XYZ WEBSITE: PREDICTING POTENTIAL PREMIUM SUBSCRIBERS

2024-10-26

Loading Packages & Data

```r
set.seed(111)
library(rpart) #Decision tree
library(caret) # Model training
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(pROC) # ROC curve
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(rpart.plot)
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(ggplot2) # Visualization
xyz_data <- read.csv("XYZData.csv")
head(xyz_data)
```

```
##   user_id age male friend_cnt avg_friend_age avg_friend_male friend_country_cnt
## 1      10  24    0         20       26.33333       0.7777778                  6
## 2     137  22    1          4       21.25000       0.7500000                  1
## 3     306  18    0          3       18.50000       0.6666667                  1
## 4     336  24    1        131       23.90741       0.4310345                 22
## 5     355  23    1         15       21.55556       0.4615385                  2
## 6     382  22    1         13       21.90909       0.7692308                  3
##   subscriber_friend_cnt songsListened lovedTracks posts playlists shouts
## 1                     0         37804           4    20         1     47
## 2                     0           774           0     0         0      3
## 3                     0         14036           1     0         1     10
## 4                     4          3457         227     1         2    247
## 5                     1          7506           0    18         1     28
## 6                     1          2409          12     0         0      3
##   delta_friend_cnt delta_avg_friend_age delta_avg_friend_male
## 1                0           0.22222222            0.00000000
## 2                0           1.00000000            0.00000000
## 3                0           0.00000000            0.00000000
## 4                6           0.31517440            0.01285266
## 5                3           0.84126984           -0.03846154
## 6                1           0.09090909           -0.06410256
##   delta_friend_country_cnt delta_subscriber_friend_cnt delta_songsListened
## 1                        0                           0                  54
## 2                        0                           0                   0
## 3                        0                           0                   0
## 4                        0                           2                 865
## 5                        0                           0                  -4
## 6                        1                           0                 630
##   delta_lovedTracks delta_posts delta_playlists delta_shouts tenure
## 1                 0           0               0            0     79
## 2                 0           0               0            0     60
## 3                 0           0               0            0     41
## 4                 7           0               0            4     79
## 5                 0           0               0            6     70
## 6                 3           0               0            1     10
##   good_country delta_good_country adopter
## 1            0                  0       0
## 2            0                  0       0
## 3            1                  0       0
## 4            0                  0       0
## 5            0                  0       0
## 6            0                  0       0
```

```r
#Checking total no. of nulls
sum(is.na(xyz_data))
```

```
## [1] 0
```

Train-Test Split

We are splitting previous year campaign data into 70% training and 30% testing datasets for our model, instead of following the standard 80-20 split. This is because of the following reasons:

1. In cases of imbalanced datasets, having a larger test set ensures that we capture enough instances of the minority class(which is adopters in a freemium model). This will help us better understand how well our model performs for the minority class and can accordingly improve perfomance metrics

2.This can also help ensure that your model's evaluation metrics are not overly optimistic and that the model will generalize better in production which might reduce the chance of overfitting

```r
train_rows <- createDataPartition(xyz_data$adopter, p = 0.7, list = FALSE)
train_set <- xyz_data[train_rows, ]
test_set <- xyz_data[-train_rows, ]

# Need to ensure the target variable (adopter) is a factor
train_set$adopter <- as.factor(train_set$adopter)
levels(train_set$adopter) <- c("No", "Yes")
```

Base Model Training & Feature Importance

In this model, we aim to predict the minority class i.e. users who had opted for premium subscription, and decision trees are a good choice for this task. They effectively handle imbalanced datasets by focusing on key features that distinguish the minority class, even when it appears less frequently. Hence we decided to try out decision trees for our problem statement.

Decision trees are also generally better at handling outliers compared to other models like Naive Bayes as they are not affected by the scale of the data, and they split based on feature values that best separate the classes

We did initially try Naive Bayes model at first but finalised decision tree based on model performance and other issues with naive bayes such as handling outliers which are discussed more towards the end

Defining function to check model performance

```r
f1_summary <- function(data, lev = NULL, model = NULL) {
  positive_class<- "Yes"
  # Calculate Precision
  precision <- posPredValue(data$pred, data$obs, positive = positive_class)

  # Calculate Recall (Sensitivity)
  recall <- sensitivity(data$pred, data$obs, positive = positive_class)

  # Calculate F1 Score
```

```
    f1 <- (2 * precision * recall) / (precision + recall)  # Avoid division by zero

  # Return as a named vector for caret to use
  c(Precision = precision, Recall = recall, F1 = f1)
}
```

