Predictive Analysis: Classification and Numeric Prediction

2023-07-23

GROUP PART – technical document

Table of contents:

Things to Notice before Analysis

Don't use Accuracy in Imbalanced dataset; Use AUC, Precision and Recall_p

From EDA later we will know the proportion of 2 levels in response variable is severely imbalanced, indicating we shouldn't use accuracy as evaluation standard since it will conclude incorrect results.

Thus, we select models based on AUC with metrics **precision** and **recall_positive** because:

1. **higher recall_p allows targeting as many RIGHT users as possible with less budget**, and
2. **precision focuses on customers being targeted really convert to premium users.**

However, we are going to be not so restricted: if there's no really big difference in those metrices, we suggest more explainable model in business point of view.

Understanding the scope of our analysis: prediction task, not causal analysis

Note that all the data are originally non-subscribers. It is essential to emphasize our goal for understanding which people would be likely to convert from free users to premium subscribers in the next 6 month period if they are targeted by our promotion campaign. We care about correlation and can't say "they turn into premium users DUE TO our promotion" since it's NOT an causal problem which statistical method we have now cannot solve.

So please notice the description here: **we are NOT going to use CAUSE, DUE TO…, we say RELATED.**

Problem defining and our overall rationale to solve it

General goal for analysis

To get a deeper understanding of which people would be likely to convert from free users to premium subscribers in the next 6 month period if they are targeted by our promotion campaign.

Specified goal for analysis

1. we conduct EDA before moving on to further analysis, and we are going to combine analysis from that as well as model selection to provide data support for business solutions.
2. **model fitting and performance evaluation:**
   - **(1) fit the models with the normalized, selected features**
   - **(2) 10-folds cross-validation with oversampled training set within each folds**
   - **(3) select the better model on AUC;**
     if similar, go to (4) and pick a better recall_p- precision combination
   - **(4) generate dashboard of thresholds and their corresponding precision and recall_p**
3. we will make suggestion based on analyzing results, and more detailed business strategies will be presented in our managerial document.

We are going to present this figure in the beginning and go through the whole procedure for selecting our best model - **Logistic regression with oversampled training set + PCA or top 10 features.**

Metric / Model types	Oversampled training set	Feature selection	AUC	Recall_positive	Precision	Relative advantages
Logistic regression (Best performance among all)	Y	PCA (dimension reduction)	**0.72**	**0.88** (Ranges with threshold)	**0.06**	Less dimensions but can keep all the factors: More easily explainable, not dropping information
	Y	Filter (top 10 features)	**0.74**	**0.89** (Ranges with threshold)	**0.06**	Being able to focus on top 10 important variables
	N	N	0.71	0.78 (Ranges with threshold)	0.06	N
K-NN model	Y	Filter (top 10 features)	0.70	0.44	0.08	N
Decision tree	Y	Filter (top 10 features)	0.61	0.30	0.11	N
Naïve Bayes model	Y	Filter (top 10 features)	0.71	(Ranges with threshold)	Always around 0.034	N

EDA before Prediction

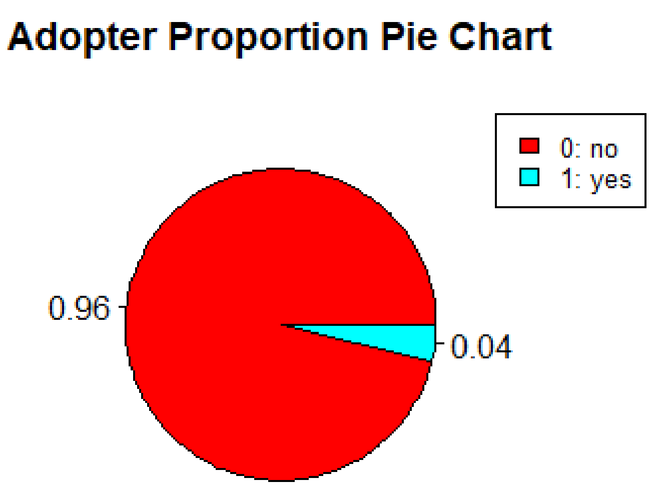
```r
setwd('C:/Users/Yvonne/Desktop/UMN Courses/6131')
library(dplyr)
library(ggplot2)
library(caret)
library(pROC)
library(ROSE)

## Loaded ROSE 0.0-4

xyzdata <- read.csv('XYZData.csv')
```

Check missing values: no missing values

```r
sum(is.na(xyzdata == TRUE)) # no missing values

## [1] 0
```

Check class proportion: imbalanced data

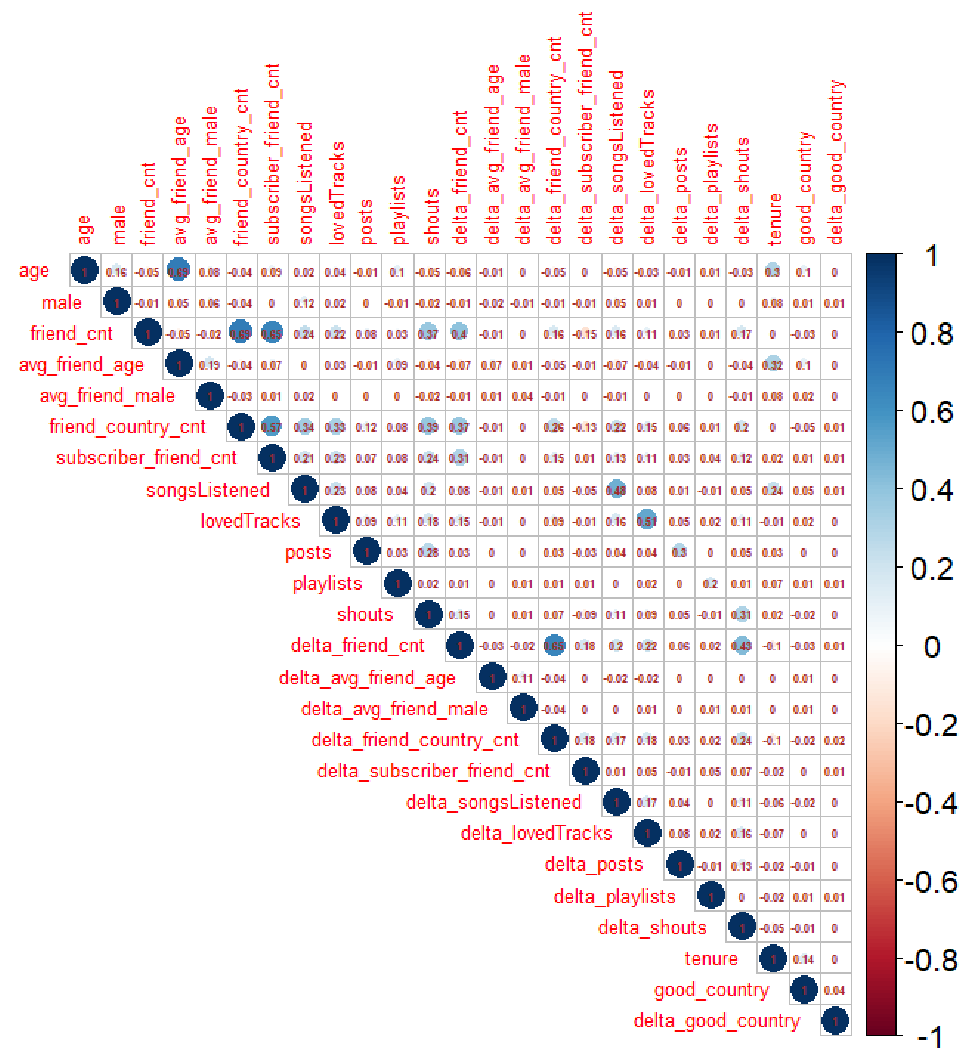
```r
# pie chart for the response variable
pie(table(xyzdata$adopter), labels = round(table(xyzdata$adopter)/41540, 2), main = "A
dopter Proportion Pie Chart", col = rainbow(2))
legend("topright", c("0: no","1: yes"), cex = 0.8, fill = rainbow(2))
```



Due to the severe imbalanced data, oversampled training set to train the model is needed.

Check the correlation among variables to get a conceptual understanding of the dataset.

```
library(corrplot)

## corrplot 0.92 loaded

corrplot(cor(xyzdata[, 2:26]), type = 'upper', addCoef.col = 'brown', tl.cex = 0.5, nu
mber.cex = 0.3)
```



No significant negative correlation but some positive correlations we might want to notice later.

The followings are some reasons we consider why that positive correlation happened:

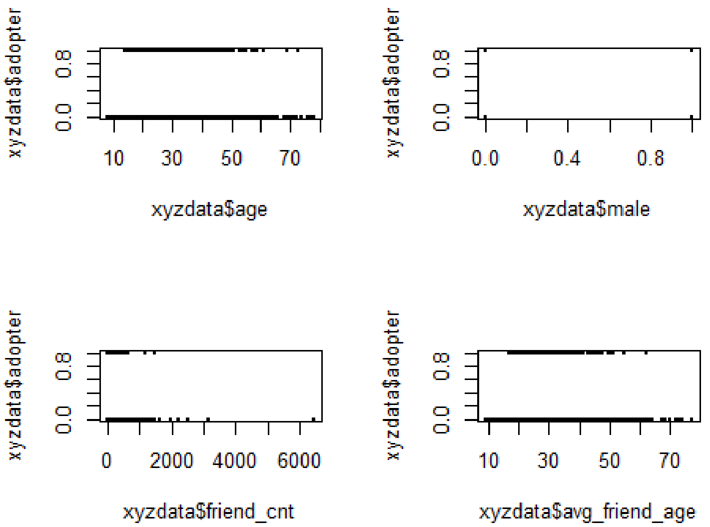
1. age / avg_friend_age: people generally tends to have friends with similar age range.

2. friend_cnt / friend_country_cnt: a person with more friends tends to have more friends from more different countries

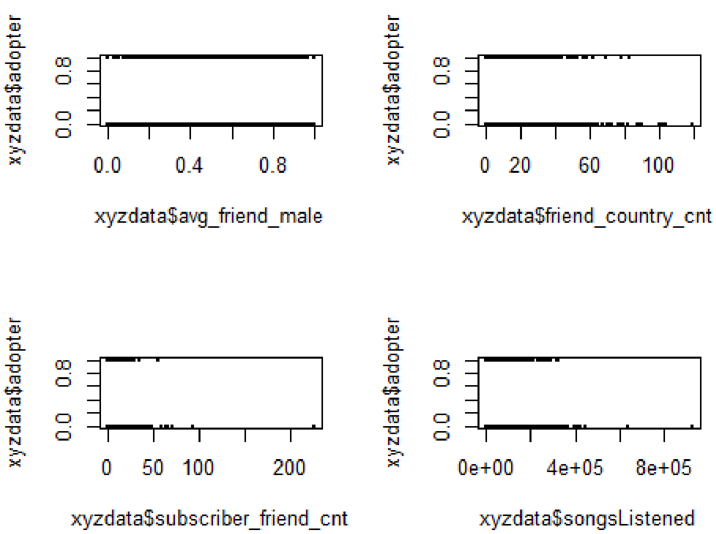
3. friend_cnt / subscriber_friend_cnt: a person with more friends tends to have more friends that are subscribers.

Check relationship between response and each predictor.

