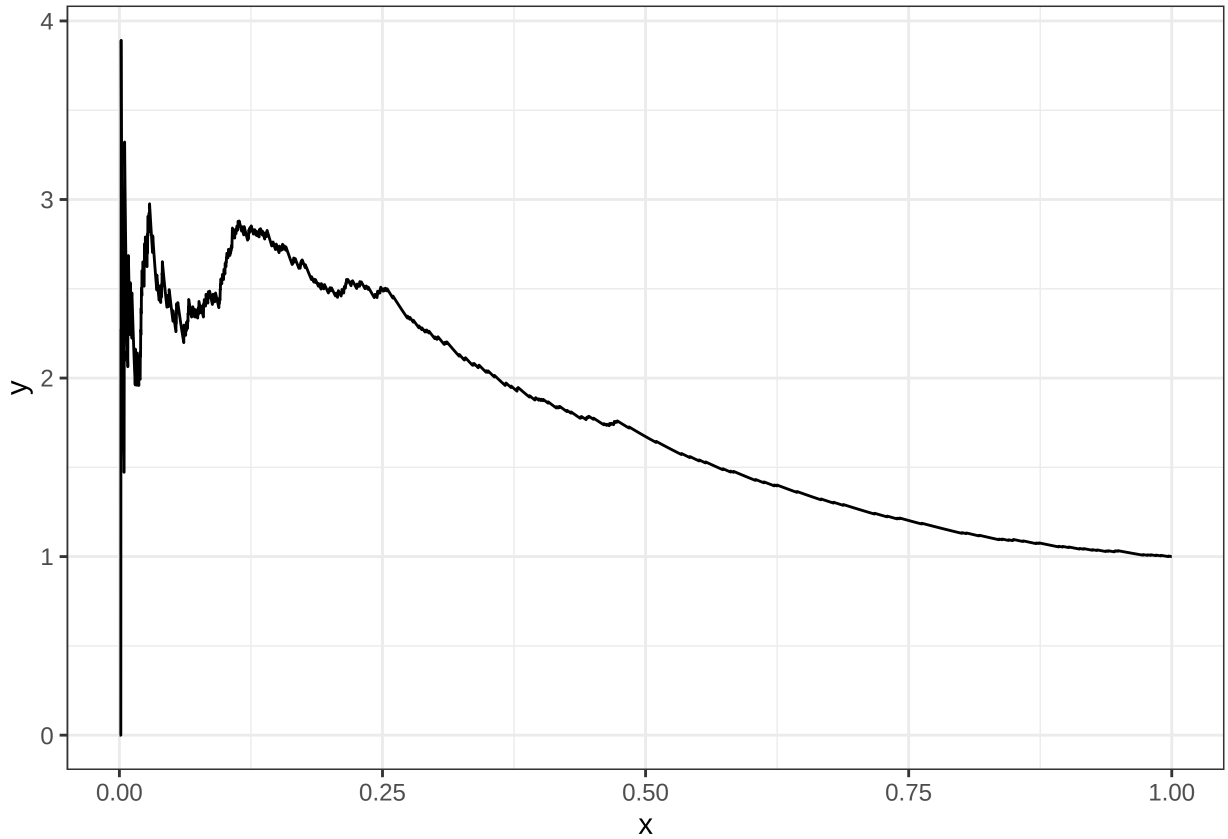
Plotting the lift curve helps us understand the performance of the decision tree model, as it focuses on positive class, which is the minority in this imbalanced dataset. we get a 2.5X improvement at 25% of the top selected test data.

```r
#create a new dataset based on testing data
music_data_test_cr = music_data_test %>%
mutate(prob = pred_tree[,2]) %>%

#sort the dataset above based on the descending order of the probability
arrange(desc(prob)) %>%

#adds two column x and y
# x indicates the accumulative proportion of data points take in the whole dataset
#y is cumulative proportion of positive cases as x changes
mutate(ado_yes = ifelse(adopter==1,1,0))%>%
mutate(x = row_number()/nrow(music_data_test),
       y = cumsum(ado_yes)/sum(ado_yes)/x)

#plot the lift curve
ggplot(data = music_data_test_cr, aes(x = x, y = y)) +
geom_line() +
theme_bw()
```



Exploring Best Features through KNN-Model

While decision tree already selects the best parameters, we also wanted to get an idea of the other possibly important parameters to help XYZ understand their data better. We did this through KNN. We used Recall and Precision to determine the importance of a parameter.

```
#create formula to normalze the data
normalize = function(x){
return ((x - min(x))/(max(x) - min(x)))}
# normalize
XYZ_norm = XYZData %>% mutate_at(2:26, normalize)

#train and test data
train_rows = createDataPartition(y = XYZ_norm$adopter, p=0.7, list = FALSE)
XYZ_train = XYZ_norm[train_rows,]
y = XYZ_train %>%
  filter(adopter == 1)
XYZ_train = rbind(XYZ_train,y,y,y,y,y,y)
XYZ_test = XYZ_norm[-train_rows,]
XYZ_test$adopter = as.factor(XYZ_test$adopter)
XYZ_train$adopter = as.factor(XYZ_train$adopter)

recall = c()
precision = c()

#create for loop to iterate over 26 columns
for(i in 2:26){
```

```r
  XYZtrain = XYZ_train[,-i]
  XYZtest = XYZ_test[,-i]
  pred_knn = knn(train = XYZtrain[,2:25],
                 test = XYZtest[,2:25],
                 cl = XYZtrain[,26,drop=TRUE],
                 k = 10)
  confusionMatrix(data = pred_knn,
                  reference = XYZtest$adopter,
                  positive = '1',
                  mode = "prec_recall")
  recall = c(recall, confusionMatrix(data = pred_knn,
                  reference = XYZtest$adopter,
                  positive = '1',
                  mode = "prec_recall")$byClass['Recall'])
  precision = c(precision, confusionMatrix(data = pred_knn,
                  reference = XYZtest$adopter,
                  positive = '1',
                  mode = "prec_recall")$byClass['Precision'])
}
recall
```

```
##    Recall    Recall    Recall    Recall    Recall    Recall    Recall    Recall
## 0.3272311 0.3157895 0.3135011 0.3226545 0.3272311 0.3180778 0.3089245 0.3066362
##    Recall    Recall    Recall    Recall    Recall    Recall    Recall    Recall
## 0.3226545 0.3157895 0.3157895 0.3089245 0.3203661 0.3135011 0.3135011 0.3226545
##    Recall    Recall    Recall    Recall    Recall    Recall    Recall    Recall
## 0.3318078 0.3043478 0.3089245 0.3135011 0.3043478 0.3157895 0.3157895 0.3203661
##    Recall
## 0.3226545
```

```r
precision
```

```
##  Precision  Precision  Precision  Precision  Precision  Precision  Precision
## 0.06155833 0.06052632 0.05912818 0.06000000 0.06338652 0.06017316 0.05831533
##  Precision  Precision  Precision  Precision  Precision  Precision  Precision
## 0.05900484 0.05964467 0.05958549 0.05882353 0.05796479 0.06018917 0.05874786
##  Precision  Precision  Precision  Precision  Precision  Precision  Precision
## 0.05887409 0.06005111 0.06041667 0.05755084 0.05851756 0.05935875 0.05767563
##  Precision  Precision  Precision  Precision
## 0.05955978 0.06052632 0.05985464 0.06067126
```

Here, we can remove the column with index 6 as it gives the best recall and precision, implying that the model performs well without this feature.

By repeating this process over and over, we selected the top 8 parameters with which we do not have significant information loss.