Project

Akshay & Rasagnya

March 28, 2019

library(data.table)  
library(caret)  
library(xgboost)  
library(fasttime)

rm(train)

## Warning in rm(train): object 'train' not found

rm(test)

## Warning in rm(test): object 'test' not found

data\_full <- fread("/home/arjun/Downloads/data/train.csv/mnt/ssd/kaggle-talkingdata2/competition\_files/train.csv", drop = c("attributed\_time"), showProgress=F)[(.N - 50e6):.N]   
  
set.seed(500)  
train\_index <- sample(c(1:dim(data\_full)[1]),dim(data\_full)[1]\*0.6)  
train<- data\_full[train\_index,]  
test <- data\_full[-train\_index,]  
table(train$is\_attributed)

##   
## 0 1   
## 29923272 76728

#find the propotion in percentage  
round(prop.table(table(train$is\_attributed)\*100),digits = 3)

##   
## 0 1   
## 0.997 0.003

#perking into the test data set.  
table((test$is\_attributed))

##   
## 0 1   
## 19948684 51317

#propotionate  
round(prop.table(table(test$is\_attributed)\*100),digits = 3)

##   
## 0 1   
## 0.997 0.003

This shows that data is imbalanced #Data Processing and Feature Engineering :

# data without the feature engineering:  
str(train)

## Classes 'data.table' and 'data.frame': 30000000 obs. of 7 variables:  
## $ ip : int 356995 34486 93523 65746 16426 26298 249244 120657 327824 204806 ...  
## $ app : int 3 9 24 8 12 2 2 3 2 3 ...  
## $ device : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ os : int 19 43 22 20 12 19 13 13 8 19 ...  
## $ channel : int 205 107 105 145 124 236 435 205 435 173 ...  
## $ click\_time : chr "2017-11-09 13:28:16" "2017-11-09 11:51:51" "2017-11-09 15:33:27" "2017-11-09 07:41:27" ...  
## $ is\_attributed: int 0 0 0 0 0 0 0 0 0 0 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

#test data without feature engineering:  
str(test)

## Classes 'data.table' and 'data.frame': 20000001 obs. of 7 variables:  
## $ ip : int 18251 62245 76788 37185 259940 23203 111025 74666 38641 9955 ...  
## $ app : int 3 3 9 12 18 2 12 2 8 3 ...  
## $ device : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ os : int 18 70 25 13 17 13 20 22 19 8 ...  
## $ channel : int 280 379 466 205 107 477 259 219 145 379 ...  
## $ click\_time : chr "2017-11-09 00:54:23" "2017-11-09 00:54:23" "2017-11-09 00:54:23" "2017-11-09 00:54:23" ...  
## $ is\_attributed: int 0 0 0 0 0 0 0 0 0 0 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

# number of rows in each data set :

print("train")

## [1] "train"

nrow(train)

## [1] 30000000

print("test")

## [1] "test"

nrow(test)

## [1] 20000001

# best way to feature engineering is to combine both the test and train data set.

# we make use of the fasttime library to engineer time data:  
# the value to be predicted is the ~~ is\_attributed of the train data set  
y <- train$is\_attributed  
ty <- test$is\_attributed  
#make a copy of the train data set for further training before combining   
copy\_train <- 1:nrow(train)  
#combine both training and testing data set  
train\_test <- rbind(train, test, fill = T)  
#remove memory to free up the ram   
#rm(train, test); gc()  
# feature engineering  
train\_test [, `:=`(hour = hour(click\_time), min = minute(click\_time), click\_time = fastPOSIXct(click\_time))  
 ][, next\_clk := as.integer(click\_time - shift(click\_time))  
 ][, click\_time := NULL  
 ][, ip\_f := .N, by = "ip"  
 ][, app\_f := .N, by = "app"  
 ][, channel\_f := .N, by = "channel"  
 ][, device\_f := .N, by = "device"  
 ][, os\_f := .N, by = "os"  
 ][, app\_f := .N, by = "app"  
 ][, ip\_app\_f := .N, by = "ip,app"  
 ][, ip\_dev\_f := .N, by = "ip,device"  
 ][, ip\_os\_f := .N, by = "ip,os"  
 ][, ip\_chan\_f := .N, by = "ip,channel"  
 ][, c("ip", "is\_attributed") := NULL ]  
#convert the click time to hour , minute   
#---------------------------