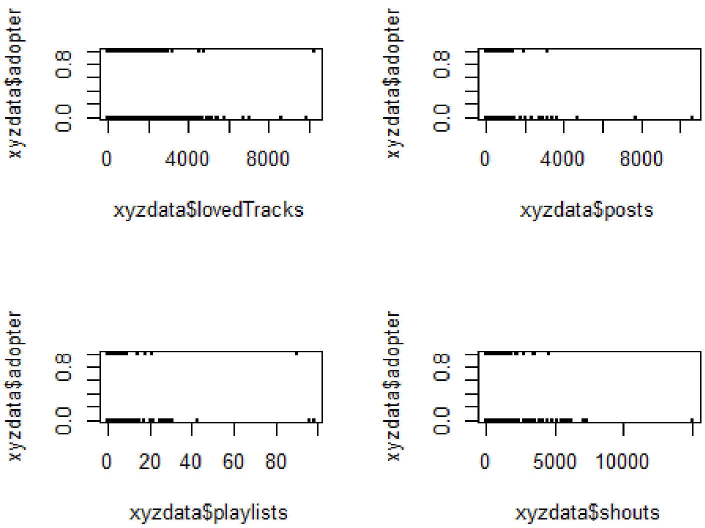
```r
par(mfrow = c(2, 2))
plot(xyzdata$age, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$male, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$friend_cnt, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$avg_friend_age, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
```



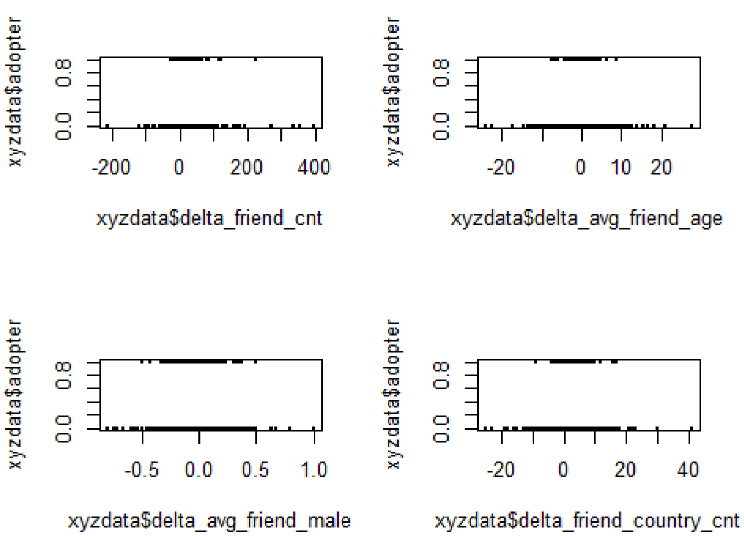

```r
par(mfrow = c(2, 2))
plot(xyzdata$avg_friend_male, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$friend_country_cnt, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$subscriber_friend_cnt, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$songsListened, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
```



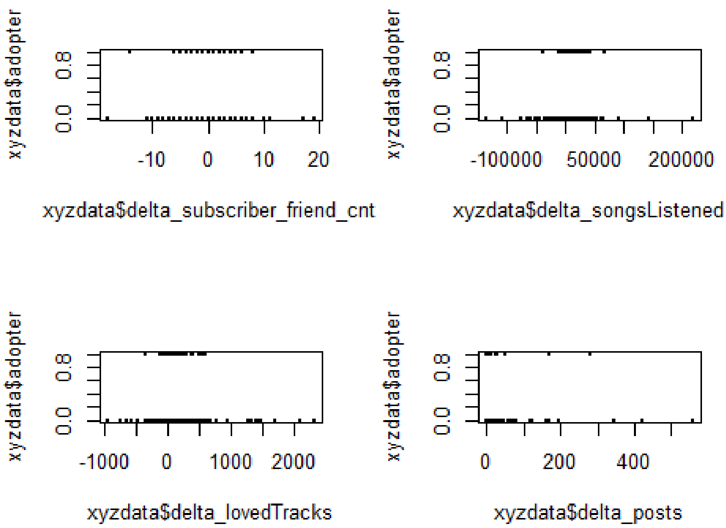
```r
par(mfrow = c(2, 2))
plot(xyzdata$lovedTracks, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$posts, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$playlists, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$shouts, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
```



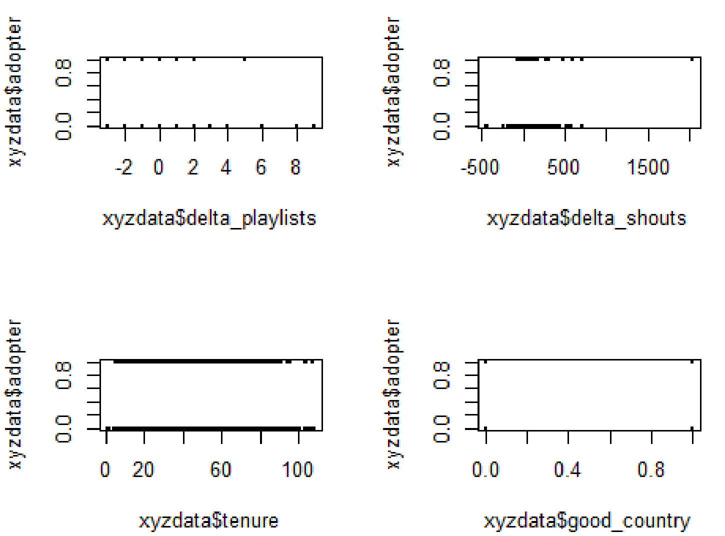

```r
par(mfrow = c(2, 2))
plot(xyzdata$delta_friend_cnt, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$delta_avg_friend_age, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$delta_avg_friend_male, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$delta_friend_country_cnt, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
```



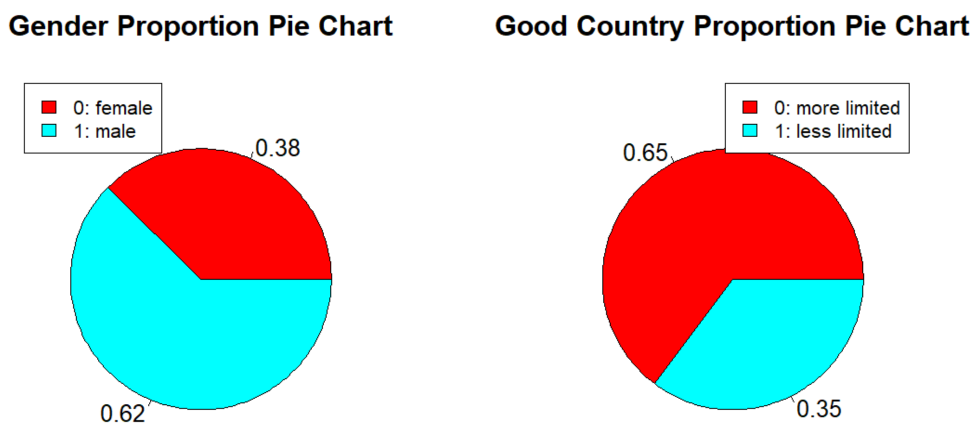
```r
par(mfrow = c(2, 2))
plot(xyzdata$delta_subscriber_friend_cnt, xyzdata$adopter, type = "p", pch = 20, cex =
 0.5)
plot(xyzdata$delta_songsListened, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$delta_lovedTracks, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$delta_posts, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
```



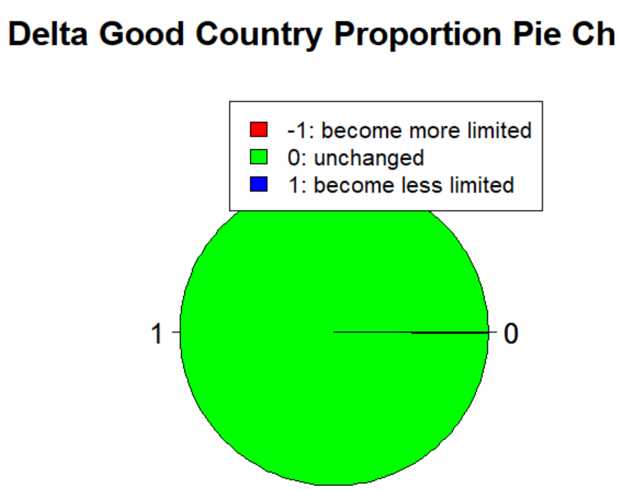

```r
par(mfrow = c(2, 2))
plot(xyzdata$delta_playlists, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$delta_shouts, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$tenure, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
plot(xyzdata$good_country, xyzdata$adopter, type = "p", pch = 20, cex = 0.5)
```



```
par(mfrow = c(1, 2))
pie(table(xyzdata$male), labels = round(table(xyzdata$male)/41540, 2), main = "Gender
Proportion Pie Chart", col = rainbow(2))
legend("topright", c("0: female","1: male"), cex = 0.8, fill = rainbow(2))

pie(table(xyzdata$good_country), labels = round(table(xyzdata$good_country)/41540, 2),
 main = "Good Country Proportion Pie Chart", col = rainbow(2))
legend("topright", c("0: more limited","1: less limited"), cex = 0.8, fill = rainbow
(2))
```




```
pie(table(xyzdata$delta_good_country), labels = round(table(xyzdata$delta_good_country)
/41540, 2), main = "Delta Good Country Proportion Pie Chart", col = rainbow(3))
legend("topright", c("-1: become more limited", "0: unchanged", "1: become less limite
d"), cex = 0.8, fill = rainbow(3))
```



Results from EDA:

Variables seems to have pattern, and please note that this is a little bit subjective judgement:

(>>>>> means we consider the variable acts (or ranges) more differently in adopter_1 and adopter_0 compared to others, i.e., more possible patterns)

delta_shouts >>>>>

delta_playlist delta_posts >>>

delta_lovedTracks >>>>>

delta_songListened >>>>>

delta_subscriber_friend_cnt

delta_avg_friend_age >>>>>

delta_avg_friend_male >>>

delta_friend_cnt >>>>>

posts lovedTracks >>>>>

shouts >>>>>

playlists songListened >>>>>

subscriber_friend_cnt

friend_country_cnt

avg_friend_age friend_cnt

And we have more limited countries without changing status and more males than females.

Another thing we want to mention is that **we guess the performance of model fitting might not be so ideal since we can't observe clear pattern in EDA**, so we might focus on finding a model that can get the most senses in business perspective rather than "torchering" data to get as much as information we want. We will definitely do adjustment to find better one, we are saying that although numerically thinking the performance might not be perfect, yet how we can apply that in business strategy is more important.

Normalization

Implement min-max normalization before predicting.

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

# And transfer the response into factors for the two datasets.
xyzdata$adopter <- as.factor(xyzdata$adopter)

# use the mutate_at() to specify the indexes of columns needed normalization
# we can and need to normalize both binary and numerical data, except for adopter and
user_id
xyzdata_normalized <- xyzdata %>% mutate_at(c(2:26), normalize)

# we drop the user_id since it' just an index
xyzdata_normalized_drop_user_id <- xyzdata_normalized[, -1]

# use createDataPartition() fo split the training and testing dataset for w. delta dat
a
train_rows <- createDataPartition(y = xyzdata_normalized_drop_user_id$adopter, p = 0.7
5, list = FALSE)
xyzdata_normalized_drop_user_id_train <- xyzdata_normalized_drop_user_id[train_rows, ]
xyzdata_normalized_drop_user_id_test <- xyzdata_normalized_drop_user_id[-train_rows, ]
```

Before Feature Selection: with / without oversampled training set + no feature selection

Logistic regression: no oversampled training set, no feature selection

```
fit_log <- glm(adopter ~ ., data = xyzdata_normalized_drop_user_id_train, family = bin
omial)
summary(fit_log)
## Call:
## glm(formula = adopter ~ ., family = binomial, data = xyzdata_normalized_drop_user_i
d_train)
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -9.565845   2.045353  -4.677 2.91e-06 ***
## age                           1.547539   0.398181   3.887 0.000102 ***
## male                          0.425819   0.070714   6.022 1.73e-09 ***
## friend_cnt                  -28.773317   7.533190  -3.820 0.000134 ***
## avg_friend_age                1.610021   0.507382   3.173 0.001508 **
## avg_friend_male              -0.018396   0.104836  -0.175 0.860706
## friend_country_cnt            5.243808   0.848074   6.183 6.28e-10 ***
## subscriber_friend_cnt         9.006915   3.194428   2.820 0.004809 **
## songsListened                 4.228177   0.908598   4.654 3.26e-06 ***
## lovedTracks                   5.458826   0.820954   6.649 2.94e-11 ***
## posts                         1.234433   1.908698   0.647 0.517800
```

```
## playlists                    7.512352   2.077220   3.617 0.000299 ***
## shouts                      -2.127623   2.671817  -0.796 0.425846
## delta_friend_cnt            -2.354521   3.857038  -0.610 0.541565
## delta_avg_friend_age         0.722323   2.029636   0.356 0.721924
## delta_avg_friend_male       -2.796477   1.028651  -2.719 0.006556 **
## delta_friend_country_cnt     4.302475   2.602352   1.653 0.098269 .
## delta_subscriber_friend_cnt -3.103975   1.596256  -1.945 0.051831 .
## delta_songsListened         10.916540   3.376688   3.233 0.001225 **
## delta_lovedTracks            1.522996   1.848926   0.824 0.410099
## delta_posts                  0.840844   1.894402   0.444 0.657146
## delta_playlists             -0.054931   1.738319  -0.032 0.974791
## delta_shouts                10.615251   3.646044   2.911 0.003598 **
## tenure                       0.001077   0.188151   0.006 0.995433
## good_country                -0.472410   0.069739  -6.774 1.25e-11 ***
## delta_good_country           0.133220   1.617930   0.082 0.934376
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9877.8  on 31154  degrees of freedom
## Residual deviance: 9261.3  on 31129  degrees of freedom
## AIC: 9313.3
##
## Number of Fisher Scoring iterations: 6
```

Prediction

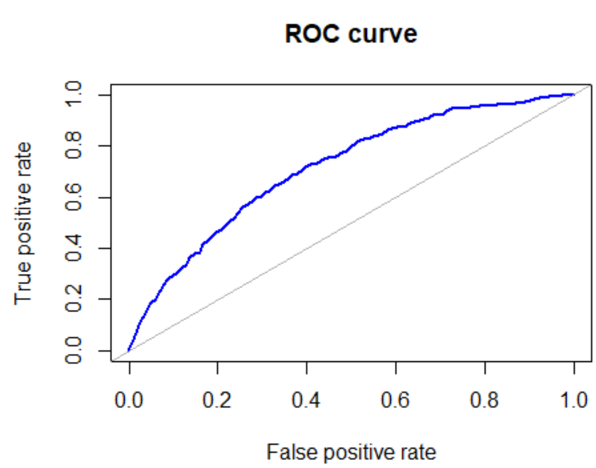
```r
pred_fit_log <- predict(fit_log, newdata = xyzdata_normalized_drop_user_id_test, type
= "response")
```

Check the accuracy by measuring AUC

```r
roc_log <- roc.curve(xyzdata_normalized_drop_user_id_test$adopter, pred_fit_log, col =
 "blue", lwd = 2)
```

```r
roc_log$auc
```

```
## [1] 0.7156439
```



Generate a dataframe of cutoff and corresponding recall_p and precision.