Base model Training & Performance evaluation

```
# Using train_control() function as it is used to specify the settings and parameters for training mach
train_control <- trainControl(method = "cv", #Applying 10 fold cross-validation
                              number = 10,
                              sampling = "down", # Sampling method selected
                              summaryFunction = f1_summary,
                              savePredictions = "all")

#Training a decision tree model with cross-validation
decision_tree_cv_model <- train(adopter ~ .,
                                data = train_set,
                                method = "rpart",
                                trControl = train_control,
                                metric = "Recall")
cv_results <- decision_tree_cv_model$resample  # Access resample statistics
print(paste("Mean Recall scores:", mean(cv_results$Recall)))
```

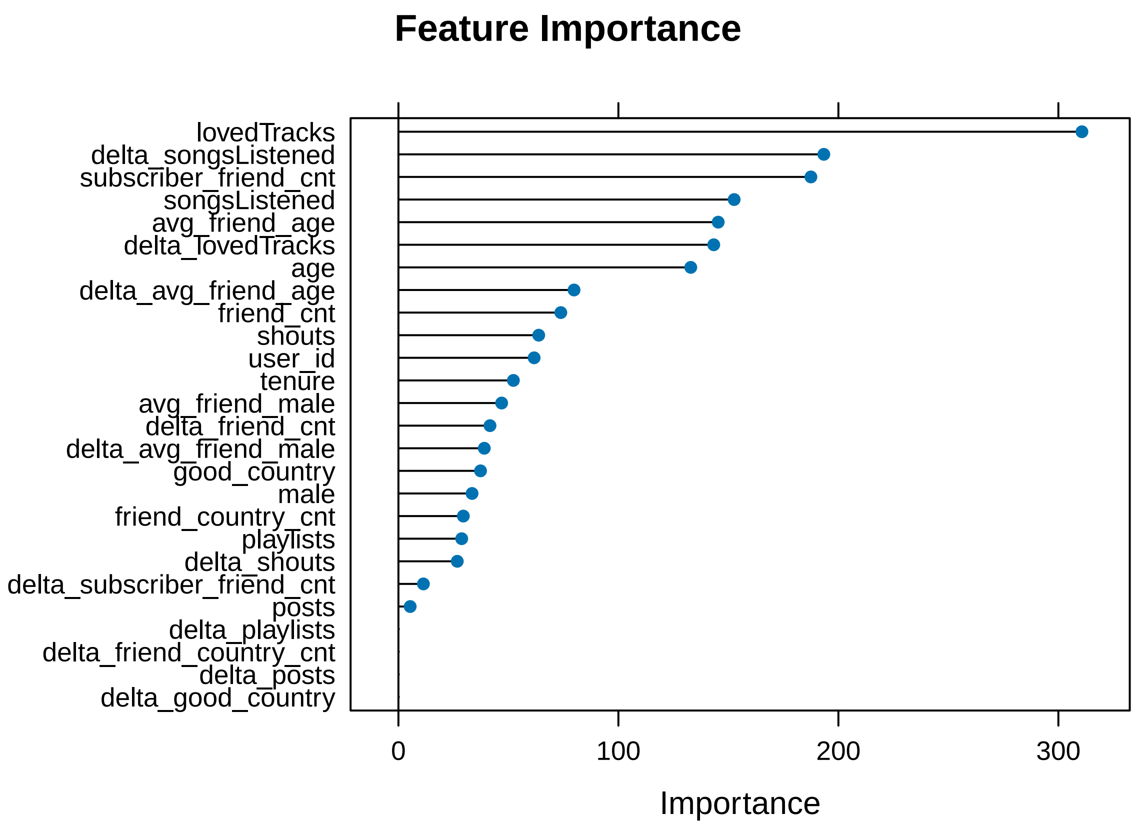
```
## [1] "Mean Recall scores: 0.687524131274131"
```

```
print(paste("Mean F1 scores:", mean(cv_results$F1)))
```

```
## [1] "Mean F1 scores: 0.137756425491424"
```

Feature Importance: Identifying features with higher importance scores as they have a greater impact on classification decisions

```
importance <- varImp(decision_tree_cv_model, scale = FALSE)
# Plotimportance for visualization
plot(importance, main = "Feature Importance")
```



Feature Selection: Only keeping the top 20 features for final model training

```
test_set_user <-test_set %>%select(c(user_id))
test_set_filt <- test_set %>%select(c(adopter,
                                      delta_songsListened,
                                      lovedTracks,
                                      delta_lovedTracks,
                                      friend_country_cnt,
                                      subscriber_friend_cnt,
                                      male,
                                      friend_cnt,
                                      delta_shouts,
                                      shouts,
                                      delta_avg_friend_male,
                                      delta_friend_cnt,
                                      avg_friend_age,
                                      delta_posts,
                                      delta_friend_country_cnt,
                                      delta_avg_friend_age,
                                      posts,
                                      age,
                                      tenure,
                                      songsListened))
```

```r
train_set_user <-train_set %>%select(c(user_id))
train_set_filt <- train_set %>%select(c(adopter,
                                      delta_songsListened,
                                      lovedTracks,
                                      delta_lovedTracks,
                                      friend_country_cnt,
                                      subscriber_friend_cnt,
                                      male,
                                      friend_cnt,
                                      delta_shouts,
                                      shouts,
                                      delta_avg_friend_male,
                                      delta_friend_cnt,
                                      avg_friend_age,
                                      delta_posts,
                                      delta_friend_country_cnt,
                                      delta_avg_friend_age,
                                      posts,
                                      age,
                                      tenure,
                                      songsListened))
```