# prepare the data and make partation:

#delete the train data set from the combined and assign to ntest.  
ntest <- xgb.DMatrix(data = data.matrix(train\_test[-copy\_train]),label = ty)  
#assign the training data set to train\_test  
train\_test <- train\_test[copy\_train]; gc()

## used (Mb) gc trigger (Mb) max used (Mb)  
## Ncells 2018913 107.9 3719000 198.7 3085943 164.9  
## Vcells 747357795 5701.9 1755575315 13394.0 1747417752 13331.8

# make the partation in training data set  
copy\_train <- caret::createDataPartition(y, p = 0.9, list = F)  
#make a new training data with is\_attributed values intact   
ntrain <- xgb.DMatrix(data = data.matrix(train\_test[copy\_train]), label = y[copy\_train])  
#valuation data set or Testing data set  
nval <- xgb.DMatrix(data = data.matrix(train\_test[-copy\_train]), label = y[-copy\_train])  
cols <- colnames(train\_test)  
  
#rm(train\_test, y, copy\_train); gc()

# train the model with xgboost algorithm

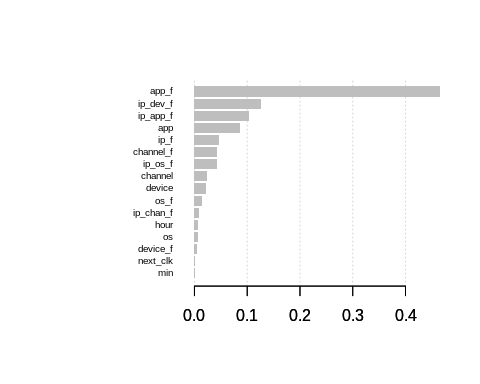
#setup the tuning parameter  
#nrouds 600 defined.  
#nrounds 2000 took more than 5 hours to execute   
#at around 600th iteration the maximum auroc was achived and the improvement after that was not significant  
#nrounds: The maximum number of iterations (number of trees in final model).  
#colsample\_bytree: The number of features, expressed as a ratio, to sample when building a tree. Default is 1 (100% of the features).  
#min\_child\_weight: The minimum weight in the trees being boosted. Default is 1.eta: Learning rate, which is the contribution of each tree to the solution. Default is 0.3.  
#gamma: Minimum loss reduction required to make another leaf partition in a tree.subsample: Ratio of data observations. Default is 1 (100%).max\_depth: Maximum depth of the individual trees.  
  
list\_xgb <- list(objective = "binary:logistic",  
 booster = "gbtree",  
 eval\_metric = "auc",  
 nthread = 8,  
 eta = 0.07,  
 max\_depth = 4,  
 min\_child\_weight = 96,  
 gamma = 6.1142,  
 subsample = 1,  
 colsample\_bytree = 0.5962,  
 colsample\_bylevel = 0.5214,  
 alpha = 0,  
 lambda= 21.0033,  
 max\_delta\_step = 5.0876,  
 scale\_pos\_weight = 150,  
 nrounds = 600)  
#training xgb model using the library xgboost   
m\_xgb <- xgb.train(list\_xgb, ntrain, list\_xgb$nrounds, list(val = nval), print\_every\_n = 50, early\_stopping\_rounds = 500)

## [1] val-auc:0.934295   
## Will train until val\_auc hasn't improved in 500 rounds.  
##   
## [51] val-auc:0.963140   
## [101] val-auc:0.967905   
## [151] val-auc:0.969916   
## [201] val-auc:0.971423   
## [251] val-auc:0.972214   
## [301] val-auc:0.972752   
## [351] val-auc:0.973026   
## [401] val-auc:0.973320   
## [451] val-auc:0.973452   
## [501] val-auc:0.973639   
## [551] val-auc:0.973819   
## [600] val-auc:0.973946

#before the final testing   
#let us check the importance   
  
(imp <- xgb.importance(cols, model=m\_xgb))

## Feature Gain Cover Frequency  
## 1: app\_f 0.4647812732 0.127868227 0.10359551  
## 2: ip\_dev\_f 0.1255781134 0.075225646 0.06056180  
## 3: ip\_app\_f 0.1031957319 0.068424975 0.07865169  
## 4: app 0.0869160079 0.143834191 0.11595506  
## 5: ip\_f 0.0473326233 0.108816645 0.08067416  
## 6: channel\_f 0.0425115589 0.082996588 0.09887640  
## 7: ip\_os\_f 0.0421232870 0.067125907 0.07022472  
## 8: channel 0.0245339929 0.039561319 0.08101124  
## 9: device 0.0213343662 0.029291145 0.01707865  
## 10: os\_f 0.0133877569 0.068408326 0.06505618  
## 11: ip\_chan\_f 0.0085831153 0.051282350 0.05168539  
## 12: hour 0.0073275775 0.049763281 0.04898876  
## 13: os 0.0058146001 0.050224467 0.04853933  
## 14: device\_f 0.0049249773 0.008404571 0.01213483  
## 15: next\_clk 0.0008822337 0.013689604 0.03662921  
## 16: min 0.0007727845 0.015082759 0.03033708

xgb.plot.importance(imp, top\_n = 30)



