title: “Music Recommendation System”

author: “Prem Kumar Kamasani”

date: “December 14, 2017”

output: html_document

–Project Benefits to Company-

Recommender system suggests songs to the user based on the songs they have heard before. It provides a lot of advantages to the company as it engages the user and helps deliver relevant content, which makes the application user friendly. It also increases the time spent on the application by the user, it helps offer advice and direction to the user.

Abstract

Recommender systems use algorithms to provide users with product or service recommendations. Recently these systems have been using Machine Learning algorithms from the field of Artificial Intelligence. Recommender systems are used to help users find new items or services, such as books, music, transportation or even people based on information about the user or the recommended item. These systems also play an important role in decision making, helping users to maximize profits or minimize risks.

Introduction

The data has been taken from a competition on Kaggle which was hosted by KKBOX. It is a Music Recommendation System that challenges to provide more accuracy at the prediction of the user’s Music choice.

Not many years ago, it was inconceivable that the same person would listen to the Beatles, Vivaldi, and Lady Gaga on their morning commute. But, the glory days of Radio DJs have passed, and musical gatekeepers have been replaced with personalizing algorithms and unlimited streaming services.

WSDM has challenged the Kaggle ML community to help solve these problems and build a better music recommendation system. The dataset is from KKBOX, Asia's leading music streaming service, holding the world's most comprehensive Asia-Pop music library with over 30 million tracks. They currently use a collaborative filtering based algorithm with matrix factorization and word embedding in their recommendation system but believe new techniques could lead to better results.

The analysis we have performed on this dataset is using the train.csv file.

This file contains the following attributes.

train.csv

msno: user id song_id: song id source_system_tab: the name of the tab where the event was triggered. System tabs are used to categorize KKBOX mobile apps functions. source_screen_name: name of the layout a user sees. source_type: an entry point a user first plays music on mobile apps. target: this is the target variable. target=1 means there are recurring listening event(s) triggered within a month after the user's very first observable listening event, target=0 otherwise.

We are predicting the target value and finding out whether the user has heard the song in the period of one month or not.Hence we will recommend the song that has been heard by the user more than once.

Literature Survey

The use of Machine Learning Algorithms in Recommendation Systems: A systematic review. By Ivens Portugal, Paulo Alencar and Donald Covan.

This paper forms the basis of our algorithms that we intend to use here. We extracted the Machine Learning Algorithms from here.

https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/

This paper explains how how recommendation works and creates basic popularity model and a collaborative filtering model.

https://www.analyticsvidhya.com/blog/2015/11/easy-methods-deal-categorical-variables-predictive-modeling/

This paper helps deal with the categorical variables of the data and provides various approaches to handle these variables.

Project Methodology

We are using the train.csv table perform our predictions on this dataset.

We faced a few challenges while using categorical data, and hence tried to overcome them by converting them into numerical values. A brief overview of the challenges faced are listed here

-Since categorical data has too many levels, it pulls down performance of the model. -In case the categorical values are masked, it becomes very difficult to decipher their meaning. -We can't fit categorical variable into the regression equation, it must be treated differently. -Most of the algorithms provide better results with numerical variables.

We are dividing the data into train and test in 70 and 30 ratio.

We then apply Linear regression model, Logistic model, Decision Tree model and Random forest on train data to derive the maximum level of accuracy for this on the test data and predict the target values in test data. We extract the accuracy and sensitivity and select the best model from this information.

When the target value is predicted to be 1 it will be recommended to the user.

The Case Study

A major challenge that we faced during the implementation was while trying to execute the categorical variables as they were generating a lot of levels and we had to convert it into numerical variables for easier implementation and more accuracy.

We converted the data to numerical data by

```
# Reading data
data<- read.csv(file.choose(), header = T)

str(data)

## 'data.frame':    7377418 obs. of  6 variables:
##  $ msno            : Factor w/ 30755 levels
"//4hBneqk/4/TtgL1XXQ+eKx7fJTeSvSNt0ktxjSIYE=",..: 10819 28630 28630
28630 10819 10819 28630 10819 25303 25303 ...
##  $ song_id         : Factor w/ 359966 levels
"///2+ie5adFFlKPB/f3WwbFA2VXJQtnACfTg6qHz6mI=",..: 81924 83974
176420 23724 32516 30823 309907 86694 1360 115721 ...
##  $ source_system_tab : Factor w/ 10 levels "","discover",..: 3 5 5 5 3 3 5 3
5 5 ...
##  $ source_screen_name: Factor w/ 21 levels "","Album more",..: 9 10 10
10 9 9 10 9 10 10 ...
##  $ source_type     : Factor w/ 13 levels "","album","artist",..: 8 6 6 6 8 8 6
8 5 5 ...
##  $ target          : int  1 1 1 1 1 1 1 1 1 1 ...
```