Model Training & Hyper parameter Tuning

We used 5 fold cross validation instead of 10 fold so that we will get atleast 20% data from the training sample to be used for testing in the cross-validation. This is to ensure that we get a decent size of testing data as our overall data size is limited

Selected the F1 score metric in how the decision tree model is evaluated and optimized during the training process because even though it will help improve recall (i.e. how well our model captures the minority class),but also keeping false positives in check(not classifying too many non-adopters as adopters).

Performing Hyperparameter Tuning on cp (complexity parameter - controls the pruning of the tree).We tested 10 values between 0 and 0.05 to find the optimal level of pruning that maximizes model performance

Sampling method selection:

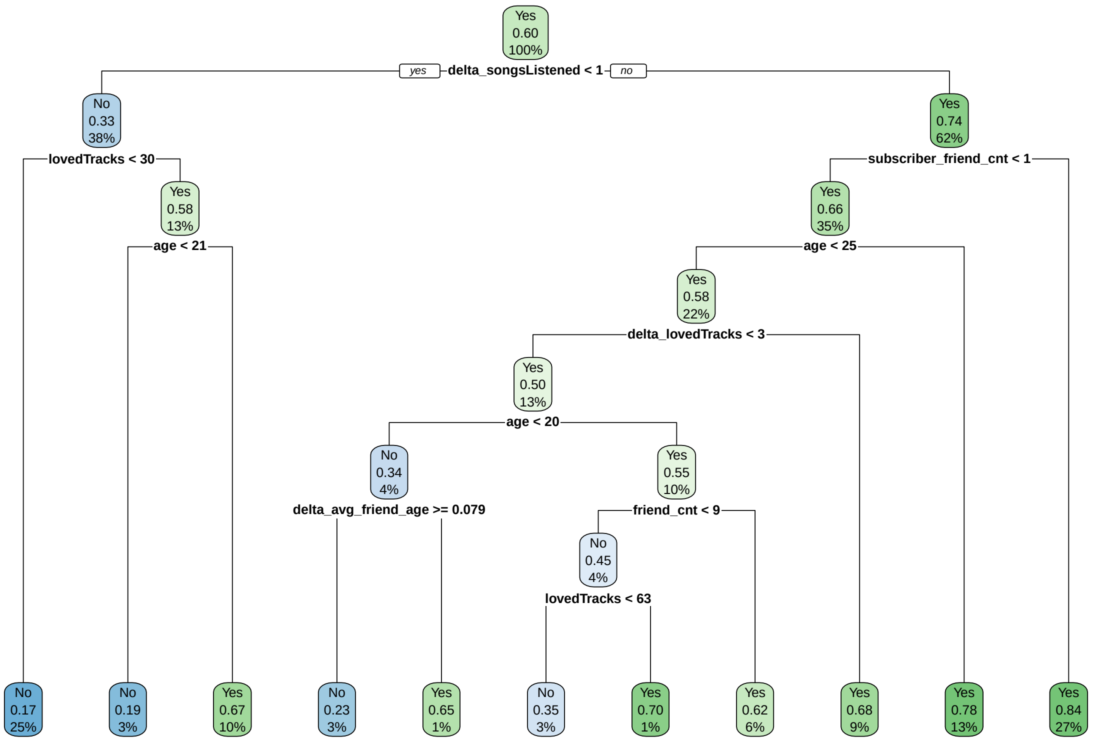
We used downsampling for the final model because upsampling was increasing the number of false positives as compared to "down". When predicting a minority class (e.g., adopters) in an imbalanced dataset, upsampling can sometimes create too many synthetic or duplicated minority class instances, which may cause the model to become overly biased toward the minority class, leading to higher false positive rates.This issue was clearly visible in our model making us choose upsampling over downsampling

```r
train_control <- trainControl(method = "cv",
                              number = 5,  # 5-fold cross-validation
                              classProbs = TRUE,
                              summaryFunction = f1_summary,
                              sampling = "down",
                              )

#Train a decision tree model with cross-validation and hyperparameter tuning
decision_tree_cv_model <- train(adopter ~ .,
                                data = train_set_filt, # using the training data subset with top 20 fea
                                method = "rpart",
                                trControl = train_control,
                                metric='F1', # selected method for model optimization as explained abov
                                parms = list(prior = c(0.4, 0.6)),
                                tuneGrid = data.frame(cp = seq(0, 0.05, length = 10)))# Tune Grid : to 


ptree <- (decision_tree_cv_model$finalModel)

# Plot the decision tree
rpart.plot(ptree)
```



Model Evaluation

The performance metric that we selected to evaluate the model was primarily recall score.