```
# initialize the dataframe
dashboard <- data.frame()

# initialize vectors
cutoff <- c()
precision <- c()
recall_p <- c()

# for loop to get corresponding recall_p and precision for each cutoff value
threshold <- roc_log$thresholds
for (i in 1:(length(threshold))){
  cutoff <- c(cutoff, threshold[i])
  binary_predictions <- ifelse(pred_fit_log >= threshold[i], 1, 0)
  confusion_matrix <- confusionMatrix(data = factor(binary_predictions), reference = x
yzdata_normalized_drop_user_id_test$adopter, mode = "prec_recall", positive = "1")
  recall_p <- c(recall_p, roc_log$true.positive.rate[i])
  precision <- c(precision, confusion_matrix$byClass[["Precision"]])
}

## Warning in confusionMatrix.default(data = factor(binary_predictions), reference
## = xyzdata_normalized_drop_user_id_test$adopter, : Levels are not in the same
## order for reference and data. Refactoring data to match.

## Warning in confusionMatrix.default(data = factor(binary_predictions), reference
## = xyzdata_normalized_drop_user_id_test$adopter, : Levels are not in the same
## order for reference and data. Refactoring data to match.

dashboard <- data.frame(cutoff, recall_p, precision)
dashboard

##          cutoff   recall_p  precision
## 1          -Inf 1.00000000 0.03707270
## 2   0.005866574 1.00000000 0.03708341
## 3   0.012193376 1.00000000 0.03768598
## 4   0.012981379 1.00000000 0.03805476
## 5   0.013603754 1.00000000 0.03842699
## 6   0.014299764 1.00000000 0.03883397
## 7   0.015102938 1.00000000 0.03921369
## 8   0.015794379 1.00000000 0.03964576
## 9   0.016386933 1.00000000 0.04009581
## 10  0.016924559 0.99740260 0.04043382
## 11  0.017366165 0.98441558 0.04041373
## 12  0.017755115 0.98441558 0.04085372
## 13  0.018114629 0.98181818 0.04115854
## 14  0.018464446 0.98181818 0.04163913
## 15  0.018810457 0.97922078 0.04205712
## 16  0.019121862 0.97922078 0.04253639
## 17  0.019443725 0.97922078 0.04302180
## 18  0.019772422 0.97922078 0.04353851
```

```
## 19  0.020058614 0.97662338 0.04401264
## 20  0.020364516 0.97662338 0.04453921
## 21  0.020660517 0.97662338 0.04505692
## 22  0.020943687 0.97402597 0.04553734
## 23  0.021230234 0.96883117 0.04585690
## 24  0.021506600 0.96363636 0.04622477
## 25  0.021809578 0.96103896 0.04675850
## 26  0.022112005 0.95844156 0.04721085
## 27  0.022395547 0.95584416 0.04769929
## 28  0.022681088 0.95584416 0.04838286
## 29  0.022973564 0.95324675 0.04893986
## 30  0.023254192 0.94545455 0.04922245
## 31  0.023554033 0.94285714 0.04971922
## 32  0.023893461 0.93246753 0.05002787
## 33  0.024212561 0.92987013 0.05061501
## 34  0.024535666 0.92467532 0.05103211
## 35  0.024913203 0.91688312 0.05136787
## 36  0.025274383 0.91168831 0.05180047
## 37  0.025602707 0.89350649 0.05167493
## 38  0.025961601 0.88831169 0.05218984
## 39  0.026313671 0.88571429 0.05285183
## 40  0.026637246 0.88311688 0.05357706
## 41  0.026985593 0.87792208 0.05420141
## 42  0.027321770 0.87792208 0.05512967
## 43  0.027638669 0.87532468 0.05585944
## 44  0.027970726 0.87532468 0.05687764
## 45  0.028311466 0.86233766 0.05693706
## 46  0.028666769 0.85194805 0.05737275
## 47  0.029007020 0.83376623 0.05708696
## 48  0.029341875 0.82077922 0.05737110
## 49  0.029677618 0.80779221 0.05756061
## 50  0.030007770 0.80779221 0.05869032
## 51  0.030363612 0.80259740 0.05951464
## 52  0.030718512 0.80000000 0.06045142
## 53  0.031019769 0.78701299 0.06079454
## 54  0.031300837 0.78441558 0.06196143
## 55  0.031638117 0.77662338 0.06259158
## 56  0.032002571 0.76623377 0.06325043
## 57  0.032356925 0.75324675 0.06359649
## 58  0.032719916 0.74545455 0.06445093
## 59  0.033095073 0.73766234 0.06539259
## 60  0.033497635 0.73766234 0.06690224
## 61  0.033886162 0.72987013 0.06774349
## 62  0.034221489 0.71948052 0.06864932
## 63  0.034613508 0.70909091 0.06943032
## 64  0.035075575 0.70129870 0.07034914
## 65  0.035489493 0.68831169 0.07117916
## 66  0.035873621 0.68051948 0.07257618
## 67  0.036305913 0.66753247 0.07315685
## 68  0.036794261 0.66493506 0.07518355
## 69  0.037286689 0.65974026 0.07680677
```

```
## 70  0.037785998 0.64155844 0.07697102
## 71  0.038254006 0.61818182 0.07672469
## 72  0.038714505 0.60259740 0.07754011
## 73  0.039232801 0.59220779 0.07897471
## 74  0.039836986 0.57142857 0.07907980
## 75  0.040474155 0.56103896 0.08089888
## 76  0.041071880 0.54545455 0.08158508
## 77  0.041748843 0.52727273 0.08248679
## 78  0.042477095 0.50909091 0.08308605
## 79  0.043186775 0.50129870 0.08573967
## 80  0.043898355 0.48831169 0.08732002
## 81  0.044660408 0.47532468 0.08913785
## 82  0.045478098 0.46493506 0.09222050
## 83  0.046412460 0.45194805 0.09482289
## 84  0.047557027 0.43116883 0.09612044
## 85  0.048770051 0.42597403 0.10086101
## 86  0.050069857 0.40519481 0.10290237
## 87  0.051535441 0.39740260 0.10812721
## 88  0.053060642 0.37402597 0.10967251
## 89  0.054662065 0.36363636 0.11599006
## 90  0.056262911 0.35064935 0.12184116
## 91  0.058097232 0.32987013 0.12738215
## 92  0.060584984 0.31168831 0.13407821
## 93  0.063524847 0.28571429 0.13924051
## 94  0.066793349 0.25454545 0.14475628
## 95  0.070940605 0.22597403 0.14948454
## 96  0.076919141 0.16623377 0.13763441
## 97  0.085424948 0.13766234 0.14804469
## 98  0.099824231 0.08831169 0.13492063
## 99  0.128613995 0.05454545 0.13815789
## 100 0.573966926 0.00000000 0.00000000
## 101         Inf 0.00000000         NA
```

Cross validation for logistics regression, oversampled training set, no feature selection

We use 10 folds cross-validation. Note that if we want to combind cross-validation and oversampling, we should oversample the 9 folds as training each time INSIDE the loop.

```
library(caret)

# create a list of row indexes that correspond to each folds
cv <- createFolds(y = xyzdata_normalized_drop_user_id$adopter, k = 10)

# a vector to store auc from each fold
AUC_cv <- c()

for(test_rows in cv){
  xyz_train <- xyzdata_normalized_drop_user_id[-test_rows, ]
  xyz_test <- xyzdata_normalized_drop_user_id[test_rows, ]

  # oversample the training set
  library(ROSE)
  xyz_train_oversample_cv <- ROSE(adopter ~., data = xyz_train, seed = 123)$data

  # train the model then evaluate its performance
  fit_log_cv <- glm(adopter ~ ., data = xyz_train_oversample_cv, family = binomial)

  # predict
  pred_fit_log_cv <- predict(fit_log_cv, newdata = xyz_test, type = "response")

  # get auc
  roc_cv <- roc(xyz_test$adopter, pred_fit_log_cv, col = "blue", lwd = 2)
  auc_cv <- roc_cv$auc

  # add auc of current folds
  AUC_cv <- c(AUC_cv, auc_cv)
}

# report average accuracy across folds
mean(AUC_cv)

## [1] 0.7488959
```

Oversampled training set helps in performance

Check multicollinearity

```r
library(car)
vif(fit_log_cv)
##                        age                       male
##                   1.308288                   1.022346
##                 friend_cnt             avg_friend_age
##                   1.272211                   1.313443
##            avg_friend_male         friend_country_cnt
##                   1.017983                   1.314433
##      subscriber_friend_cnt               songsListened
##                   1.137700                   1.178350
##                lovedTracks                      posts
##                   1.056810                   1.033002
##                  playlists                     shouts
##                   1.018901                   1.097887
##           delta_friend_cnt       delta_avg_friend_age
##                   1.128950                   1.034451
##      delta_avg_friend_male   delta_friend_country_cnt
##                   1.034472                   1.107142
## delta_subscriber_friend_cnt       delta_songsListened
##                   1.021985                   1.121256
##          delta_lovedTracks                delta_posts
##                   1.046237                   1.008677
##            delta_playlists               delta_shouts
##                   1.013001                   1.020589
##                     tenure               good_country
##                   1.115771                   1.020204
##         delta_good_country
##                   1.006480
```

Variance Inflation Factors: VIF = 1/(1 - R_squared^2), detects multicollinearity in regression analysis. Multicollinearity happens when independent variables in a regression model are highly correlated to each other, making it hard to interpret the model and also causes problems in performance.

Reading VIF:

VIF of 1.9 indicates the variance of a particular coefficient is 90% higher than what we would expect if there was no multicollinearity, i.e. the variance of a particular coefficient is 90% higher than being orthogonal.

Usually VIF < 2 is not going to cause problems, which is the case here. But we still want to do PCA later since too many variables makes the model hard to interpret and some variables might measure similar factors.

Feature Selection: Filter Approach

Filtering

We choose top 10 variables after many try and find the 10 can get higher AUC and more stable as well.

In some of our trails the top K (K > 10) will give different results, which we consider unstable and shouldn't be included in the model.