The data contains 7377418 observations and 6 variables 5 variables are factors and the target variable is of the type int. For further analysis we have to convert target variable to a factor since the number of levels are huge and it becomes challenging to implement the models and also convert the predictor variables to factor variables.

```
# converting types of variables
data$msno=as.numeric(data$msno)
data$song_id=as.numeric(data$song_id)
data$source_system_tab=as.numeric(data$source_system_tab)
data$source_screen_name=as.numeric(data$source_screen_name)
data$source_type=as.numeric(data$source_type)
```

```
data$target=as.factor(data$target)
str(data)
```

```
## 'data.frame':    7377418 obs. of  6 variables:
##  $ msno             : num  10819 28630 28630 28630 10819 ...
##  $ song_id          : num  81924 83974 176420 23724 32516 ...
##  $ source_system_tab : num  3 5 5 5 3 3 5 3 5 5 ...
##  $ source_screen_name: num  9 10 10 10 9 9 10 9 10 10 ...
##  $ source_type       : num  8 6 6 6 8 8 6 8 5 5 ...
##  $ target           : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

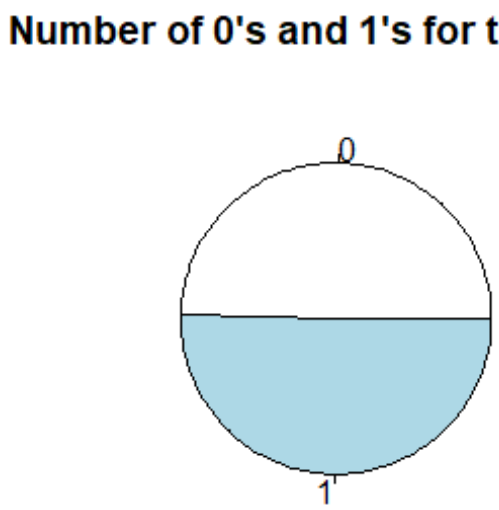
We have now converted the predictor variables to num and the target value to factor

Partitioning the data into train and test models in the ratio 70:30.

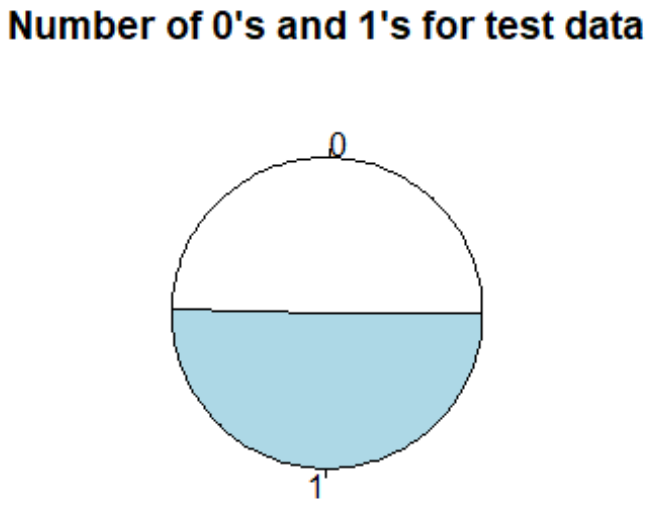
```
ind<-sample(2,nrow(data),replace=T, prob = c(0.70,0.30))
train<-data[ind==1,]
test<-data[ind==2,]

table(train$target)
```

```
##
##       0       1
## 2563765 2600389
```

```
pie(table(train$target),c(0,1),main = "Number of 0's and 1's for train data")
```



```
table(test$target)

##
##       0       1
## 1098997 1114267

pie(table(test$target),c(0,1),main = "Number of 0's and 1's for test data")
```



This is to find the ratio of 1's and 0's in the target variable. The proportion is almost equal and hence the data can be used for further analysis now.

