Project 5 - AdTracking Fraud Detection

Team 4

4/25/2017

Step 0: Load the packages, and get the directories

```
#
packages.used=c("lubridate","caret","dplyr","DMwR","ROSE","ggplot2","randomForest","rpart","rpart.plot","data.table","e1071","gridExtra","xgboost","haven","tidyverse","plyr","pROC","ROCR","data.table","DT","magrittr","corrplot","Rmisc","ggalluvial","ModelMetrics","scales","irlba","forcats","forecast","TSA","zoo")
# packages.needed=setdiff(packages.used, intersect(installed.packages()[,1], packages.used))
# if(length(packages.needed)>0){
#   install.packages(packages.needed, dependencies = TRUE)
# }
# # load packages
# library("lubridate")
# library("caret")
# library("dplyr")
# library("DMwR")
# library("ROSE")
# library("ggplot2")
# library("randomForest")
# library("rpart")
# library("rpart.plot")
# library("data.table")
# library("e1071")
# library("gridExtra")
# library("xgboost")
# library("haven")
# library("tidyverse")
# library("plyr")
# library("pROC")
# library("ROCR")
# library("data.table")
# library("DT")
# library("magrittr")
# library("corrplot")
# library("Rmisc")
# library("ggalluvial")
# library("ModelMetrics")
# require("scales")
# library("irlba")
# library("forcats")
```

```
# library("forecast")
# library("TSA")
# library("zoo")
# library("lightgbm")
```

Step 1: Load and process the data

Here, we downloaded the data from kaggle competition for this project. Because the data is too big to upload to Github. Here we just uploaded the sample data of the trainset, which is processed by Kaggle. For the entire training set, we uploaded it to the Google Drive. You can download the data from here if you want:

https://drive.google.com/drive/u/1/folders/1qDnsaZxyTxnPY89h2Gsh084l-QPqshav

```
train.data <- read.csv("../data/train_sample.csv")
test.data <- read.csv("../data/test.csv")

# For the model using SMOTE data
set.seed(1234)
smote_train <- SMOTE(is_attributed ~ ., data  = train_val)
table(smote_train$is_attributed)
```

Step 2: EDA of Train Dataset and Test Dataset

Please see our eda.Rmd in the doc folder.

From the EDA, we got some insights about the train and test dataset. Following are some conclusion we made in the EDA:

1. Application ID is definitely going to be one of the important feature to differentiate user downloaded the application or not.
2. OS feature is not an important feature for prediction.
3. IP Address could play an important role in prediction as clear diffrenation exist between 2 groups.
4. Device is not an important variable for our analysis.
5. Channel got some predictive power, and we can use this for our feature analysis.
6. Hour plays a least important feature.

Conclusion: After EDA, we select ip, app, channel, hour as our predictors.

Step 3: Model

Here, we write out the models we used in this project. Models include Naive Bayesian, Decision Tree, Random Forest, LightGBM and XgBoost.