```
library(FSelectorRcpp)
IG <- information_gain(adopter ~ ., data = xyzdata_normalized_drop_user_id)

# select top 10
top10 <- cut_attrs(IG, k = 10)

# the whole normalized dataset
xyzdata_normalized_drop_user_id_top10 <- xyzdata_normalized_drop_user_id %>% select(to
p10, adopter)

# training set
xyzdata_normalized_drop_user_id_train_top10 <- xyzdata_normalized_drop_user_id_train %
>% select(top10, adopter)

# testing set
xyzdata_normalized_drop_user_id_test_top10 <- xyzdata_normalized_drop_user_id_test %>%
 select(top10, adopter)
```

Cross validation for logistics regression, oversampled training set, top 10 features

We use 10 folds cross-validation

```
library(caret)

# create a list of row indexes that correspond to each folds
cv <- createFolds(y = xyzdata_normalized_drop_user_id_top10$adopter, k = 10)
# a vector to store auc from each fold
AUC_cv_filter <- c()

for(test_rows in cv){
  xyz_train_f <- xyzdata_normalized_drop_user_id_top10[-test_rows, ]
  xyz_test_f <- xyzdata_normalized_drop_user_id_top10[test_rows, ]

  # oversample the training folds
  xyz_train_oversample_filter <- ROSE(adopter ~ lovedTracks + delta_songsListened + de
lta_lovedTracks + subscriber_friend_cnt + songsListened + friend_cnt + friend_country_
cnt + delta_friend_cnt + delta_subscriber_friend_cnt + delta_avg_friend_male, data = x
yz_train_f, seed = 123)$data

  # train the model then evaluate its performance
```

```
  fit_log_oversample_filter_cv <- glm(adopter ~ lovedTracks + delta_songsListened + de
lta_lovedTracks + subscriber_friend_cnt + songsListened + friend_cnt + friend_country_
cnt + delta_friend_cnt + delta_subscriber_friend_cnt + delta_avg_friend_male, data = x
yz_train_oversample_filter, family = binomial)

  # predict
  pred_fit_log_ftiler_cv <- predict(fit_log_oversample_filter_cv, newdata = xyz_test_f,
 type = "response")

  # get auc
  roc_cv_filter <- roc.curve(xyz_test_f$adopter, pred_fit_log_ftiler_cv)
  auc_cv_filter <- roc_cv_filter$auc

  # add auc of current folds
  AUC_cv_filter <- c(AUC_cv_filter, auc_cv_filter)
}

# report average accuracy across folds
mean(AUC_cv_filter)

## [1] 0.7413279
```

Check multicollinearity

```
vif(fit_log_oversample_filter_cv)

##                 lovedTracks         delta_songsListened
##                    1.083114                    1.151555
##            delta_lovedTracks       subscriber_friend_cnt
##                    1.090058                    1.196114
##               songsListened                  friend_cnt
##                    1.169008                    1.407244
##           friend_country_cnt            delta_friend_cnt
##                    1.444149                    1.115594
## delta_subscriber_friend_cnt       delta_avg_friend_male
##                    1.021396                    1.001556
```

So far, we've seen some important points:

1. oversampled training set is needed.

2. no severe multicollinearity problems needed to concern, but we will still try PCA later.

3. some top 10 variables being selected act in relatively clear pattern in EDA previously.

Generate a dataframe of cutoff and corresponding recall_p and precision.

How to utilize the dashboard depends on business strategy:

For example, if the company has more budget, it sets the threshold as a low value, like `0.4021989`, that is, even if there's only a little chance that the user will convert to premium service, the company still want to target that person.

Thus, it will get recall_positive of `0.889610390 with precision 0.05959113`

```
# initialize the dataframe
dashboard_filter_log <- data.frame()

# initialize vectors
cutoff_filter_log <- c()
precision_filter_log <- c()
recall_p_filter_log <- c()

# for loop to get corresponding recall_p and precision for each cutoff value
threshold <- roc_cv_filter$thresholds
for (i in 1:(length(threshold))){
  cutoff_filter_log <- c(cutoff_filter_log, threshold[i])
  binary_predictions <- ifelse(pred_fit_log_ftiler_cv >= threshold[i], 1, 0)
  confusion_matrix <- confusionMatrix(data = factor(binary_predictions), reference = x
yz_test_f$adopter, mode = "prec_recall", positive = "1")
  recall_p_filter_log <- c(recall_p_filter_log, roc_cv_filter$true.positive.rate[i])
  precision_filter_log <- c(precision_filter_log, confusion_matrix$byClass[["Precision
"]])
}

dashboard_filter_log <- data.frame(cutoff_filter_log, recall_p_filter_log, precision_f
ilter_log)
dashboard_filter_log
```

```
##    cutoff_filter_log recall_p_filter_log precision_filter_log
## 1               -Inf         1.000000000           0.03707270
## 2          0.2553300         1.000000000           0.03709949
## 3          0.3665202         0.993506494           0.03726254
## 4          0.3715558         0.993506494           0.03771260
## 5          0.3718179         0.987012987           0.03905447
## 6          0.3719519         0.987012987           0.03942931
## 7          0.3721398         0.987012987           0.03984273
## 8          0.3723844         0.980519481           0.04003181
## 9          0.3726462         0.980519481           0.04042838
## 10         0.3729173         0.974025974           0.04061738
## 11         0.3732752         0.974025974           0.04110715
## 12         0.3736990         0.967532468           0.04127424
## 13         0.3741634         0.967532468           0.04184218
## 14         0.3747084         0.967532468           0.04223356
## 15         0.3753262         0.967532468           0.04282840
```

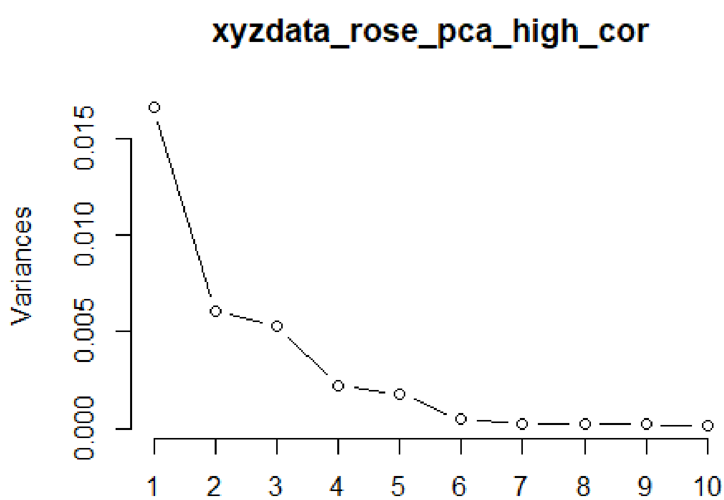
```
## 16          0.3759284     0.967532468      0.04326365
## 17          0.3763906     0.967532468      0.04375918
## 18          0.3768006     0.967532468      0.04442457
## 19          0.3772591     0.967532468      0.04489304
## 20          0.3777707     0.967532468      0.04555182
## 21          0.3782850     0.967532468      0.04608723
## 22          0.3788205     0.961038961      0.04632238
## 23          0.3794952     0.961038961      0.04689480
## 24          0.3801802     0.961038961      0.04752730
## 25          0.3808212     0.961038961      0.04806755
## 26          0.3814993     0.961038961      0.04878049
## 27          0.3822065     0.961038961      0.04949833
## 28          0.3829804     0.961038961      0.05008460
## 29          0.3838857     0.961038961      0.05082418
## 30          0.3849129     0.954545455      0.05125523
## 31          0.3859416     0.954545455      0.05199859
## 32          0.3869075     0.954545455      0.05266929
## 33          0.3877884     0.948051948      0.05309091
## 34          0.3887440     0.928571429      0.05290418
## 35          0.3900031     0.928571429      0.05347794
## 36          0.3914230     0.915584416      0.05369383
## 37          0.3927386     0.915584416      0.05456656
## 38          0.3939525     0.915584416      0.05544632
## 39          0.3951780     0.915584416      0.05637745
## 40          0.3965720     0.902597403      0.05648111
## 41          0.3978985     0.902597403      0.05748553
## 42          0.3991680     0.896103896      0.05788591
## 43          0.4006415     0.889610390      0.05854701
## 44          0.4021989     0.889610390      0.05959113
## 45          0.4036303     0.863636364      0.05871965
## 46          0.4050166     0.831168831      0.05765766
## 47          0.4066215     0.811688312      0.05741847
## 48          0.4083132     0.811688312      0.05835668
## 49          0.4100430     0.798701299      0.05862726
## 50          0.4114741     0.792207792      0.05933852
## 51          0.4128420     0.785714286      0.06004963
## 52          0.4145845     0.785714286      0.06139016
## 53          0.4164126     0.772727273      0.06153051
## 54          0.4183327     0.753246753      0.06114918
## 55          0.4203142     0.720779221      0.05996759
## 56          0.4222516     0.707792208      0.06042129
## 57          0.4240354     0.707792208      0.06137387
## 58          0.4260878     0.707792208      0.06289671
## 59          0.4284154     0.681818182      0.06216696
## 60          0.4308158     0.649350649      0.06060606
## 61          0.4331357     0.642857143      0.06168224
## 62          0.4351628     0.636363636      0.06242038
## 63          0.4374856     0.629870130      0.06369009
## 64          0.4400788     0.623376623      0.06451613
## 65          0.4430116     0.610389610      0.06523248
## 66          0.4459552     0.603896104      0.06600426
```

```
## 67           0.4487214          0.590909091           0.06656913
## 68           0.4519922          0.577922078           0.06747536
## 69           0.4549241          0.571428571           0.06885759
## 70           0.4576215          0.564935065           0.07016129
## 71           0.4608823          0.558441558           0.07136929
## 72           0.4647877          0.538961039           0.07155172
## 73           0.4690014          0.538961039           0.07397504
## 74           0.4731872          0.525974026           0.07555970
## 75           0.4772120          0.519480519           0.07699711
## 76           0.4809774          0.512987013           0.07900000
## 77           0.4855836          0.512987013           0.08246347
## 78           0.4912550          0.500000000           0.08415301
## 79           0.4963844          0.493506494           0.08685714
## 80           0.5012927          0.474025974           0.08711217
## 81           0.5065899          0.461038961           0.08919598
## 82           0.5110277          0.454545455           0.09333333
## 83           0.5153706          0.441558442           0.09577465
## 84           0.5205059          0.441558442           0.10104012
## 85           0.5275486          0.409090909           0.09952607
## 86           0.5361984          0.383116883           0.10000000
## 87           0.5469354          0.350649351           0.09747292
## 88           0.5575843          0.331168831           0.10079051
## 89           0.5675452          0.318181818           0.10425532
## 90           0.5788846          0.285714286           0.10256410
## 91           0.5927726          0.272727273           0.10937500
## 92           0.6072127          0.246753247           0.10982659
## 93           0.6221135          0.227272727           0.11627907
## 94           0.6392739          0.188311688           0.11196911
## 95           0.6572604          0.162337662           0.11363636
## 96           0.6799630          0.129870130           0.10989011
## 97           0.7158846          0.110389610           0.12056738
## 98           0.7597021          0.077922078           0.11428571
## 99           0.8214377          0.032467532           0.08771930
## 100          0.9310767          0.006493506           0.04761905
## 101                Inf          0.000000000                   NA
```

PCA:

oversampled training set, logistic regression, integrate relatively highly correlated variables

```
# we use oversampled training set
library(ROSE)
xyz_train_oversample_pca <- ROSE(adopter ~., data = xyzdata_normalized_drop_user_id_tr
ain, seed = 123)$data

# get PCs
high_cor <- xyz_train_oversample_pca[, c(1, 3, 4, 6, 7, 8, 9, 12, 13, 16, 18, 19, 22)]
xyzdata_rose_eigen_high_cor <- eigen(cor(high_cor))
xyzdata_rose_pca_high_cor <- prcomp(high_cor)
summary(xyzdata_rose_pca_high_cor)

## Importance of components:
##                          PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     0.129 0.07781 0.07265 0.04670 0.04227 0.02096 0.01498
## Proportion of Variance 0.498 0.18119 0.15793 0.06526 0.05348 0.01314 0.00671
## Cumulative Proportion  0.498 0.67920 0.83714 0.90240 0.95588 0.96902 0.97573
##                            PC8     PC9    PC10    PC11    PC12     PC13
## Standard deviation     0.01429 0.01336 0.01175 0.01122 0.01013 0.007853
## Proportion of Variance 0.00611 0.00534 0.00413 0.00377 0.00307 0.001850
## Cumulative Proportion  0.98184 0.98718 0.99132 0.99509 0.99815 1.000000

# colnames(xyz_train_oversample_pca[, c(1, 3, 4, 6, 7, 8, 9, 12, 13, 16, 18, 19, 22)])

screeplot(xyzdata_rose_pca_high_cor, type = "lines")
```



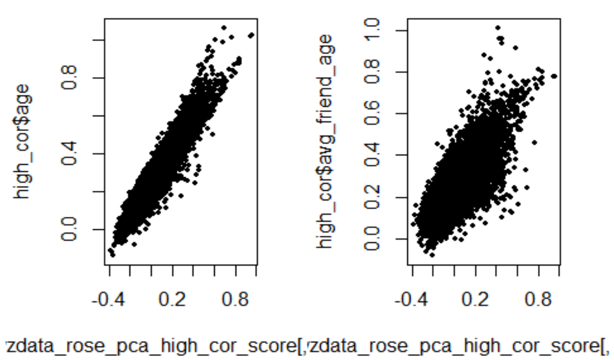
From the screeplot and summary of PCs, it seems the first 6 PCs are more important since the cumulative proportion of variance explained by them is about 96% for those variables with higher correlations.

We also check relationships of PCs and those variables with higher correlations.

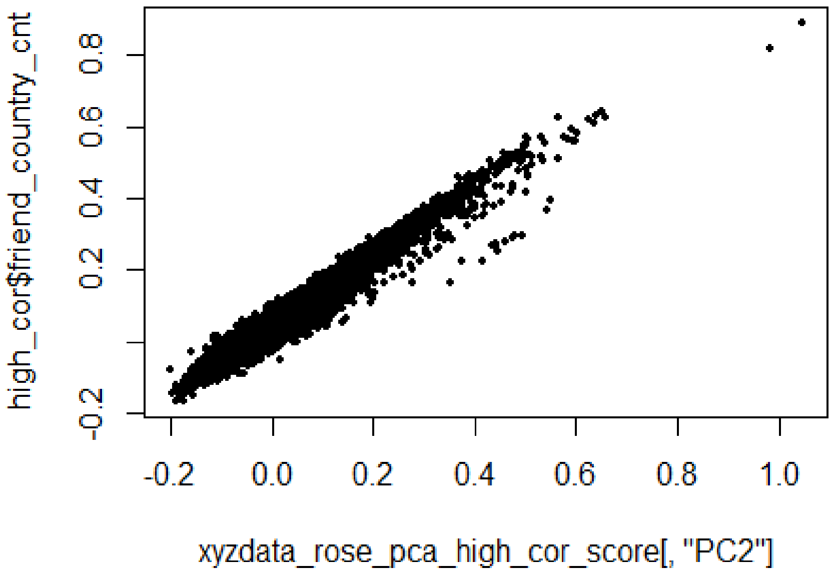
(the code output is replaced with the labeled picture PDF file after rendering to help reading)