```
table(train$source_screen_name)

##
##     1     2     3     4     5     6     7     8     9
##  290920  294142  176767    36  149696  170960   57640   11217
50734
##    10    11    12    13    14    15    16    17    18
## 2258757   53244   4515  905603  141053    8  332506  209049   9599
##    19    20    21
##   9585    140   37983

table(train$source_system_tab)

##
##     1     2     3     4     5     6     7     8     9
##  12941 1525914  117504  148435 2578944   4349   4499  334121
```

```
435933
##     10
##   1514

table(train$source_type)

##
##      1      2      3      4      5      6      7      8      9
##  15115  334099   2106  134798 1582437  755315    455 1377640
338565
##     10     11     12     13
##  171542  147309  296912   7861
```

The source_screen_name contains data from all the factors that are available and hence calls for a good test data to implement the analysis.

We tried implementing the following models - Linear Regression Model

```
#linear regression model
linearmodel<-
glm(target~msno+song_id+source_system_tab+source_screen_name+sourc
e_type,data = train, family = 'binomial')
summary(linearmodel)

##
## Call:
## glm(formula = target ~ msno + song_id + source_system_tab +
source_screen_name +
##     source_type, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4958  -1.1456   0.9324   1.1396   1.5167
##
## Coefficients:
##                     Estimate Std. Error  z value Pr(>|z|)
## (Intercept)        8.061e-01  4.211e-03  191.412  < 2e-16 ***
## msno              -4.006e-08  9.961e-08   -0.402    0.688
## song_id            8.327e-09  8.526e-09    0.977    0.329
## source_system_tab   3.164e-03  4.214e-04    7.508 5.98e-14 ***
## source_screen_name -1.234e-02  2.105e-04  -58.602  < 2e-16 ***
## source_type        -1.018e-01  3.693e-04 -275.574  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7158778  on 5164153  degrees of freedom
## Residual deviance: 7071805  on 5164148  degrees of freedom
## AIC: 7071817
```

```
##
## Number of Fisher Scoring iterations: 4
```

Linear Regression model is run on all predictor variables and the model is stored in a variable called linearmodel.

From the summary we can gather that msno and song_id variables are not significant, hence excluding them from the analysis might not affect the accuracy of the model.

```
linearmodel<-
glm(target~source_system_tab+source_screen_name+source_type,data =
train, family = 'binomial')
summary(linearmodel)

##
## Call:
## glm(formula = target ~ source_system_tab + source_screen_name +
##     source_type, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4950  -1.1458   0.9327   1.1399   1.5163
##
## Coefficients:
##                    Estimate Std. Error  z value Pr(>|z|)
## (Intercept)        0.8070024  0.0036167  223.134  < 2e-16 ***
## source_system_tab   0.0031590  0.0004213    7.498  6.5e-14 ***
## source_screen_name -0.0123356  0.0002105  -58.605  < 2e-16 ***
## source_type        -0.1017558  0.0003693 -275.574  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7158778  on 5164153  degrees of freedom
## Residual deviance: 7071806  on 5164150  degrees of freedom
## AIC: 7071814
##
## Number of Fisher Scoring iterations: 4
```

Linear Model is now built excluding these two variables.

```
p<-predict(linearmodel,train)
linearpredict<-ifelse(p>0.5,1,0)
tab1=table(predicted=linearpredict,actual=train$target)
tab1

##         actual
## predicted       0       1
```

```
##         0 2363669 2469993
##         1  200096  130396
```

From the above developed Linear model we predicted the target variable of the train data, if the probability is greater than 0.5 then target is 1, else 0. Confusion matrix is developed on predictions and train data and the values are stored in tab1 variable.

2367542 values are predicted correctly as 0 and 127645 values are predicted correctly as 1 196851 values are wrongly predicted as 0 and 127645 are wrongly predicted as 1

```
p<-predict(linearmodel,test)
linearpredict<-ifelse(p>0.5,1,0)
tab1<-table(predicted=linearpredict,actual=test$target)
tab1
```

```
##         actual
## predicted      0      1
##         0 1012981 1058677
##         1   86016   55590
```

```
accuracy=sum(diag(tab1))/sum(tab1)
accuracy
```

```
## [1] 0.4828032
```

The accuracy for this model is 48.302%

Next model we implemented was Logistic Model

```
library(nnet)
logisticmodel<-
multinom(target~source_screen_name+source_system_tab+source_type,dat
a = train)
```

```
## # weights:  5 (4 variable)
## initial  value 3579518.785259
## final  value 3535902.877983
## converged
```

```
predict=predict(logisticmodel,train)
tab1=table(predict,train$target)
tab1
```