Recall Score = (True Positive)/(True Positive+False Negative)

The focus of our entire model is to identify users who might opt for premium subscription if targeted, which aligns with recall score as it helps us get the % of users predicted to convert and actually converted out of all the users who actually converted(i.e. opted for premium subscription)

```r
# Display the k-fold cross-validation results (k=5)
print(decision_tree_cv_model)
```

```
## CART
##
## 29078 samples
##    19 predictor
##     2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 23262, 23262, 23263, 23263, 23262
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   cp           Precision   Recall     F1
##   0.000000000  0.06568826  0.7531491  0.1208257
##   0.005555556  0.06975067  0.8581344  0.1289657
##   0.011111111  0.06452704  0.8940654  0.1202818
##   0.016666667  0.06286511  0.9102371  0.1175969
##   0.022222222  0.06286511  0.9102371  0.1175969
##   0.027777778  0.06073292  0.9254838  0.1139769
##   0.033333333  0.06073292  0.9254838  0.1139769
##   0.038888889  0.06073292  0.9254838  0.1139769
##   0.044444444  0.06168379  0.8976811  0.1153856
##   0.050000000  0.06168379  0.8976811  0.1153856
##
## F1 was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.005555556.
```

```r
cv_results <- decision_tree_cv_model$resample  # Access resample statistics

#Best evaluation metric scores at k-fold (k=5)
print(paste(" Recall score:", mean(cv_results$Recall)))
```

```
## [1] " Recall score: 0.858134367551408"
```

```r
print(paste(" F1 score:", mean(cv_results$F1)))
```

```
## [1] " F1 score: 0.128965670415757"
```

Model Testing : Testing the unseen 30% data

```r
#Ensuring the test - target variable (adopter) is a factor
test_set_filt$adopter <- as.factor(test_set_filt$adopter)
levels(test_set_filt$adopter) <- c("No", "Yes")

#Making predictions on the test set
test_pred <- predict(decision_tree_cv_model, test_set_filt)
```

Test Model Predictions & Evaluation

Confusion Matrix

```r
test_pred <- factor(test_pred)                # Ensuring predictions are factors
conf_matrix <- confusionMatrix(test_pred, test_set_filt$adopter, positive = 'Yes')
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  6309   47
##        Yes 5727  379
##
##                Accuracy : 0.5367
##                  95% CI : (0.5279, 0.5455)
##     No Information Rate : 0.9658
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0557
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.88967
##             Specificity : 0.52418
##          Pos Pred Value : 0.06207
##          Neg Pred Value : 0.99261
##              Prevalence : 0.03418
##          Detection Rate : 0.03041
##    Detection Prevalence : 0.48997
##       Balanced Accuracy : 0.70692
##
##        'Positive' Class : Yes
##
```

```r
precision <- conf_matrix$byClass["Pos Pred Value"]
recall <- conf_matrix$byClass["Sensitivity"]
f1 <- (2 * precision * recall) / (precision + recall)
print(paste("Recall score for our test data at default probability cut-off 0.5:", round(recall,2)))
```

```
## [1] "Recall score for our test data at default probability cut-off 0.5: 0.89"
```

```r
print(paste("F1 score for our test data at default probability cut-off 0.5:", round(f1,2)))
```

```
## [1] "F1 score for our test data at default probability cut-off 0.5: 0.12"
```

‘

Iterating through probability cutoff to find the threshold giving us optimal recall score

```r
# Geting the predicted probabilities from the model
predictions <- predict(decision_tree_cv_model,test_set_filt,type='prob')[,2]

#We defined the custom probability cutoff to be 0.4 for out model as it improved our recall score witho
prob_cutoff <- 0.4

# Created new predictions based on the adjusted cutoff
predicted_classes <- ifelse(predictions > prob_cutoff, "Yes", "No")

# Evaluate performance (e.g., calculate confusion matrix, Precision, Recall, F1 score)
confusion <- confusionMatrix(as.factor(predicted_classes), test_set_filt$adopter, positive = "Yes")

# Print the confusion matrix
print(confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  6309   47
##        Yes 5727  379
##
##                Accuracy : 0.5367
##                  95% CI : (0.5279, 0.5455)
##     No Information Rate : 0.9658
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0557
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.88967
##             Specificity : 0.52418
##          Pos Pred Value : 0.06207
##          Neg Pred Value : 0.99261
##              Prevalence : 0.03418
##          Detection Rate : 0.03041
##    Detection Prevalence : 0.48997
##       Balanced Accuracy : 0.70692
##
##        'Positive' Class : Yes
##
```

```
# Extract F1 score, Precision, and Recall from the confusion matrix
precision <- confusion$byClass["Pos Pred Value"]
recall <- confusion$byClass["Sensitivity"]
f1 <- (2 * precision * recall) / (precision + recall)

print(paste("Recall at probability cut-off 0.4:", round(recall,2)))
```

```
## [1] "Recall at probability cut-off 0.4: 0.89"
```

```
print(paste("F1 Score at probability cut-off 0.4:", round(f1,2)))
```