```
> xyzdata_rose_pca_high_cor$rotation
                                  PC1         PC2           PC3          PC4           PC5
age                       0.842770646 -0.04409311 -5.361060e-01 -0.015671438 -0.0011486449
friend_cnt               -0.004915786 -0.08826167  4.390783e-03 -0.004819973 -0.0001716309
avg_friend_age            0.537284569  0.06017231  8.404130e-01 -0.013737417  0.0261958111
friend_country_cnt       -0.008133575 -0.92829939  7.197600e-02 -0.306094543 -0.0757187589
subscriber_friend_cnt     0.010221443 -0.09635543  8.530220e-03  0.002278925 -0.0106739658
songsListened            -0.003757317 -0.20689276 -3.384470e-03  0.412110536  0.8784672701
lovedTracks               0.023591416 -0.22765156  3.129953e-02  0.851439655 -0.4526264699
shouts                   -0.007870433 -0.07088095  3.044130e-03  0.010169611  0.0230035941
delta_friend_cnt         -0.008085100 -0.07608059 -7.886802e-04 -0.006460875 -0.0374492620
delta_friend_country_cnt -0.007459309 -0.06158391 -2.220850e-03 -0.020542642 -0.0344021275
delta_songsListened      -0.006376046 -0.03332280 -2.985958e-03  0.051268870  0.0900551290
delta_lovedTracks        -0.003251021 -0.05042978 -8.954417e-05  0.088673515 -0.0594477783
delta_shouts             -0.007955057 -0.05227312 -2.998324e-03 -0.000402623 -0.0461586098
                                  PC6           PC7           PC8           PC9         PC10
age                       0.005161538  0.0008245148  0.0007999585  0.0030442992 -0.007971594
friend_cnt                0.046565921 -0.0315254856 -0.3074473172 -0.0816562317  0.127262905
avg_friend_age            0.019963681 -0.0033593164  0.0008464988 -0.0005279535 -0.008792855
friend_country_cnt       -0.126791639  0.0394070688  0.1146348983  0.0188944237 -0.030190494
subscriber_friend_cnt     0.058795290 -0.1063047211 -0.5630892882 -0.3155582069  0.561083565
songsListened             0.027059036 -0.0019226311  0.0372114271  0.0089205418  0.036063990
lovedTracks              -0.069250423  0.0107928665  0.0291653216  0.0896746615  0.046944480
shouts                    0.147909609  0.2166278436 -0.6903115081  0.4086121533 -0.514032440
delta_friend_cnt          0.409107426 -0.2087443181 -0.1512293576 -0.1952540379  0.139064956
delta_friend_country_cnt  0.436978788 -0.7446595717  0.1087805285  0.3742976580 -0.049750095
delta_songsListened       0.080627856 -0.1115326572  0.0803895013 -0.1557310258 -0.193476517
delta_lovedTracks         0.177373648 -0.1035438113 -0.0113023023 -0.7173367649 -0.562724903
delta_shouts              0.744437200  0.5637728625  0.2349575787  0.0354493518  0.155873990
```

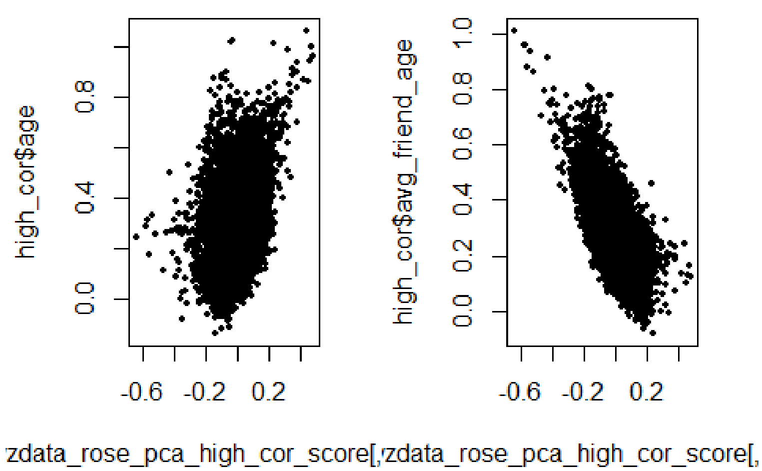
```
xyzdata_rose_pca_high_cor_score <- xyzdata_rose_pca_high_cor$x

# plot the relationship
par(mfrow = c(1, 2))
plot(xyzdata_rose_pca_high_cor_score[, "PC1"], high_cor$age, pch = 20, cex = 0.8)
plot(xyzdata_rose_pca_high_cor_score[, "PC1"], high_cor$avg_friend_age, pch = 20, cex
= 0.8)
```



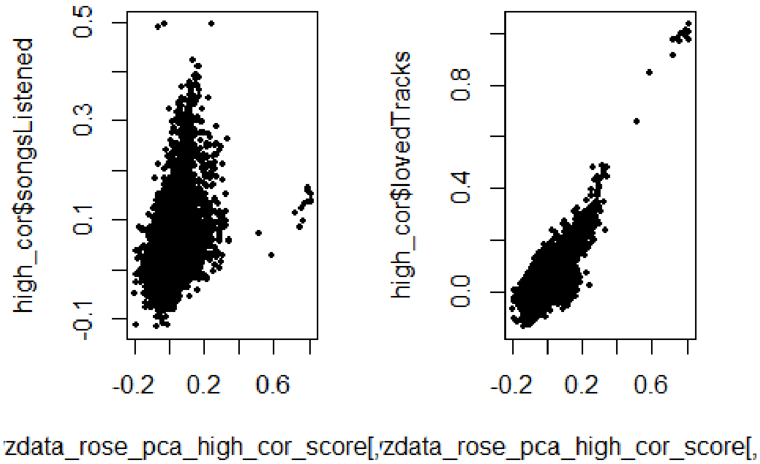
```r
par(mfrow = c(1, 1))
plot(xyzdata_rose_pca_high_cor_score[, "PC2"], high_cor$friend_country_cnt, pch = 20,
cex = 0.8)
```



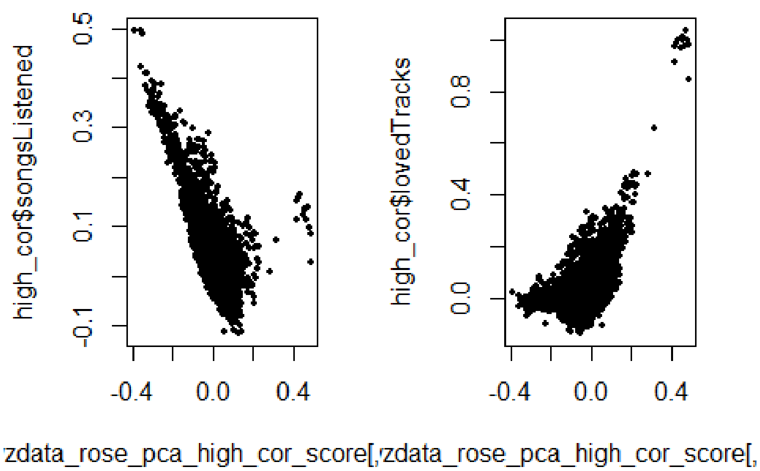

```r
par(mfrow = c(1, 2))
plot(xyzdata_rose_pca_high_cor_score[, "PC3"], high_cor$age, pch = 20, cex = 0.8)
plot(xyzdata_rose_pca_high_cor_score[, "PC3"], high_cor$avg_friend_age, pch = 20, cex
= 0.8)
```



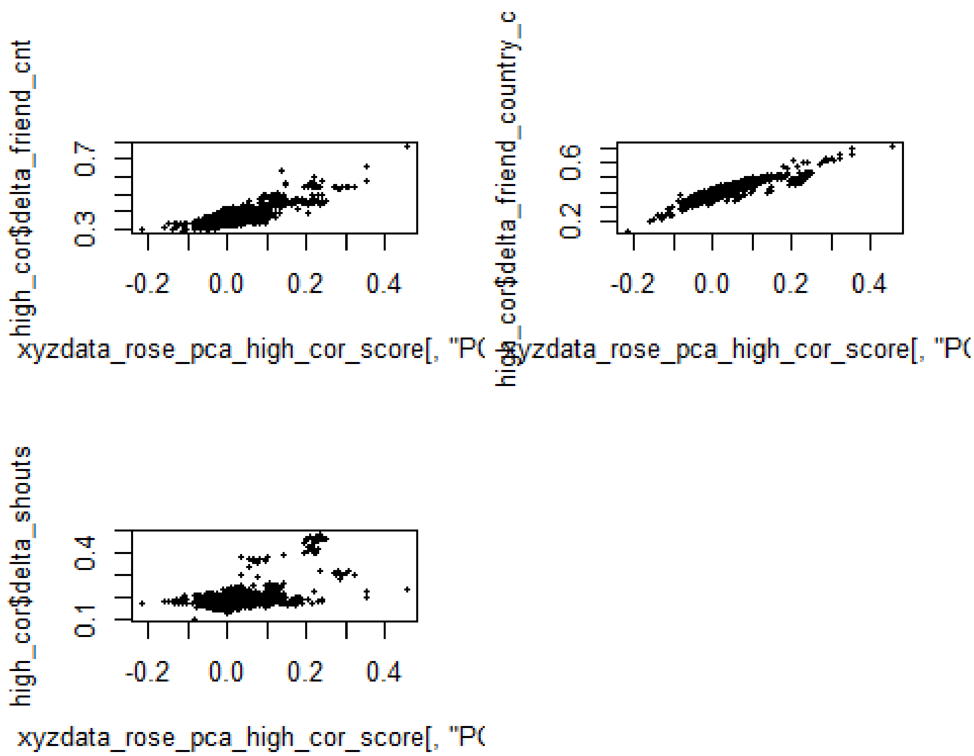
```
par(mfrow = c(1, 2))
plot(xyzdata_rose_pca_high_cor_score[, "PC4"], high_cor$songsListened, pch = 20, cex =
 0.8)
plot(xyzdata_rose_pca_high_cor_score[, "PC4"], high_cor$lovedTracks, pch = 20, cex = 0.
8)
```




```
par(mfrow = c(1, 2))
plot(xyzdata_rose_pca_high_cor_score[, "PC5"], high_cor$songsListened, pch = 20, cex =
 0.8)
plot(xyzdata_rose_pca_high_cor_score[, "PC5"], high_cor$lovedTracks, pch = 20, cex = 0.
8)
```



```
par(mfrow = c(2, 2))
plot(xyzdata_rose_pca_high_cor_score[, "PC6"], high_cor$delta_friend_cnt, pch = 20, ce
x = 0.8)
plot(xyzdata_rose_pca_high_cor_score[, "PC6"], high_cor$delta_friend_country_cnt, pch
= 20, cex = 0.8)
plot(xyzdata_rose_pca_high_cor_score[, "PC6"], high_cor$delta_shouts, pch = 20, cex =
0.8)
```




```
# identify the PCs, give up PC3
age_related <- xyzdata_rose_pca_high_cor_score[, "PC1"]
friend_diversity <- -1*xyzdata_rose_pca_high_cor_score[, "PC2"]
tend_to_exploration <- xyzdata_rose_pca_high_cor_score[, "PC4"]
tend_to_concentration <- xyzdata_rose_pca_high_cor_score[, "PC5"]
information_received <- xyzdata_rose_pca_high_cor_score[, "PC6"]

# fit the logistic regression
PC1 <- scale(age_related)
PC2 <- scale(friend_diversity)
PC4 <- scale(tend_to_exploration)
PC5 <- scale(tend_to_concentration)
PC6 <- scale(information_received)
```

Also consider the plots above:

So far we decide to extract the first 6 PCs except for the 3rd since the variable it measures is already in PC1 and its magnitude is too small and important information may already be included in PC1.

PC1: capture age_related information, maybe usage and age, including age, avg_friend_age.

PC2: capture friend_diversity information, including friend_country_cnt.

PC4: capture tend_to_exploration information, meaning that users might have many already loved content and also tend to try new things, including songsListened, lovedTracks

PC5: capture tend_to_concentration information, meaning that users might have many already loved content but not tend to try new things, including songsListened, lovedTracks

PC6: capture information_received ability since delta_shots is the changed number of wall posts received, and increase in number of friends and friend's countries will also have effects on information_received ability, including delta_shouts, delta_friend_cnt, delta_friend_country_cnt.

Thus, we identify the 5 PCs:

PC1(age_related), PC2(friend_diversity), PC4(tend_to_exploration), PC5(tend_to_concentration), PC6(information_received).

Prepare PCA training and testing set for PCA model.