```
##
## predict      0      1
##       0 1419484  918425
##       1 1144281 1681964
```

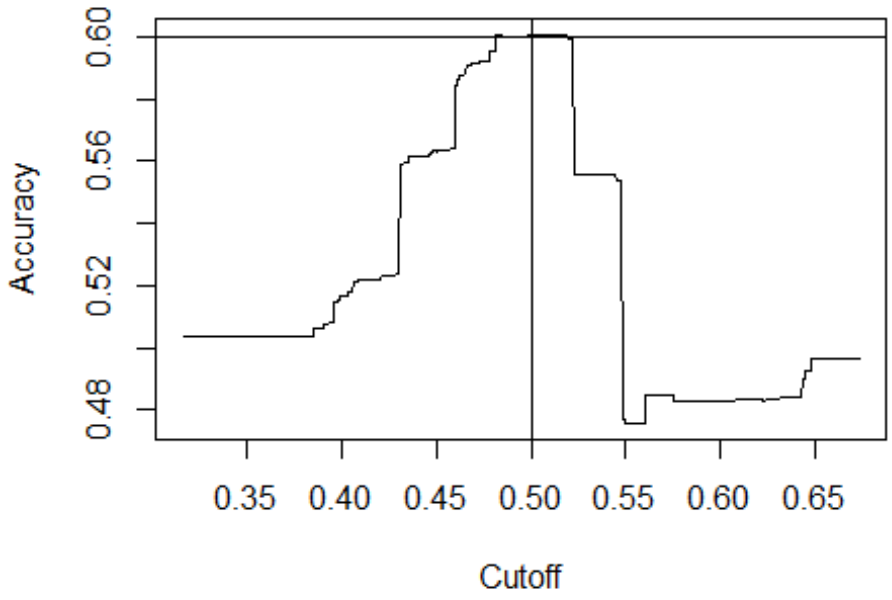
This is the logistic regression model to predict the target variable of the train data.

The Confusion Matrix for this model has 1418774 values correctly predicted as 0 and 917921 values wrongly predicted.

```
accuracy=sum(diag(tab1))/sum(tab1)
accuracy

## [1] 0.6005723

predict=predict(logisticmodel,test)
tab1=table(predict,test$target)
tab1

##
## predict      0      1
##       0 607212 393152
##       1 491785 721115

accuracy=sum(diag(tab1))/sum(tab1)
accuracy

## [1] 0.6001665
```

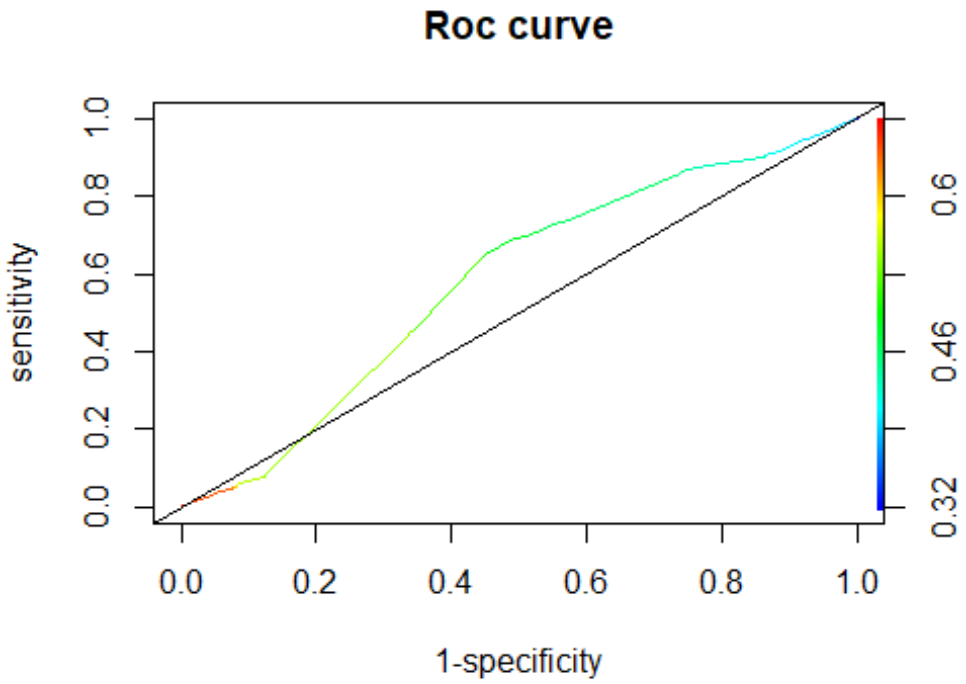
The train and test accuracy levels is given to be 60.05%

```
library(ROCR)

## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

pred <- predict(logisticmodel, train, type="prob")
pred <- prediction(pred, train$target)
eval <- performance(pred,"acc")
plot(eval)
abline(h=0.60, v=0.50)
```



We are evaluating this model to produce some refinement to the accuracy. We are assuming that the model would take the target variable to be 1 for all the values that have the probability about 0.5 and less than that is considered as 0.