#examine gain , cover , frequency to get insights from the trained model   
imp

## Feature Gain Cover Frequency Importance  
## 1: app\_f 0.4647812732 0.127868227 0.10359551 0.4647812732  
## 2: ip\_dev\_f 0.1255781134 0.075225646 0.06056180 0.1255781134  
## 3: ip\_app\_f 0.1031957319 0.068424975 0.07865169 0.1031957319  
## 4: app 0.0869160079 0.143834191 0.11595506 0.0869160079  
## 5: ip\_f 0.0473326233 0.108816645 0.08067416 0.0473326233  
## 6: channel\_f 0.0425115589 0.082996588 0.09887640 0.0425115589  
## 7: ip\_os\_f 0.0421232870 0.067125907 0.07022472 0.0421232870  
## 8: channel 0.0245339929 0.039561319 0.08101124 0.0245339929  
## 9: device 0.0213343662 0.029291145 0.01707865 0.0213343662  
## 10: os\_f 0.0133877569 0.068408326 0.06505618 0.0133877569  
## 11: ip\_chan\_f 0.0085831153 0.051282350 0.05168539 0.0085831153  
## 12: hour 0.0073275775 0.049763281 0.04898876 0.0073275775  
## 13: os 0.0058146001 0.050224467 0.04853933 0.0058146001  
## 14: device\_f 0.0049249773 0.008404571 0.01213483 0.0049249773  
## 15: next\_clk 0.0008822337 0.013689604 0.03662921 0.0008822337  
## 16: min 0.0007727845 0.015082759 0.03033708 0.0007727845

#testing with validation data set:  
valid\_fit <- round(predict(m\_xgb,nval), 6)  
#we use area under roc as the performance metric:

# testng the data with test

fit\_test <- round(predict(m\_xgb,ntest), 6)

library(caret)  
fit\_test\_c <- ifelse(fit\_test > 0.5 , 1, 0)  
fit\_test\_c <- factor(fit\_test\_c)  
ty<- factor(ty)  
confusionMatrix(fit\_test\_c , ty)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 19616236 7319  
## 1 332448 43998  
##   
## Accuracy : 0.983   
## 95% CI : (0.983, 0.9831)  
## No Information Rate : 0.9974   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.2021   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.9833   
## Specificity : 0.8574   
## Pos Pred Value : 0.9996   
## Neg Pred Value : 0.1169   
## Prevalence : 0.9974   
## Detection Rate : 0.9808   
## Detection Prevalence : 0.9812   
## Balanced Accuracy : 0.9204   
##   
## 'Positive' Class : 0   
##

fit\_test\_c <- ifelse(fit\_test > 0.6 , 1, 0)  
fit\_test\_c <- factor(fit\_test\_c)  
#ty<- factor(ty)  
confusionMatrix(fit\_test\_c , ty)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 19676856 7886  
## 1 271828 43431  
##   
## Accuracy : 0.986   
## 95% CI : (0.986, 0.9861)  
## No Information Rate : 0.9974   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.2336   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.9864   
## Specificity : 0.8463   
## Pos Pred Value : 0.9996   
## Neg Pred Value : 0.1378   
## Prevalence : 0.9974   
## Detection Rate : 0.9838   
## Detection Prevalence : 0.9842   
## Balanced Accuracy : 0.9164   
##   
## 'Positive' Class : 0   
##

Comments:

Initially the dataset was imbalanced, to balance it we incorporated some feature engineering techniques. The model was built using XGBoost Algorithm.

We have plotted the confusion matrix to find out the accuracy which was noticed to be ~98.3%. Which means the dataset is 98.3% efficient in detecting the fraudulent clicks on mobile app ads.

**Sensitivity :** How "sensitive" is the classifier to detecting positive instances?

0.9864

**Specificity:** When the actual value is negative, how often is the prediction correct?

0.8463