```
# remained
remained_train <- xyz_train_oversample_pca[, c(-1, -3, -4, -6, -7, -8, -9, -12, -13, -
16, -18, -19, -22)]
remained_test <- xyzdata_normalized_drop_user_id_test[, c(-1, -3, -4, -6, -7, -8, -9,
-12, -13, -16, -18, -19, -22)]

# prepare PCA training set
train_data_pca <- predict(xyzdata_rose_pca_high_cor, xyz_train_oversample_pca[, -which
(names(xyzdata_normalized_drop_user_id_test) == "adopter")])
train_data_pca_6pc <- train_data_pca[, c(1:2, 4:6)] #give up pc3
train_data_pca_6pcAndremained <- cbind(remained_train, train_data_pca_6pc)

# prepare PCA testing set
test_data_pca <- predict(xyzdata_rose_pca_high_cor, xyzdata_normalized_drop_user_id_te
st[, -which(names(xyzdata_normalized_drop_user_id_test) == "adopter")])
test_data_pca_6pc <- test_data_pca[, c(1:2, 4:6)] #give up pc3
test_data_pca_6pcAndremained <- cbind(remained_test, test_data_pca_6pc)

# fit the model
fit_PCA <- glm(adopter ~ PC1 + PC2 + PC4 + PC5 + PC6 + male + avg_friend_male + posts
+ playlists + delta_avg_friend_age + delta_avg_friend_male + delta_subscriber_friend_c
nt + delta_posts + delta_playlists + tenure + good_country + delta_good_country, data
= train_data_pca_6pcAndremained, family = binomial)
```

```r
# check variance inflation factors
library(car)
vif(fit_log_cv) # logistics regression, oversampled training set, no feature selection
```

```
##                        age                       male
##                   1.308288                   1.022346
##                 friend_cnt             avg_friend_age
##                   1.272211                   1.313443
##            avg_friend_male         friend_country_cnt
##                   1.017983                   1.314433
##      subscriber_friend_cnt              songsListened
##                   1.137700                   1.178350
##                lovedTracks                      posts
##                   1.056810                   1.033002
##                  playlists                     shouts
##                   1.018901                   1.097887
##           delta_friend_cnt       delta_avg_friend_age
##                   1.128950                   1.034451
##      delta_avg_friend_male   delta_friend_country_cnt
##                   1.034472                   1.107142
## delta_subscriber_friend_cnt       delta_songsListened
##                   1.021985                   1.121256
##          delta_lovedTracks                delta_posts
##                   1.046237                   1.008677
##            delta_playlists               delta_shouts
##                   1.013001                   1.020589
##                     tenure               good_country
##                   1.115771                   1.020204
##         delta_good_country
##                   1.006480
```

```r
vif(fit_PCA)
```

```
##                        PC1                        PC2
##                   1.069210                   1.076717
##                        PC4                        PC5
##                   1.054404                   1.040487
##                        PC6                       male
##                   1.053990                   1.011793
##            avg_friend_male                      posts
##                   1.006894                   1.024196
##                  playlists       delta_avg_friend_age
##                   1.015588                   1.041135
##      delta_avg_friend_male delta_subscriber_friend_cnt
##                   1.041530                   1.018150
##                delta_posts            delta_playlists
##                   1.017242                   1.008820
##                     tenure               good_country
##                   1.102706                   1.016778
##         delta_good_country
##                   1.003330
```

Check VIFs of the logistics regression, oversampled training set, no feature selection again, they are not in a big problem of multicollinearity, but after fitting PCs in the model, they become lower, which is a good thing.

And if we check again summary of (logistics regression, oversampled training set, no feature selection) and (logistic regression, oversampled training set, PCA for transforming relatively highly correlated variables):

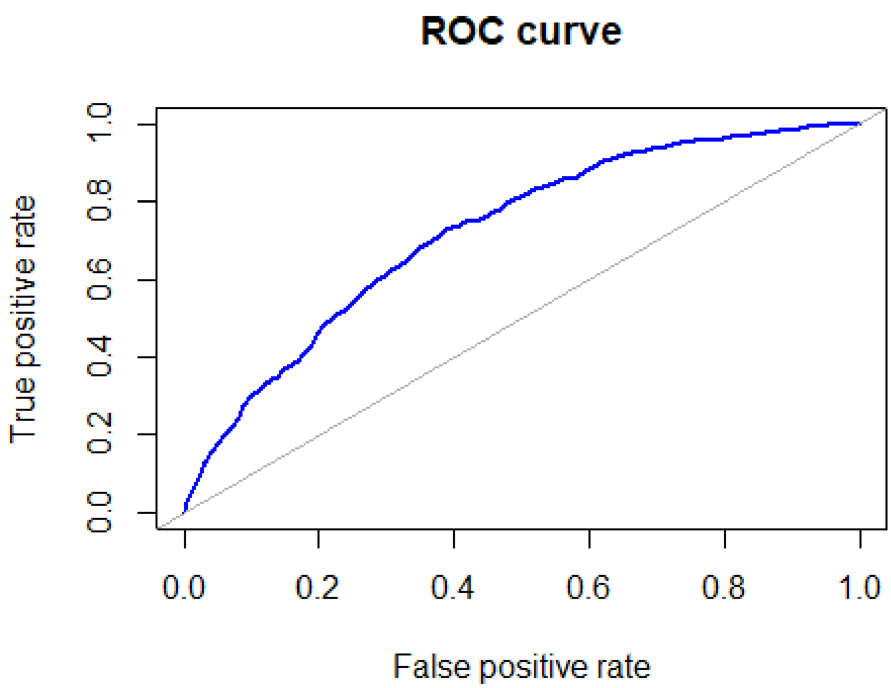
```
summary(fit_log)

##
## Call:
## glm(formula = adopter ~ ., family = binomial, data = xyzdata_normalized_drop_user_i
d_train)
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -9.565845   2.045353  -4.677 2.91e-06 ***
## age                        1.547539   0.398181   3.887 0.000102 ***
## male                       0.425819   0.070714   6.022 1.73e-09 ***
## friend_cnt               -28.773317   7.533190  -3.820 0.000134 ***
## avg_friend_age             1.610021   0.507382   3.173 0.001508 **
## avg_friend_male           -0.018396   0.104836  -0.175 0.860706
## friend_country_cnt         5.243808   0.848074   6.183 6.28e-10 ***
## subscriber_friend_cnt      9.006915   3.194428   2.820 0.004809 **
## songsListened              4.228177   0.908598   4.654 3.26e-06 ***
## lovedTracks                5.458826   0.820954   6.649 2.94e-11 ***
## posts                      1.234433   1.908698   0.647 0.517800
## playlists                  7.512352   2.077220   3.617 0.000299 ***
## shouts                    -2.127623   2.671817  -0.796 0.425846
## delta_friend_cnt          -2.354521   3.857038  -0.610 0.541565
## delta_avg_friend_age       0.722323   2.029636   0.356 0.721924
## delta_avg_friend_male     -2.796477   1.028651  -2.719 0.006556 **
## delta_friend_country_cnt   4.302475   2.602352   1.653 0.098269 .
## delta_subscriber_friend_cnt -3.103975  1.596256  -1.945 0.051831 .
## delta_songsListened       10.916540   3.376688   3.233 0.001225 **
## delta_lovedTracks          1.522996   1.848926   0.824 0.410099
## delta_posts                0.840844   1.894402   0.444 0.657146
## delta_playlists           -0.054931   1.738319  -0.032 0.974791
## delta_shouts              10.615251   3.646044   2.911 0.003598 **
## tenure                     0.001077   0.188151   0.006 0.995433
## good_country              -0.472410   0.069739  -6.774 1.25e-11 ***
## delta_good_country         0.133220   1.617930   0.082 0.934376
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9877.8  on 31154  degrees of freedom
## Residual deviance: 9261.3  on 31129  degrees of freedom
```

```
## AIC: 9313.3
## Number of Fisher Scoring iterations: 6

summary(fit_PCA)

##
## Call:
## glm(formula = adopter ~ PC1 + PC2 + PC4 + PC5 + PC6 + male +
##     avg_friend_male + posts + playlists + delta_avg_friend_age +
##     delta_avg_friend_male + delta_subscriber_friend_cnt + delta_posts +
##     delta_playlists + tenure + good_country + delta_good_country,
##     family = binomial, data = train_data_pca_6pcAndremained)
##
## Coefficients:
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  0.38222    0.48270   0.792  0.42846
## PC1                          2.26720    0.09914  22.868  < 2e-16 ***
## PC2                          6.62770    0.18751  35.346  < 2e-16 ***
## PC4                          7.93645    0.31567  25.142  < 2e-16 ***
## PC5                         -1.47294    0.32384  -4.548 5.41e-06 ***
## PC6                          9.87168    0.72262  13.661  < 2e-16 ***
## male                         0.32967    0.02152  15.316  < 2e-16 ***
## avg_friend_male             -0.02113    0.03455  -0.612  0.54081
## posts                        3.29206    1.24509   2.644  0.00819 **
## playlists                    6.11167    0.63021   9.698  < 2e-16 ***
## delta_avg_friend_age         1.70593    0.67667   2.521  0.01170 *
## delta_avg_friend_male       -2.07805    0.31877  -6.519 7.08e-11 ***
## delta_subscriber_friend_cnt -0.68305    0.49785  -1.372  0.17006
## delta_posts                  2.08114    1.00519   2.070  0.03842 *
## delta_playlists              1.06955    0.70461   1.518  0.12903
## tenure                      -0.11054    0.05586  -1.979  0.04784 *
## good_country                -0.32015    0.02145 -14.926  < 2e-16 ***
## delta_good_country          -0.61894    0.40319  -1.535  0.12476
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43186  on 31154  degrees of freedom
## Residual deviance: 39622  on 31137  degrees of freedom
## AIC: 39658
##
## Number of Fisher Scoring iterations: 4
```

There become less insignificant variables in the latter and we also reduce dimension, making the model more easily to explain while still have nice AUC.

Prediction

```r
# predict
pred_fit_PCA <- predict(fit_PCA, newdata = test_data_pca_6pcAndremained, type = "respo
nse")

# auc
roc_pca <- roc.curve(test_data_pca_6pcAndremained$adopter, pred_fit_PCA, , col = "blue
", lwd = 2)
```




```r
auc_pca <- roc_pca$auc
roc_pca

## Area under the curve (AUC): 0.722
```

If AUC not changing a lot, a more easily explainable model may be preferred.

Generate a dataframe of cutoff and corresponding recall_p and precision.