```
roc<- performance(pred, "tpr","fpr")
plot(roc,colorize=T,main="Roc curve", ylab="sensitivity", xlab="1-
specificity")
abline(0,1)
```

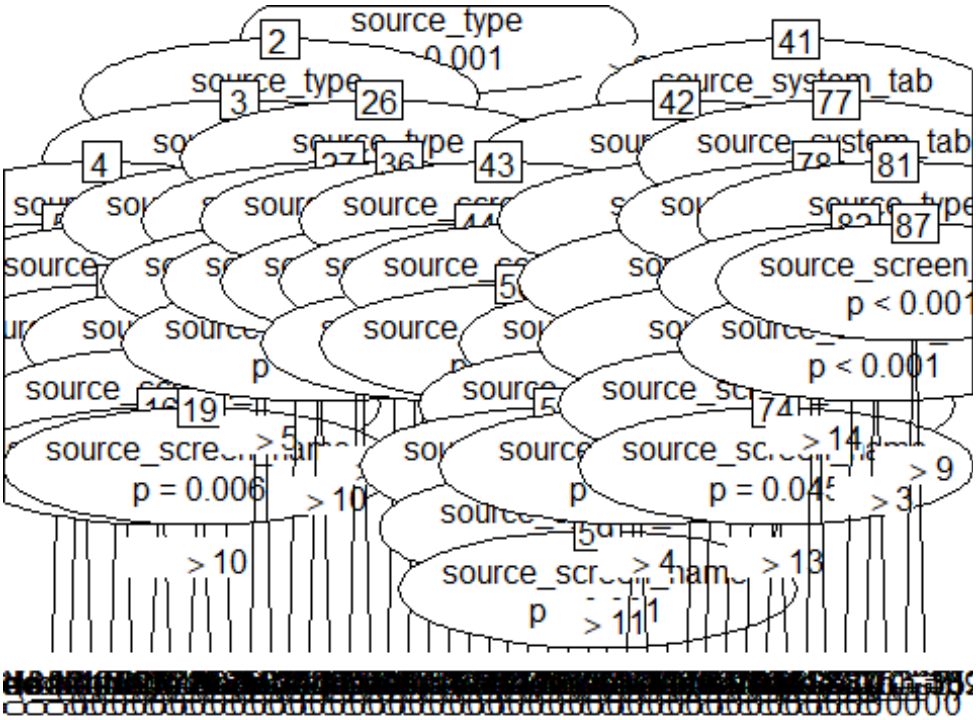


In this as we have predicted the peak values are close to 0.5 and hence the sensitivity and specificity do not play a major role in this curve. It is more or less the same as we started off with and hence does not affect the accuracy very much.

Next model is the Decision Tree Model

```
#decision trees using party

ind1<-sample(2,nrow(data),replace=T, prob = c(0.1,0.9))
sample<-data[ind1==1,]
ind2<-sample(2,nrow(sample),replace=T, prob=c(0.8,0.2))
sampletrain<-sample[ind2==1,]
sampletest<-sample[ind2==2,]

library(party)

## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

treemodel=ctree(target~source_system_tab+source_screen_name+source_t
ype,data=sampletrain)
summary(treemodel)

##     Length      Class       Mode
##          1 BinaryTree         S4

plot(treemodel)
```



Running this model on the entire dataset was very difficult and was unable to produce results hence we partitioned the data and took only 10% of the entire data Of this 10% we divided it into a train dataset of 80% and test dataset of 20% and implemented the decision tree model.

The decision tree is plotted above.