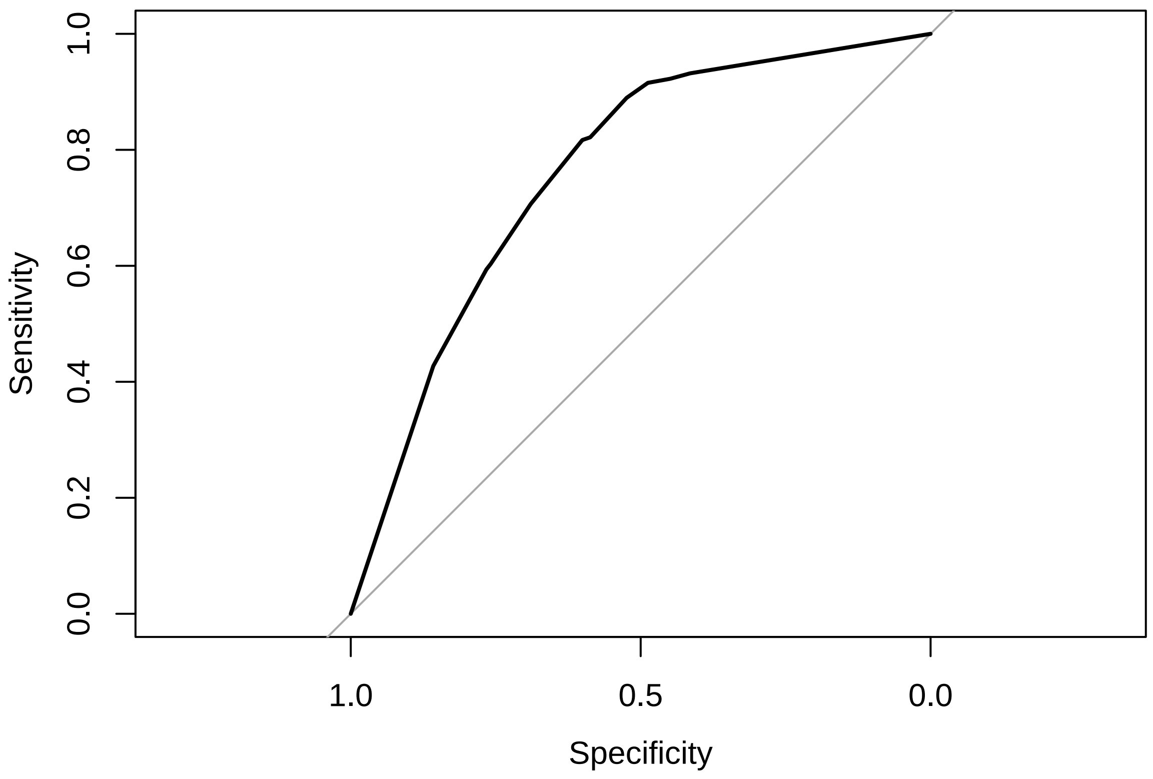
```
## [1] "F1 Score at probability cut-off 0.4: 0.12"
```

```
# Probability predictions
prob_pred_tree = predict(decision_tree_cv_model, test_set_filt, type = "prob")
xyz_test_prob= test_set_filt %>%
  mutate(prob = prob_pred_tree[,"Yes"]) %>%
  arrange(desc(prob)) %>%
  mutate(acc_yes = ifelse(adopter=="Yes",1,0))
```

ROC Curve for Premium Subscription Adopters

```
xyz_test_roc = xyz_test_prob %>%mutate(TPR = cumsum(acc_yes)/sum(acc_yes),FPR = cumsum(1-acc_yes)/sum(1

roc_t = roc(response = xyz_test_roc$acc_yes,
             predictor = xyz_test_roc$prob)
plot(roc_t)
```