Here is all the procedure we have for this algorithm.

```
aiveBayes <- function(TRAIN = "traindata.csv", TEST = "testdata.csv",IP=1,
APP=1, OS=1, DV=1, CH=1, TM=0)
```

```r
{
  TimeToNum <- function(x)
  {
    Hour <- as.numeric(substr(x, start = 12, stop = 13)) * 2
    HourHalf <- as.numeric(substr(x, start = 15, stop = 15))
    if (HourHalf >= 3)
      Hour <- Hour + 1
    return(Hour)
  }
  CalcProb <- function(User, deno)
  {
    if (is.na(tmp[User]))
      return(lambda / deno)
    return((tmp[User] + lambda) / deno)
  }

  TrainData <- read.csv(file = TRAIN)
  TrainData$Hour <- mapply(TimeToNum, TrainData)
  #head(TrainData)
  #table(TrainData$is_attributed)

  IpCount <- table(TrainData$ip)
  AppCount <- table(TrainData$app)
  OsCount <- table(TrainData$os)
  DvCount <- table(TrainData$device)#TrainData$device: 0,1,2,3(Others)
  ChCount <- table(TrainData$channel)
  TmCount <- table(TrainData$Hour)

  TrainData$ip <- as.character(TrainData$ip)
  TrainData$app <- as.character(TrainData$app)
  TrainData$os <- as.character(TrainData$os)
  TrainData$device <- as.character(TrainData$device)
  TrainData$channel <- as.character(TrainData$channel)
  TrainData$is_attributed <- as.numeric(TrainData$is_attributed) + 1

  lambda <- 0.1
  Prior <- rep(NA, 2)
  Prior[1] <- (sum(1 - TrainData$is_attributed) + lambda) / (nrow(TrainData)
+ lambda * 2)
  Prior[2] <- (sum(TrainData$is_attributed) + lambda) / (nrow(TrainData) +
lambda * 2)

  #calc IP Prob
  if (IP==1){
    IpTest <-  names(table(TestData$ip))
    IpTestNew <- setdiff(IpTest, names(IpCount))
    IpIntersect <- intersect(IpTest, names(IpCount))
    #IpProbNew <- matrix(0, nrow = length(IpTestNew), ncol = 2)
```

```
    IpProb <- matrix(0, nrow = length(IpIntersect) + length(IpTestNew), ncol
= 2)
    row.names(IpProb) <- c(IpIntersect, IpTestNew)

    tmp <- table(TrainData$ip[which(TrainData$is_attributed == 0)])
    deno <- (sum(1 - TrainData$is_attributed) + length(IpCount) * lambda)
    IpProb[,1] <- mapply(CalcProb, row.names(IpProb), deno = deno)

    tmp <- table(TrainData$ip[which(TrainData$is_attributed == 1)])
    deno <- (sum(TrainData$is_attributed) + length(IpCount) * lambda)
    IpProb[,2] <- mapply(CalcProb, row.names(IpProb), deno = deno)
  }
  #calc App prob
  if (APP==1){
    AppTest <-  names(table(TestData$app))
    AppTestNew <- setdiff(AppTest, names(AppCount))
    AppIntersect <- intersect(AppTest, names(AppCount))
    #AppProbNew <- matrix(0, nrow = length(AppTestNew), ncol = 2)
    AppProb <- matrix(0, nrow = length(AppCount) + length(AppTestNew), ncol =
2)
    row.names(AppProb) <- c(names(AppCount), AppTestNew)

    tmp <- table(TrainData$app[which(TrainData$is_attributed == 0)])
    deno <- (sum(1 - TrainData$is_attributed) + length(AppCount) * lambda)
    AppProb[,1] <- mapply(CalcProb, row.names(AppProb), deno = deno)
    tmp <- table(TrainData$app[which(TrainData$is_attributed == 1)])
    deno <- (sum(TrainData$is_attributed) + length(AppCount) * lambda)
    AppProb[,2] <- mapply(CalcProb, row.names(AppProb), deno = deno)
  }
  #calc Os prob
  if (OS==1){
    OsTest <-  names(table(TestData$os))
    OsTestNew <- setdiff(OsTest, names(OsCount))
    OsIntersect <- intersect(OsTest, names(OsCount))
    #AppProbNew <- matrix(0, nrow = length(AppTestNew), ncol = 2)
    OsProb <- matrix(0, nrow = length(OsCount) + length(OsTestNew), ncol = 2)
    row.names(OsProb) <- c(names(OsCount), OsTestNew)

    tmp <- table(TrainData$os[which(TrainData$is_attributed == 0)])
    deno <- (sum(1 - TrainData$is_attributed) + length(OsCount) * lambda)
    OsProb[,1] <- mapply(CalcProb, row.names(OsProb), deno = deno)
    tmp <- table(TrainData$os[which(TrainData$is_attributed == 1)])
    deno <- (sum(TrainData$is_attributed) + length(OsCount) * lambda)
    OsProb[,2] <- mapply(CalcProb, row.names(OsProb), deno = deno)
  }
  #calc Ch prob
  if (CH==1){
    ChTest <-  names(table(TestData$channel))
    ChTestNew <- setdiff(ChTest, names(ChCount))