```r
# initialize the dataframe
dashboard_pca <- data.frame()

# initialize vectors
cutoff_pca <- c()
precision_pca <- c()
recall_p_pca <- c()
```

```r
# for loop to get corresponding recall_p and precision for each cutoff value
threshold <- roc_pca$thresholds
for (i in 1:(length(threshold))){
  cutoff_pca <- c(cutoff_pca, threshold[i])
  binary_predictions <- ifelse(pred_fit_PCA >= threshold[i], 1, 0)
  confusion_matrix <- confusionMatrix(data = factor(binary_predictions), reference = t
est_data_pca_6pcAndremained$adopter, mode = "prec_recall", positive = "1")
  recall_p_pca <- c(recall_p_pca, roc_pca$true.positive.rate[i])
  precision_pca <- c(precision_pca, confusion_matrix$byClass[["Precision"]])
}

dashboard_pca <- data.frame(cutoff_pca, recall_p_pca, precision_pca)
dashboard_pca
```

```
##     cutoff_pca recall_p_pca precision_pca
## 1         -Inf   1.00000000    0.03707270
## 2    0.1291656   1.00000000    0.03708699
## 3    0.2524574   1.00000000    0.03764177
## 4    0.2640654   1.00000000    0.03803220
## 5    0.2739709   1.00000000    0.03842699
## 6    0.2815913   0.99740260    0.03874483
## 7    0.2879162   0.99480519    0.03908163
## 8    0.2940541   0.99480519    0.03946013
## 9    0.2990818   0.99220779    0.03977509
## 10   0.3032385   0.98961039    0.04009260
## 11   0.3077979   0.98441558    0.04035349
## 12   0.3121585   0.98441558    0.04080095
## 13   0.3160698   0.98181818    0.04115406
## 14   0.3198483   0.97922078    0.04156560
## 15   0.3234469   0.97662338    0.04189882
## 16   0.3267260   0.97402597    0.04225828
## 17   0.3297660   0.97142857    0.04262594
## 18   0.3327506   0.96883117    0.04310146
## 19   0.3358806   0.96883117    0.04357477
## 20   0.3390186   0.96883117    0.04412635
## 21   0.3421601   0.96103896    0.04439645
## 22   0.3453871   0.95844156    0.04483052
## 23   0.3486296   0.95844156    0.04540421
## 24   0.3516853   0.95844156    0.04598704
## 25   0.3544337   0.95584416    0.04646465
## 26   0.3573566   0.95324675    0.04694896
## 27   0.3603295   0.95064935    0.04744620
## 28   0.3630938   0.94805195    0.04802632
## 29   0.3655756   0.94285714    0.04840000
## 30   0.3681878   0.93766234    0.04879038
## 31   0.3712986   0.93506494    0.04934887
## 32   0.3746139   0.92987013    0.04982603
## 33   0.3776000   0.92467532    0.05033225
## 34   0.3803974   0.92467532    0.05098826
## 35   0.3831334   0.91948052    0.05153589
## 36   0.3856975   0.91688312    0.05214180
```

```
## 37   0.3881618   0.90909091    0.05253678
## 38   0.3907168   0.90389610    0.05307305
## 39   0.3932764   0.89350649    0.05332507
## 40   0.3956191   0.88571429    0.05371771
## 41   0.3980581   0.87532468    0.05395453
## 42   0.4005326   0.86493506    0.05426104
## 43   0.4028535   0.85974026    0.05481948
## 44   0.4051010   0.85714286    0.05568680
## 45   0.4075894   0.84675325    0.05597527
## 46   0.4101424   0.84155844    0.05668300
## 47   0.4124432   0.83636364    0.05733618
## 48   0.4148293   0.83116883    0.05807623
## 49   0.4173422   0.82337662    0.05860603
## 50   0.4201624   0.81298701    0.05900094
## 51   0.4230760   0.80519481    0.05967276
## 52   0.4256850   0.79740260    0.06033805
## 53   0.4280879   0.77922078    0.06031363
## 54   0.4306895   0.77402597    0.06109061
## 55   0.4335664   0.76623377    0.06166388
## 56   0.4364996   0.75584416    0.06240618
## 57   0.4393736   0.75064935    0.06336330
## 58   0.4422958   0.74805195    0.06454505
## 59   0.4450138   0.74025974    0.06544202
## 60   0.4475144   0.73246753    0.06635294
## 61   0.4504799   0.72727273    0.06751869
## 62   0.4538651   0.71428571    0.06800198
## 63   0.4572613   0.69610390    0.06821074
## 64   0.4604416   0.68831169    0.06938989
## 65   0.4634644   0.67532468    0.06989247
## 66   0.4663353   0.66233766    0.07050041
## 67   0.4693587   0.64415584    0.07067541
## 68   0.4731258   0.63636364    0.07170032
## 69   0.4771540   0.62337662    0.07257333
## 70   0.4809448   0.61038961    0.07334582
## 71   0.4848298   0.59740260    0.07440958
## 72   0.4884108   0.58181818    0.07506702
## 73   0.4919955   0.57142857    0.07623008
## 74   0.4957450   0.55064935    0.07636888
## 75   0.4995297   0.53766234    0.07747006
## 76   0.5039872   0.52207792    0.07842372
## 77   0.5093509   0.50909091    0.07957775
## 78   0.5152266   0.49350649    0.08064516
## 79   0.5207320   0.48311688    0.08230088
## 80   0.5262867   0.46233766    0.08267534
## 81   0.5323860   0.42597403    0.08062930
## 82   0.5388247   0.41298701    0.08200103
## 83   0.5452112   0.38961039    0.08169935
## 84   0.5517795   0.37922078    0.08410138
## 85   0.5590077   0.36883117    0.08781694
## 86   0.5665726   0.35064935    0.08899143
## 87   0.5742170   0.34285714    0.09282700
```

```
## 88   0.5823735   0.32987013    0.09665145
## 89   0.5913740   0.31428571    0.10041494
## 90   0.6010751   0.30389610    0.10550045
## 91   0.6117027   0.27532468    0.10610611
## 92   0.6239016   0.24675325    0.10532151
## 93   0.6377710   0.21818182    0.10579345
## 94   0.6532404   0.20000000    0.11224490
## 95   0.6713886   0.17922078    0.11937716
## 96   0.6934503   0.15584416    0.12793177
## 97   0.7205949   0.12727273    0.13207547
## 98   0.7586250   0.08831169    0.13178295
## 99   0.8212517   0.05714286    0.13924051
## 100  0.9303040   0.02337662    0.18000000
## 101        Inf   0.00000000            NA
```

So far, we have 2 feasible models:

PCA logistic regression model and **logistics regression with filtering**.

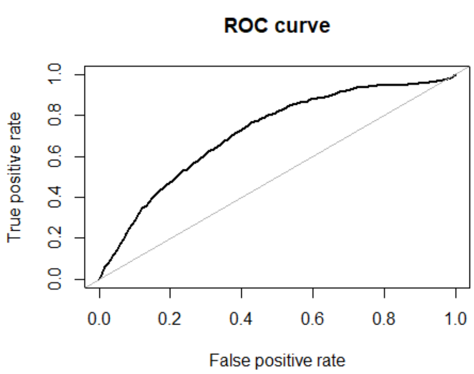
In terms of performance in numerical values, there's not really big differences, and PCA model might be more explainable since it has combined dimension measuring similar aspects of the dataset.

But what would be used in the end may depends on business strategy, and how explainable they want their model to be.

More Model Fitting and Performance Evaluation: Naïve Bayes model, top 10 features

We use the filtered dataset we got previously.

```
library(e1071)

# oversample the training set
xyzdata_normalized_drop_user_id_train_top10_oversampled <- ROSE(adopter ~ lovedTracks
+ delta_songsListened + delta_lovedTracks + subscriber_friend_cnt + songsListened + fr
iend_cnt + friend_country_cnt + delta_friend_cnt + delta_subscriber_friend_cnt + delta
_avg_friend_male, data = xyzdata_normalized_drop_user_id_train_top10, seed = 123)$data

# train the model
NB_model_oversample <- naiveBayes(adopter ~ lovedTracks + delta_songsListened + delta_
lovedTracks + subscriber_friend_cnt + songsListened + friend_cnt + friend_country_cnt
+ delta_friend_cnt + delta_subscriber_friend_cnt + delta_avg_friend_male, data = xyzda
ta_normalized_drop_user_id_train_top10_oversampled)

# predict
preb_prob_nb_oversample <- predict(NB_model_oversample, xyzdata_normalized_drop_user_i
d_test_top10, type = "raw")
# class probability predictions by setting type = "raw"

# get auc
library(pROC)

xyzdata_normalized_test_roc_curve_NB <- xyzdata_normalized_drop_user_id_test_top10 %>%
 mutate(prob = preb_prob_nb_oversample[, "1"]) %>% arrange(desc(prob)) %>% mutate(yes_
1 = ifelse(adopter == "1", 1, 0))

roc_nb <- roc.curve(xyzdata_normalized_test_roc_curve_NB$yes_1, xyzdata_normalized_tes
t_roc_curve_NB$prob)
```




```
auc_nb <- roc_nb$auc
auc_nb

## [1] 0.7143375
```

Generate a dataframe of cutoff and corresponding recall_p and precision:

```r
# initialize the dataframe
dashboard_filter <- data.frame()

# initialize vectors
cutoff_filter <- c()
precision_filter <- c()
recall_p_filter <- c()

# for loop to get corresponding recall_p and precision for each cutoff value
threshold <- roc_nb$thresholds
for (i in 1:(length(threshold))){
  cutoff_filter <- c(cutoff_filter, threshold[i])
  binary_predictions <- ifelse(xyzdata_normalized_test_roc_curve_NB$prob >= threshold
[i], 1, 0)
  confusion_matrix <- confusionMatrix(data = factor(binary_predictions), reference = x
yzdata_normalized_drop_user_id_test_top10$adopter, mode = "prec_recall", positive = "1
")
  recall_p_filter <- c(recall_p_filter, roc_nb$true.positive.rate[i])
  precision_filter <- c(precision_filter, confusion_matrix$byClass[["Precision"]])
}
```

```
## Warning in confusionMatrix.default(data = factor(binary_predictions), reference
## = xyzdata_normalized_drop_user_id_test_top10$adopter, : Levels are not in the
## same order for reference and data. Refactoring data to match.

## Warning in confusionMatrix.default(data = factor(binary_predictions), reference
## = xyzdata_normalized_drop_user_id_test_top10$adopter, : Levels are not in the
## same order for reference and data. Refactoring data to match.
```

```r
dashboard_filter <- data.frame(cutoff_filter, recall_p_filter, precision_filter)
dashboard_filter
```

```
##     cutoff_filter recall_p_filter precision_filter
## 1            -Inf      1.00000000       0.03707270
## 2     0.004737460      1.00000000       0.03709413
## 3     0.009485813      0.98441558       0.03662825
## 4     0.009503937      0.97662338       0.03618259
## 5     0.009517463      0.97142857       0.03602435
## 6     0.009529119      0.96883117       0.03592995
## 7     0.009538183      0.96623377       0.03587763
## 8     0.009545250      0.96363636       0.03566290
## 9     0.009551466      0.96103896       0.03538446
## 10    0.009557168      0.95844156       0.03538429
## 11    0.009562638      0.95844156       0.03543391
## 12    0.009567193      0.95584416       0.03543053
## 13    0.009571597      0.95324675       0.03569491
## 14    0.009576207      0.95324675       0.03559303
## 15    0.009580279      0.95064935       0.03561917
## 16    0.009583859      0.94805195       0.03558959
```

```
## 17    0.009587311      0.94545455      0.03521769
## 18    0.009591924      0.94545455      0.03467390
## 19    0.009597941      0.94545455      0.03468001
## 20    0.009605191      0.94545455      0.03435345
## 21    0.009611979      0.94545455      0.03464740
## 22    0.009618957      0.94025974      0.03404839
## 23    0.009628157      0.94025974      0.03410814
## 24    0.009637253      0.93506494      0.03411569
## 25    0.009648771      0.93506494      0.03445138
## 26    0.009662532      0.93506494      0.03424207
## 27    0.009677461      0.92987013      0.03417891
## 28    0.009695324      0.92467532      0.03425863
## 29    0.009714154      0.92207792      0.03406126
## 30    0.009733600      0.91688312      0.03388889
## 31    0.009756464      0.91428571      0.03401553
## 32    0.009787336      0.90649351      0.03397362
## 33    0.009823038      0.89870130      0.03385152
## 34    0.009861691      0.89610390      0.03377083
## 35    0.009907808      0.89090909      0.03339823
## 36    0.009956613      0.88571429      0.03359684
## 37    0.010008897      0.88311688      0.03338485
## 38    0.010066649      0.88051948      0.03344324
## 39    0.010138505      0.87792208      0.03348214
## 40    0.010220806      0.86753247      0.03337113
## 41    0.010298164      0.86493506      0.03321276
## 42    0.010366707      0.86493506      0.03312699
## 43    0.010439287      0.85714286      0.03355476
## 44    0.010520704      0.85194805      0.03348950
## 45    0.010609542      0.84675325      0.03352744
## 46    0.010718839      0.83896104      0.03323751
## 47    0.010839210      0.82597403      0.03370581
## 48    0.010983349      0.82077922      0.03412906
## 49    0.011141918      0.81038961      0.03434535
## 50    0.011323025      0.80779221      0.03465634
## 51    0.011549724      0.80000000      0.03417621
## 52    0.011772317      0.79220779      0.03421212
## 53    0.011984585      0.78441558      0.03473079
## 54    0.012213265      0.77402597      0.03486662
## 55    0.012456509      0.77142857      0.03410553
## 56    0.012762070      0.76363636      0.03436953
## 57    0.013115733      0.74805195      0.03425118
## 58    0.013454969      0.74025974      0.03501144
## 59    0.013882315      0.72727273      0.03546266
## 60    0.014382368      0.71688312      0.03585178
## 61    0.014877930      0.70909091      0.03594289
## 62    0.015434145      0.69610390      0.03599295
## 63    0.016071958      0.68311688      0.03540967
## 64    0.016779529      0.67532468      0.03429027
## 65    0.017555894      0.65974026      0.03437926
## 66    0.018402189      0.64415584      0.03489026
## 67    0.019343613      0.63636364      0.03353570
```