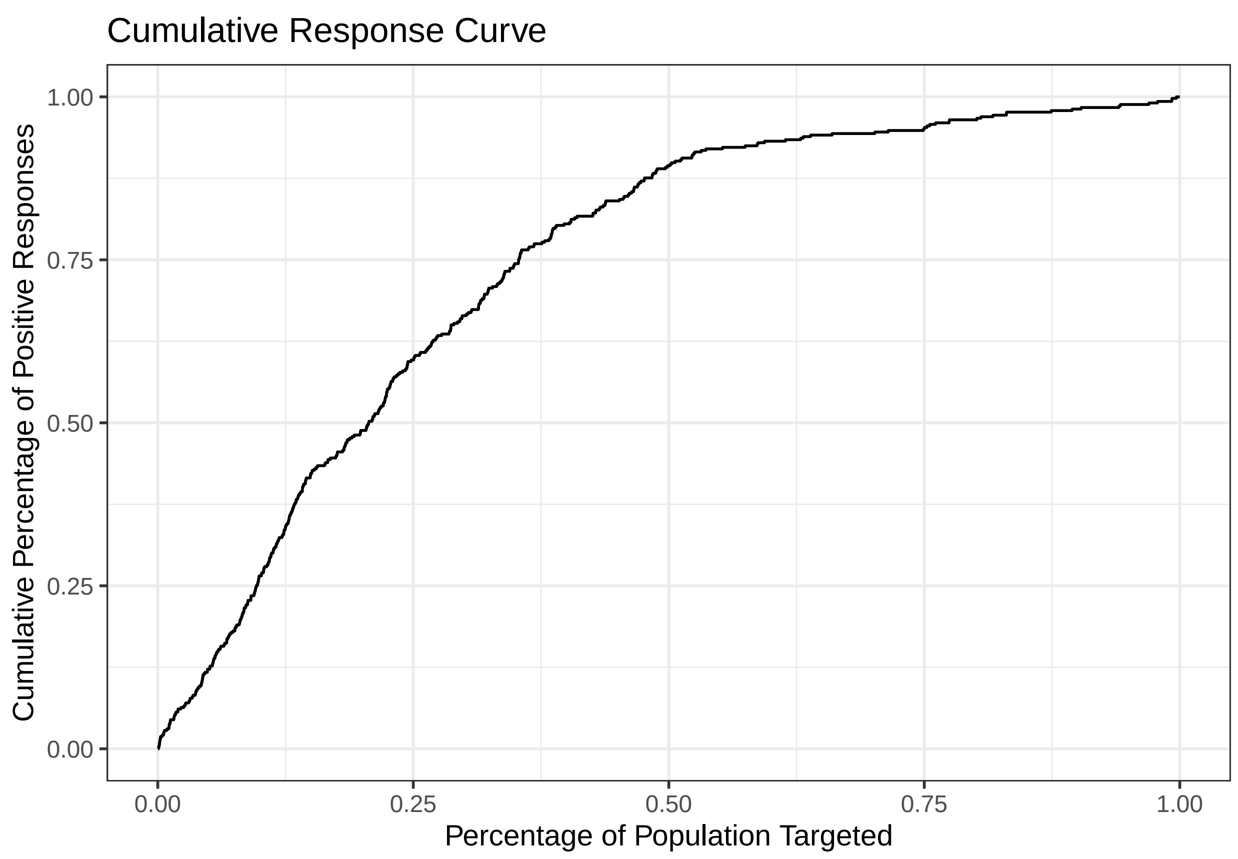
```r
auc(roc_t)
```

```
## Area under the curve: 0.7604
```

Cumulative Response Curve for Predictions - helps evaluate the effectiveness of the model in identifying positive responses

```r
xyz_test_cr = xyz_test_prob %>%mutate(y = cumsum(acc_yes)/sum(acc_yes),x = row_number()/nrow(test_set_f

# Cumulative response curve plot with axis
ggplot(data = xyz_test_cr, aes(x = x, y = y)) +
  geom_line() +
  theme_bw() +
  labs(
    x = "Percentage of Population Targeted",    # X-axis title
    y = "Cumulative Percentage of Positive Responses",   # Y-axis title
    title = "Cumulative Response Curve"
  )
```