```

```
    ChIntersect <- intersect(ChTest, names(ChCount))
    #ChProbNew <- matrix(0, nrow = length(ChTestNew), ncol = 2)
    ChProb <- matrix(0, nrow = length(ChCount) + length(ChTestNew), ncol = 2)
    row.names(ChProb) <- c(names(ChCount), ChTestNew)

    tmp <- table(TrainData$channel[which(TrainData$is_attributed == 0)])
    deno <- (sum(1 - TrainData$is_attributed) + length(ChCount) * lambda)
    ChProb[,1] <- mapply(CalcProb, row.names(ChProb), deno = deno)
    tmp <- table(TrainData$channel[which(TrainData$is_attributed == 1)])
    deno <- (sum(TrainData$is_attributed) + length(ChCount) * lambda)
    ChProb[,2] <- mapply(CalcProb, row.names(ChProb), deno = deno)
  }
  #calc device prob
  if (DV==1){
    DvTest <-  names(table(TestData$device))
    DvTestNew <- setdiff(DvTest, names(DvCount))
    DvIntersect <- intersect(DvTest, names(DvCount))
    #AppProbNew <- matrix(0, nrow = length(AppTestNew), ncol = 2)
    DvProb <- matrix(0, nrow = length(DvCount) + length(DvTestNew), ncol = 2)
    row.names(DvProb) <- c(names(DvCount), DvTestNew)

    tmp <- table(TrainData$device[which(TrainData$is_attributed == 0)])
    deno <- (sum(1 - TrainData$is_attributed) + length(DvCount) * lambda)
    DvProb[,1] <- mapply(CalcProb, row.names(DvProb), deno = deno)
    tmp <- table(TrainData$device[which(TrainData$is_attributed == 1)])
    deno <- (sum(TrainData$is_attributed) + length(DvCount) * lambda)
    DvProb[,2] <- mapply(CalcProb, row.names(DvProb), deno = deno)
  }
  #calc Time prob
  if (TM==1){
    TmTest <-  names(table(TestData$Hour))
    TmTestNew <- setdiff(TmTest, names(TmCount))
    TmIntersect <- intersect(TmTest, names(TmCount))
    #AppProbNew <- matrix(0, nrow = length(AppTestNew), ncol = 2)
    TmProb <- matrix(0, nrow = length(TmCount) + length(TmTestNew), ncol = 2)
    row.names(TmProb) <- c(names(TmCount), TmTestNew)

    tmp <- table(TrainData$Hour[which(TrainData$is_attributed == 0)])
    deno <- (sum(1 - TrainData$is_attributed) + length(TmCount) * lambda)
    TmProb[,1] <- mapply(CalcProb, row.names(TmProb), deno = deno)
    tmp <- table(TrainData$Hour[which(TrainData$is_attributed == 1)])
    deno <- (sum(TrainData$is_attributed) + length(TmCount) * lambda)
    TmProb[,2] <- mapply(CalcProb, row.names(TmProb), deno = deno)
  }

  TestData <- read.csv(TEST)

  TestData$Prob0 <- rep(Prior[1], nrow(TestData))
  TestData$Prob1 <- rep(Prior[2], nrow(TestData))
```

```r
  TestData$ip <- as.character(TestData$ip)
  TestData$app <- as.character(TestData$app)
  TestData$os <- as.character(TestData$os)
  TestData$device <- as.character(TestData$device)
  TestData$channel <- as.character(TestData$channel)
  TestData$Hour <- as.character(TestData$Hour)

  if (IP==1)
  {
    TestData$Prob0 <- TestData$Prob0 * IpProb[TestData$ip, 1]
    TestData$Prob1 <- TestData$Prob1 * IpProb[TestData$ip, 2]
  }
  if (APP==1)
  {
    TestData$Prob0 <- TestData$Prob0 * AppProb[TestData$app, 1]
    TestData$Prob1 <- TestData$Prob1 * AppProb[TestData$app, 2]
  }
  if (OS==1)
  {
    TestData$Prob0 <- TestData$Prob0 * OsProb[TestData$os, 1]
    TestData$Prob1 <- TestData$Prob1 * OsProb[TestData$os, 2]
  }
  if (DV==1)
  {
    TestData$Prob0 <- TestData$Prob0 * DvProb[TestData$device, 1]
    TestData$Prob1 <- TestData$Prob1 * DvProb[TestData$device, 2]
  }
  if (CH==1)
  {
    TestData$Prob0 <- TestData$Prob0 * ChProb[TestData$channel, 1]
    TestData$Prob1 <- TestData$Prob1 * ChProb[TestData$channel, 2]
  }
  if (TM==1)
  {
    TestData$Prob0 <- TestData$Prob0 * TmProb[TestData$Hour, 1]
    TestData$Prob1 <- TestData$Prob1 * TmProb[TestData$Hour, 2]
  }
  TestData$Ans <- round(TestData$Prob1 / (TestData$Prob0 + TestData$Prob1),
7)

  Ans <- TestData[,c("click_id", "Ans")]
  names(Ans)[2] <- "is_attributed"
  write.csv(Ans, file = "Ans.csv",row.names = FALSE)
}

train_decisiontree <- function(smote_train){
  set.seed(1234)
  # traindata has to be a matrix
  timestart <- Sys.time()
  # Cross Validation Preparation
```