```
predict<-predict(treemodel,sampletrain)
tab2=table(predicted=predict,actual=sampletrain$target)
tab2
```

```
##         actual
## predicted      0      1
##         0 186615 114976
##         1 106804 183715
```

```
accuracy=sum(diag(tab2))/sum(tab2)
accuracy
```

```
## [1] 0.6254412
```

```
predict<-predict(treemodel,sampletest)
tab2=table(predicted=predict,actual=sampletest$target)
accuracy=sum(diag(tab2))/sum(tab2)
accuracy
```

```
## [1] 0.6252538
```

The decision tree correctly predicts 186228 0 values correctly and has the wrong prediction for 114088 1 values. The accuracy for the train model is 62.62% and the accuracy for the test model is 62.37%

The sensitivity is similar as the previous model as it has predicted the 0 and 1 values almost similarly.

Now we implemented the Random Forest model

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
fit <-
randomForest(target~source_system_tab+source_screen_name+source_type
, data=sampletrain, ntree=70)
print(fit) # view results
```

```
##
## Call:
##  randomForest(formula = target ~ source_system_tab +
source_screen_name +      source_type, data = sampletrain, ntree = 70)
##               Type of random forest: classification
##                     Number of trees: 70
## No. of variables tried at each split: 1
##
##        OOB estimate of  error rate: 37.48%
## Confusion matrix:
##        0      1 class.error
## 0 187044 106375   0.3625362
## 1 115521 183170   0.3867576
```

```
importance(fit) # importance of each predictor
```

```
##                   MeanDecreaseGini
## source_system_tab         7056.320
## source_screen_name        8055.561
## source_type               7142.507
```

This model correctly predicts the 188188 values to be 0 and wrongly predicts 116094 values to be 1. Importance shows the importance for each variable.

```
predict<-predict(fit,sampletrain)
tab3<-confusionMatrix(predict,sampletrain$target)
tab3
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction      0      1
##          0 187002 115296
##          1 106417 183395
##
##                Accuracy : 0.6256
##                  95% CI : (0.6243, 0.6268)
##     No Information Rate : 0.5045
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2512
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6373
##             Specificity : 0.6140
##          Pos Pred Value : 0.6186
##          Neg Pred Value : 0.6328
##              Prevalence : 0.4955
##          Detection Rate : 0.3158
##    Detection Prevalence : 0.5105
```

```
##       Balanced Accuracy : 0.6257
##
##        'Positive' Class : 0
##
```

```
predict<-predict(fit,sampletest)
tab3<-confusionMatrix(predict,sampletest$target)
tab3
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction     0     1
##          0 46518 28456
##          1 26730 45585
##
##                Accuracy : 0.6253
##                  95% CI : (0.6228, 0.6278)
##     No Information Rate : 0.5027
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2507
##  Mcnemar's Test P-Value : 2.088e-13
##
##             Sensitivity : 0.6351
##             Specificity : 0.6157
##          Pos Pred Value : 0.6205
##          Neg Pred Value : 0.6304
##              Prevalence : 0.4973
##          Detection Rate : 0.3158
##    Detection Prevalence : 0.5090
##       Balanced Accuracy : 0.6254
##
##        'Positive' Class : 0
##
```

The accuracy for this model is 62.62%. The sensitivity is 64% and specificity turns out to be 60.9% for train data and the accuracy for test data 62.36% with sensitivity as 64.27 and specificity as 60.49.

Discussion of results

Linear Regression accuracy

Train - 48.3187% Test - 48.3026%

Logistic Regression

Train - 60.040015% Test - 60.0568%

Decision Trees

Train - 62.623858% Test - 62.374014%

Random Forest

Train - 62.62521% Test - 62.36522%

Decision Tree and Random Forest provide good accuracy for this model as compared to Logistic and Linear regression.

Model is run on a small dataset, however it projects the accuracy on the larger dataset.

Summary

Decision Trees and Random Forest have generated approximately equal accuracy and the next best model is Logistic Regression. Random Forest generated only 70 trees, however with increase in the number of trees the accuracy is expected to increase for this model. The sensitivity for all the models except Linear Regression is also almost equal. Linear regression could predict only the 0 values accurately however the 1 values were misclassified. It is suggested to use Random forest and Decision Trees for this data in terms of accuracy, sensitivity and specificity.

Lessons Learned

Working on unbiased samples in case of huge datasets is recommended. In case of the availability of large number of factor variables we must convert them into numeric values, even though this might decrease the accuracy it becomes close to impossible to develop a model with huge number of factors.

Conclusion The data we chose to work on was the Music Recommendation Challenge from Kaggle. The motive of this project to recommend music to the users. To do so, we are predicting target variables of values 1 and 0. We applies various models to extract a good level of accuracy.

Decision Tree and Random Forest are best fit models for this data and have the highest level of accuracy.

References The use of Machine Learning Algorithms in Recommendation Systems: A systematic review. By Ivens Portugal, Paulo Alencar and Donald Covan.

https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/

https://www.analyticsvidhya.com/blog/2015/11/easy-methods-deal-categorical-variables-predictive-modeling/

Lecture videos from POM-681