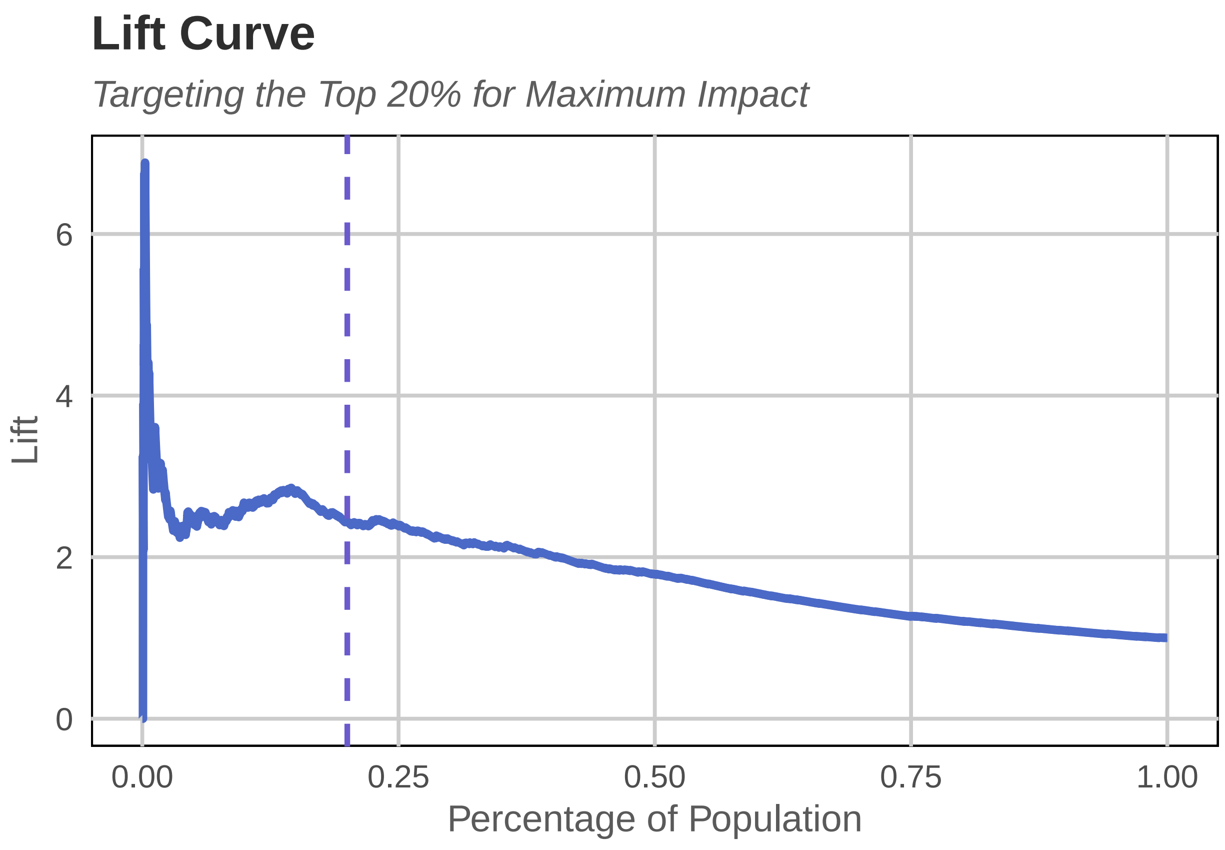
Lift Curve - evaluates the performance of your model in targeting the population based on predicted probabilities, compared to a random model

```
xyz_test_lift =xyz_test_prob %>%
  # the following two lines make the lift curve
  mutate(x = row_number()/nrow(test_set_filt),
         y = (cumsum(acc_yes)/sum(acc_yes))/x)

# Updated ggplot code with modern aesthetics
ggplot(data = xyz_test_lift, aes(x = x, y = y)) +
  geom_line(color = "#4B69C6", size = 1.5) +  # Blue line with thicker width
  theme_minimal(base_size = 15) +  # Modern minimalistic theme with base font size
  labs(title = "Lift Curve",
       subtitle = "Targeting the Top 20% for Maximum Impact",
       x = "Percentage of Population",
       y = "Lift") +  # Add titles and axis labels
  theme(
    plot.title = element_text(face = "bold", size = 18, color = "#2E2E2E"),
    plot.subtitle = element_text(face = "italic", size = 14, color = "#5D5D5D"),
    axis.title.x = element_text(size = 14, color = "#5A5A5A"),
    axis.title.y = element_text(size = 14, color = "#5A5A5A"),
    axis.text = element_text(size = 12, color = "#4D4D4D"),
    panel.grid.major = element_line(color = "gray80"),  # Light grid lines
    panel.grid.minor = element_blank(),  # Remove minor grid lines
    plot.background = element_rect(fill = "white", color = "white"),  # White background
    panel.background = element_rect(fill = "white")  # White panel background
  ) +
  geom_vline(xintercept = 0.20, linetype = "dashed", color = "#6A5ACD", size = 1) +  # Highlight 20%
  geom_text(aes(x = 0.20, y = 2.5, label = ""),
            vjust = -1, color = "#6A5ACD", size = 5)  # Annotate the lift at 2.5x
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning in geom_text(aes(x = 0.2, y = 2.5, label = ""), vjust = -1, color = "#6A5ACD", : All aesthet
## i Please consider using ‘annotate()‘ or provide this layer with data containing
##   a single row.
```