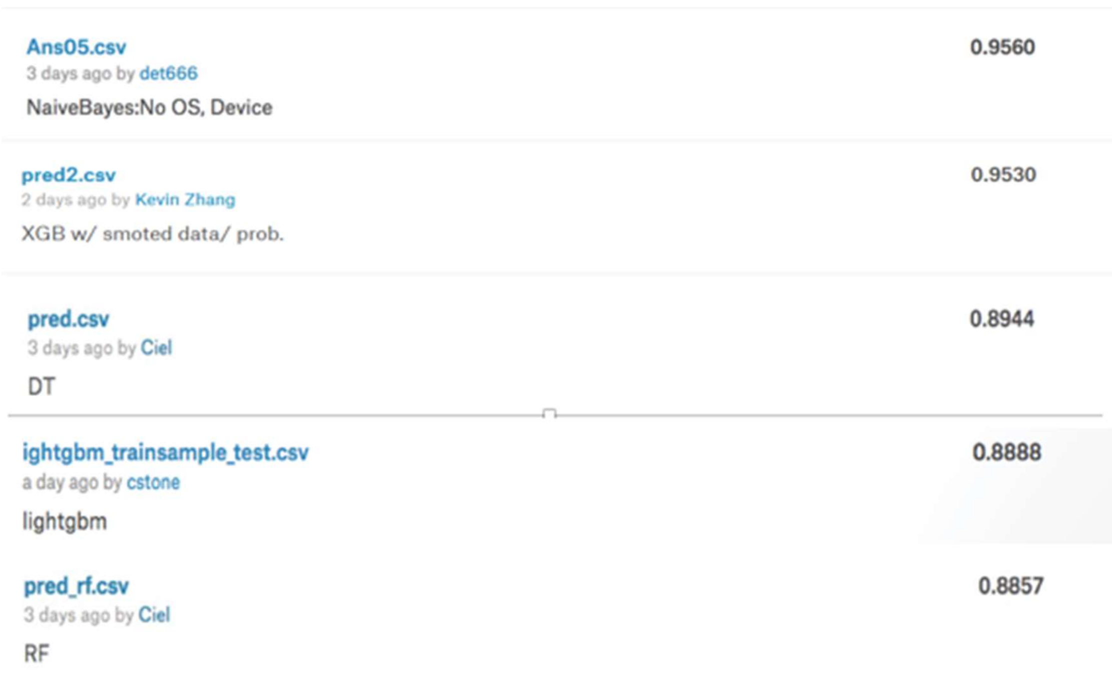
```r
  cv.3 <- createMultiFolds(smote_train$is_attributed, k = 3,
                           times = 3)
  # Control
  ctrl <- trainControl(method = "repeatedcv", number = 3,
                         repeats = 3,
                         index = cv.3)
  # Train the data
  Model_fit <- train(x = smote_train[, -4], y = smote_train[, 4],
                     method = "rpart", tuneLength = 30,
                     trControl = ctrl)
  rpart.plot(Model_CDT$finalModel, extra = 3, fallen.leaves = T)
  PRE_VDTS <- predict(Model_CDT$finalModel,
                      newdata = test_val, type ="class")
  result <- confusionMatrix(PRE_VDTS, test_val$is_attributed)
  timeend <- Sys.time()
  runningtime <- timeend - timestart
  return(list(fit = Model_fit, evaluation = result, time = runningtime))
}

train_randomforest <- function(smote_train){
  set.seed(1234)
  ind <- createDataPartition(train$is_attributed, times = 1, p = 0.8, list =
FALSE)
  train_val <- train[ind, ]
  test_val <- train[-ind, ]
  # traindata has to be a matrix
  timestart <- Sys.time()
  # Train the data
  rf_fit <- train(x = smote_train[, -4], y = smote_train[, 4],
          method = "rf", tuneLength = 3,
           ntree = 100, trControl = ctrl)
  pr.rf <- predict(rf_fit, newdata = test_val)
  result <- confusionMatrix(pr.rf, test_val$is_attributed)
  timeend <- Sys.time()
  runningtime <- timeend - timestart
  return(list(fit = rf_fit, evaluation = result, time = runningtime))
}

train_xgboost <- function(traindata,nround,cv.nfold){
  # traindata has to be a matrix
  timestart <- Sys.time()
  # Data Preparation
  xgb.train.data <- xgb.DMatrix(data = traindata[,-1],label = traindata[,1] -
1)
  # Default Parameter
  xgb_params <- list("objective" = "binary:logistic",
                     "eval_metric" = "auc",
                     "silent"="0",
                     "booster" = "gbtree")
  # Nrounds in the XgBoost
```

```
  cv_model <- xgb.cv(params      = xgb_params,
                     data        = xgb.train.data,
                     nrounds     = nround,
                     nfold       = cv.nfold,
                     verbose     = TRUE,
                     prediction  = TRUE,