```
## 68    0.020638724      0.63116883       0.03419566
## 69    0.022198947      0.62077922       0.03434529
## 70    0.023868783      0.60000000       0.03424223
## 71    0.025751433      0.58701299       0.03425775
## 72    0.027571764      0.57662338       0.03392677
## 73    0.029191442      0.56623377       0.03462749
## 74    0.031025307      0.55064935       0.03546869
## 75    0.032696920      0.53766234       0.03450863
## 76    0.034293210      0.53246753       0.03333333
## 77    0.036537380      0.51428571       0.03418803
## 78    0.039435449      0.49870130       0.03316327
## 79    0.043445995      0.48571429       0.03365810
## 80    0.048210969      0.47012987       0.03438662
## 81    0.053358911      0.45974026       0.03468490
## 82    0.059606376      0.44155844       0.03529412
## 83    0.067786786      0.43116883       0.03554120
## 84    0.077607126      0.41818182       0.03321879
## 85    0.089954472      0.40000000       0.03319252
## 86    0.109988249      0.37922078       0.03288201
## 87    0.139017633      0.35844156       0.03434066
## 88    0.177014130      0.34805195       0.03555556
## 89    0.235569735      0.32467532       0.03500398
## 90    0.312439048      0.29090909       0.03231441
## 91    0.409380343      0.27272727       0.03317536
## 92    0.525413806      0.24935065       0.03361345
## 93    0.662070389      0.22597403       0.03600465
## 94    0.806575333      0.19740260       0.03145478
## 95    0.917748935      0.17402597       0.03007519
## 96    0.976471927      0.14545455       0.02469136
## 97    0.994811827      0.11948052       0.02597403
## 98    0.999233866      0.09870130       0.03191489
## 99    0.999950083      0.07792208       0.03484321
## 100   0.999999923      0.05974026       0.03626943
## 101           Inf      0.00000000               NA
```

Precision, specifically, is not performed good in naive Bayes model.

Summary and Suggestions for Business solutions outline

From the analysis above, we can see that the overall AUC of possible adjustments we've applied on the models are around 0.73, and for company to perform business strategies, we suggest that they also consider Recall_positive and Precision. What we will suggest for business strategy is that, pick the model that is more explainable and feasible in business scope with acceptable-to-nice numerical value of performance. Here we suggest:

logistic regression with oversampled training set + feature selection(top 10 / PCA)

Again, the following figure helps us identify the best model we've got.

Metric / Model types	**Oversampled training set**	**Feature selection**	**AUC**	**Recall_positive**	**Precision**	**Relative advantages**
Logistic regression (Best performance among all)	Y	PCA (dimension reduction)	**0.72**	**0.88** (Ranges with threshold)	**0.06**	Less dimensions but can keep all the factors: More easily explainable, not dropping information
	Y	Filter (top 10 features)	**0.74**	**0.89** (Ranges with threshold)	**0.06**	Being able to focus on top 10 important variables
	N	N	0.71	0.78 (Ranges with threshold)	0.06	N
K-NN model	Y	Filter (top 10 features)	0.70	0.44	0.08	N
Decision tree	Y	Filter (top 10 features)	0.61	0.30	0.11	N
Naïve Bayes model	Y	Filter (top 10 features)	0.71	(Ranges with threshold)	Always around 0.034	N

Appendix: more model tuning and selection

Decision tree, filtering top 10 features on oversampled training set

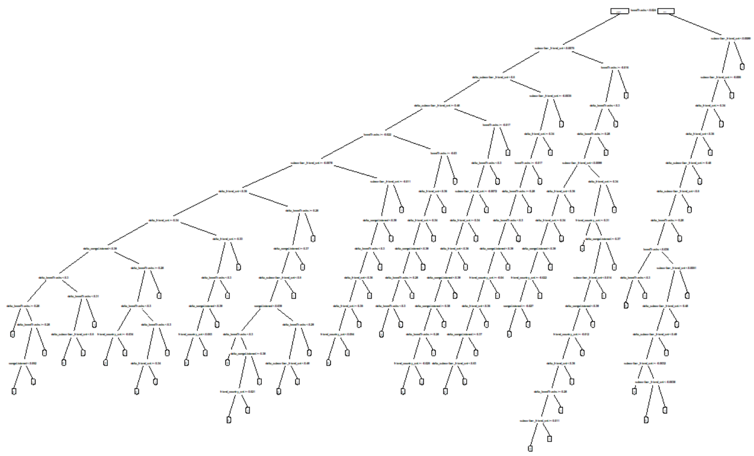
```
# fit the model
# split = "information" means we want to determine splits based on information gain
library(rpart)
tree <- rpart(adopter ~ lovedTracks + delta_songsListened + delta_lovedTracks + subscr
iber_friend_cnt + songsListened + friend_cnt + friend_country_cnt + delta_friend_cnt +
 delta_subscriber_friend_cnt + delta_avg_friend_male, data = xyzdata_normalized_drop_u
ser_id_train_top10_oversampled, method = "class", parms = list(split = "information"),
 control = list(minsplit = 2, maxdepth = 500, cp = 0.0005))

library(rpart.plot)
prp(tree, varlen = 0)

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

```
pred <- predict(tree, xyzdata_normalized_drop_user_id_test_top10, type = "class")
confusion_matrix_tree <- confusionMatrix(data = factor(pred), reference = factor(xyzda
ta_normalized_drop_user_id_test_top10$adopter), mode = "prec_recall", positive = "1")

confusion_matrix_tree

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9044  269
##          1  956  116
##
##                Accuracy : 0.882
##                  95% CI : (0.8757, 0.8882)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1107
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.10821
##                  Recall : 0.30130
##                      F1 : 0.15923
##              Prevalence : 0.03707
##          Detection Rate : 0.01117
##    Detection Prevalence : 0.10323
##       Balanced Accuracy : 0.60285
##
##        'Positive' Class : 1
##
```

Not really good performance, Precision and recall for tree model are too low.

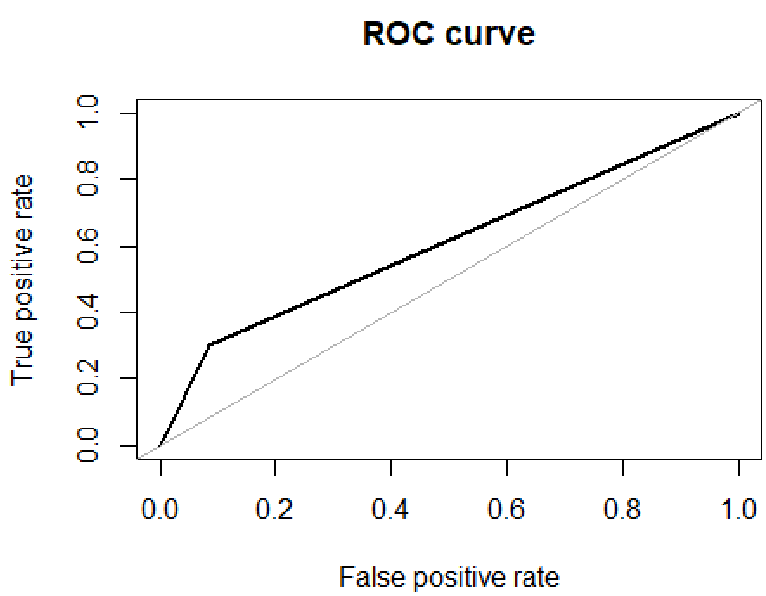
Precision : 0.10821

Recall : 0.30130



And the tree plot is not readable since it's a relatively larger dataset.

```
# auc
xyzdata_normalized_roc_curve_tree <- roc.curve(xyzdata_normalized_drop_user_id_test_to
p10$adopter, pred)
```

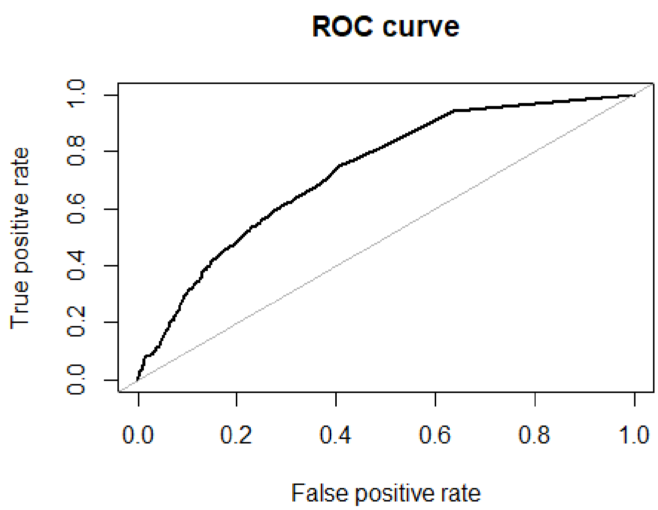



```
# plot(xyzdata_normalized_roc_curve_knn)
xyzdata_normalized_roc_curve_tree$auc

## [1] 0.6093481
```

Knn, filtering top 10 features on oversampled training set

```r
# fit the model
library(kknn)

model_knn <- kknn(adopter ~ lovedTracks + delta_songsListened + delta_lovedTracks + su
bscriber_friend_cnt + songsListened + friend_cnt + friend_country_cnt + delta_friend_c
nt + delta_subscriber_friend_cnt + delta_avg_friend_male, train = xyzdata_normalized_d
rop_user_id_train_top10_oversampled, test = xyzdata_normalized_drop_user_id_test_top10,
 k = 500, distance = 2, kernel = "rectangular")
pred_prob_knn <- model_knn$prob

# auc
xyzdata_normalized_roc_curve_knn <- roc.curve(ifelse(xyzdata_normalized_drop_user_id_t
est_top10$adopter == '1', 1, 0), pred_prob_knn[, "1"])

# plot(xyzdata_normalized_roc_curve_knn)
xyzdata_normalized_roc_curve_knn$auc

## [1] 0.7063965
```

Not bad AUC. But if checking recall_p and precision, ...




```r
confusion_matrix_knn <- confusionMatrix(data = factor(ifelse(pred_prob_knn[, "1"] > 0.
2, 1, 0)), reference = xyzdata_normalized_drop_user_id_test_top10$adopter, mode = "pre
c_recall", positive = "1")

confusion_matrix_knn

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 8162  214
##          1 1838  171
##
```

```
##                Accuracy : 0.8024
##                  95% CI : (0.7946, 0.81)
##     No Information Rate : 0.9629
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.086
##
##  Mcnemar's Test P-Value : <2e-16
##
##               Precision : 0.08512
##                  Recall : 0.44416
##                      F1 : 0.14286
##              Prevalence : 0.03707
##          Detection Rate : 0.01647
##    Detection Prevalence : 0.19345
##       Balanced Accuracy : 0.63018
##
##        'Positive' Class : 1
##
```

Precision : 0.08512
Recall : 0.44416
Not performing well even if we've tune the threshold low.

Oversampling before Filtering

Since the data set is severely imbalanced, we oversample the whole training dataset, hoping the filtering procedure could capture more information.

```
# oversample the whole training dataset
library(ROSE)
xyzdata_rose_whole_train <- ROSE(adopter ~., data = xyzdata_normalized_drop_user_id_tr
ain, seed = 123)$data

# xyzdata_normalized_drop_user_id_test: untouched

# filtering using oversampled training set
library(FSelectorRcpp)
IG_oversample_whole_train <- information_gain(adopter ~ ., data = xyzdata_rose_whole_t
rain)

# select top 10
top10_oversampled <- cut_attrs(IG_oversample_whole_train, k = 10)

# oversampled training set
xyzdata_normalized_top_10_oversampled_train <- xyzdata_rose_whole_train %>% select(top
10_oversampled, adopter)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## ℹ Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(top10_oversampled)
##
##   # Now:
##   data %>% select(all_of(top10_oversampled))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
# untouched testing, no oversampled
xyzdata_normalized_drop_user_id_test_top_10 <- xyzdata_normalized_drop_user_id_test %>%
% select(top10_oversampled, adopter)

# the whole dataset, no oversampled
xyzdata_normalized_drop_user_id_top_10 <- xyzdata_normalized_drop_user_id %>% select(t
op10_oversampled, adopter)

colnames(xyzdata_normalized_drop_user_id_top_10)
```

```
##  [1] "delta_shouts"                "delta_posts"
##  [3] "playlists"                   "delta_good_country"
##  [5] "lovedTracks"                 "delta_subscriber_friend_cnt"
##  [7] "subscriber_friend_cnt"       "delta_songsListened"
##  [9] "delta_lovedTracks"           "delta_friend_cnt"
## [11] "adopter"
```

Yet then we realized it's not stable since each time the top 10 variables are not exactly the same.

So we decided not to use it in the end.