Creating final dataset with predictions and userids

```
final_prediction_data <-test_set_filt
final_prediction_data$predicted_adopter <-predicted_classes
```

```
final_prediction_data$user_id <-test_set_user$user_id
head(final_prediction_data)
```

```
##    adopter delta_songsListened lovedTracks delta_lovedTracks friend_country_cnt
## 1       No                  54           4                 0                  6
## 3       No                   0           1                 0                  1
## 5       No                  -4           0                 0                  2
## 11      No                 380          53                 0                 15
## 28      No                   0         128                 0                  8
## 34      No                 276          24                 0                 11
##    subscriber_friend_cnt male friend_cnt delta_shouts shouts
## 1                      0    0         20            0     47
## 3                      0    0          3            0     10
## 5                      1    1         15            6     28
## 11                     4    1         84            0    102
## 28                     0    1         21            0     25
## 34                     1    0         20            0     38
##    delta_avg_friend_male delta_friend_cnt avg_friend_age delta_posts
## 1            0.000000000                0       26.33333           0
## 3            0.000000000                0       18.50000           0
## 5           -0.038461538                3       21.55556           0
## 11          -0.010371820                5       24.63333           0
## 28           0.000000000                0       26.80000           0
## 34           0.008333333                1       33.71429           0
##    delta_friend_country_cnt delta_avg_friend_age posts age tenure songsListened
## 1                         0            0.2222222    20  24     79         37804
## 3                         0            0.0000000     0  18     41         14036
## 5                         0            0.8412698    18  23     70          7506
## 11                        0            0.3297619    23  23     65         22540
## 28                        0            0.2000000     1  24     61         15957
## 34                        0            0.2527473     0  29     64         18693
##    predicted_adopter user_id
## 1                Yes      10
## 3                 No     306
## 5                 No     355
## 11               Yes     575
## 28               Yes    1658
## 34               Yes    1719
```

Additional Technique tried initially - NAIVE BAYES

```
#Importing data
xyz_data_nb= read.csv("XYZData.csv")
```

Outlier Treatment

Naive Bayes can be really sensitive outliers which made it neccessary for us to treat outliers before proceeding with the model

However, removing all outliers will also remove all of our premium subscribers. Hence we proceeded to build the model with the original data as removing even some portion of outliers based on other techniques will make us loose a large portion of our already small no. of premium subscribers

```r
#outlier treatment
# Function to identify outliers based on IQR
identify_outliers <- function(column) {
Q1 <- quantile(column, 0.25)
Q3 <- quantile(column, 0.75)
IQR <- Q3 - Q1
lower_bound <- Q1 - 2 * IQR
upper_bound <- Q3 + 2 * IQR
return(column < lower_bound | column > upper_bound)
}
outliers_matrix <- sapply(xyz_data_nb, identify_outliers)
# Combine the outlier flags across all columns (any TRUE indicates a row with outliers)
rows_with_outliers <- apply(outliers_matrix, 1, any)
# Remove rows with outliers
xyz_data_clean <- xyz_data_nb[!rows_with_outliers, ]
adopter_cnt <-xyz_data_clean %>%group_by(adopter) %>%
  summarise(count = n())
adopter_cnt
```

```
## # A tibble: 1 x 2
##   adopter count
##     <int> <int>
## 1       0 15996
```

Model Train-Test Split & Training

```r
library(e1071)
train_rows = createDataPartition(y = xyz_data_nb$adopter, p = 0.80, list = FALSE)
d_train = xyz_data_nb[train_rows,]
d_test = xyz_data_nb[-train_rows,]
d_train$adopter <- as.factor(d_train$adopter)
d_test$adopter <- as.factor(d_test$adopter)
levels(d_train$adopter) <- c("No", "Yes")
levels(d_test$adopter) <- c("No", "Yes")

NB_model = naiveBayes(adopter ~ ., data = d_train) # Training with Naive Bayes
```

Model Prediction & Evaluation

The base model predictions on test data gave extremely low recall score as compared to the decision tree which had a much better score in the base model(before tuning). This along with other aspects of this model such as sensitivity to outliers helped in taking the decision of moving to decision trees instead

```r
# Make predictions on the test set
predictions <- predict(NB_model, d_test)

# Display the confusion matrix to evaluate the model
conf_matrix <- confusionMatrix(predictions, d_test$adopter,positive = 'Yes')
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  7650  246
##        Yes  362   50
##
##                Accuracy : 0.9268
##                  95% CI : (0.921, 0.9323)
##     No Information Rate : 0.9644
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1041
##
##  Mcnemar's Test P-Value : 3.103e-06
##
##             Sensitivity : 0.168919
##             Specificity : 0.954818
##          Pos Pred Value : 0.121359
##          Neg Pred Value : 0.968845
##              Prevalence : 0.035628
##          Detection Rate : 0.006018
##    Detection Prevalence : 0.049591
##       Balanced Accuracy : 0.561868
##
##        'Positive' Class : Yes
##
```

```r
recall <- conf_matrix$byClass["Sensitivity"]
print(paste("Recall Score", round(recall,2)))
```

```
## [1] "Recall Score 0.17"
```