                     tree_method = 'exact')

  max_auc = max(cv_model[["evaluation_log"]][, 4])
  max_auc_index = max((1:nround)[cv_model[["evaluation_log"]][, 4] ==
max_auc])

  xgb_fit <- xgb.train(data = xgb.train.data,
                       nround = max_auc_index,
                       params = xgb_params,
                       tree_method = 'exact')

  timeend <- Sys.time()
  runningtime <- timeend - timestart
  return(list(fit = xgb_fit, time = runningtime))
}


light_gbm<-function(train){
timestart <- Sys.time()
#train set and validation set
tr_index <- nrow(train)
dtrain <- train %>% head(0.95 * tr_index) # 95% data for training
valid <- train %>% tail(0.05 * tr_index) # 5% data for validation
categorical_features = c("app", "device", "os", "channel", "wday", "hour")
dtrain <- lgb.Dataset(data = as.matrix(dtrain[,colnames(dtrain) !=
"is_attributed"]),label = dtrain$is_attributed,categorical_feature =
categorical_features)
dvalid <- lgb.Dataset(data = as.matrix(valid[, colnames(valid) !=
"is_attributed"]),label = valid$is_attributed,categorical_feature =
categorical_features)
#parameter
params <- list(objective = "binary",
              metric = "auc",
              learning_rate= 0.1,
              num_leaves= 7,
              max_depth= 3,
              min_child_samples= 100,
              max_bin= 100,
              subsample= 0.7,
              subsample_freq= 1,
              colsample_bytree= 0.7,
              min_child_weight= 0,
              min_split_gain= 0,
```

```
                scale_pos_weight=99.76)
#model
model <- lgb.train(params, dtrain, valids = list(validation = dvalid),
nthread = 4,nrounds = 1000, verbose= 1, early_stopping_rounds = 50, eval_freq
= 50)
#prediction
preds <- predict(model, data = as.matrix(dvalid[, colnames(dvalid)], n =
model$best_iter))
result <- confusionMatrix(preds, dvalid$is_attributed)
timeend <- Sys.time()
#time
runningtime <- timeend - timestart
return(list(evaluation = result, time = runningtime))
}
```

Step 4: Evaluation and Model Results

We used ROC to measure the performance all the models. And Kaggle provides evaluation for the prediction of models. Here is the screenshot from Kaggle.



Screenshot from Kaggle