1.Package

```
library(e1071)
library(MASS)
library(rpart)
library(tree)
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(fastAdaboost)
library(xgboost)
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

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

```
library(stringdist)
suppressMessages(library("tidyverse"))
library(tidyverse)
library(caret)
```

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

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2019a.
## 1.0/zoneinfo/America/New_York'
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

2. Loading the Dataset

```
location <- read_csv("/Users/effyhou/Desktop/6240mining/hw4/Location.csv")
```

```
## Parsed with column specification:
## cols(
##   locationID = col_integer(),
##   regionID = col_integer()
## )
```

```
category <- read_csv("/Users/effyhou/Desktop/6240mining/hw4/Category.csv")
```

```
## Parsed with column specification:
## cols(
##   categoryID = col_integer(),
##   parentCategoryID = col_integer()
## )
```

```
train <- read_csv("/Users/effyhou/Desktop/6240mining/hw4/ItemPairs_train.csv")
```

```
## Parsed with column specification:
## cols(
##   itemID_1 = col_integer(),
##   itemID_2 = col_integer(),
##   isDuplicate = col_integer(),
##   generationMethod = col_integer()
## )
```

```
train_info <- read_csv("/Users/effyhou/Desktop/6240mining/hw4/ItemInfo_train.csv")
```

```
## Parsed with column specification:
## cols(
##   itemID = col_integer(),
##   categoryID = col_integer(),
##   title = col_character(),
##   description = col_character(),
##   images_array = col_character(),
##   attrsJSON = col_character(),
##   price = col_double(),
##   locationID = col_integer(),
##   metroID = col_double(),
##   lat = col_double(),
##   lon = col_double()
## )
```

3. Data pre-processing

```
#First, combine location and regionIDs
train_info <- train_info %>% left_join(location)
```

```
## Joining, by = "locationID"
```

```
#Second, combine test and train tables with the data in info files

#Some functions to help with the renaming later on
old_cols <- colnames(train)
is_old_column <- function(x){names(x) %in% old_cols}
check_id <- function(x,id="1"){str_sub(names(x),start = -1)==id}
name_adder <- function(x,to_add="1"){paste0(x,to_add)}

#One line dplyr call to combine tables and rename things
train <- train %>%
  left_join(train_info,by=c("itemID_1" = "itemID")) %>%
  rename_if(!is_old_column(.),name_adder,to_add="1") %>%
  left_join(train_info,by=c("itemID_2" = "itemID")) %>%
  rename_if(!is_old_column(.) & !check_id(.,id="1"),name_adder,to_add="2")
```

4. creates features

```r
# This function creates features
matchPair <- function(x, y){
  ifelse(is.na(x), ifelse(is.na(y), 3, 2), ifelse(is.na(y), 2, ifelse(x==y, 1, 4)))
}

feature_creator1 <- function(x){
  x %>%
    mutate(#distance
      distance = sqrt((lat1-lat2)^2+(lon1-lon2)^2),
      #same location
      sameLoc=matchPair(locationID1 ,locationID2),
      #same metroID
      samemetro = matchPair(metroID1,metroID2),
      #price
      sameprice=matchPair(price1,price2),
      priceDiff = pmax(price1/price2, price2/price1),
      priceMin = pmin(price1, price2, na.rm=TRUE),
      priceMax = pmax(price1, price2, na.rm=TRUE),
      #title
      titleStringDist = stringdist(title1, title2, method = "jw"),
      titleStringDist2 = (stringdist(title1, title2,
                                     method = "lcs")/pmax(nchar(title1), nchar(title2),
                                                          na.rm=TRUE)),
      titleCharDiff=pmax(nchar(title1)/nchar(title2),
                         nchar(title2)/nchar(title1)),
      titleCharMin = pmin(nchar(title1), nchar(title2), na.rm=TRUE),
      titleCharMax = pmax(nchar(title1), nchar(title2), na.rm=TRUE),
      titleMatch=matchPair(title1,title2),
      descriptionMatch=matchPair(description1,description2),
      descriptionCharDiff = pmax(nchar(description1)/nchar(description2),
                                 nchar(description2)/ nchar(description1)),
     descriptionCharMin = pmin( nchar(description1),  nchar(description2), na.rm=TRUE),
     descriptionCharMax = pmax( nchar(description1),  nchar(description2), na.rm=TRUE)

    )
}


feature_creator5 <- function(x){
  x %>%
    mutate(#distance
      distance = sqrt((lat1-lat2)^2+(lon1-lon2)^2),
      #same location
      sameLoc=matchPair(locationID1 ,locationID2),
      #same metroID
      samemetro = matchPair(metroID1,metroID2),
      #price
      sameprice=matchPair(price1,price2),
      priceDiff = pmax(price1/price2, price2/price1),
      priceMin = pmin(price1, price2, na.rm=TRUE),
      priceMax = pmax(price1, price2, na.rm=TRUE),
      #title
      titleStringDist = stringdist(title1, title2, method = "jw"),
      titleStringDist2 = (stringdist(title1, title2,
                                     method = "lcs")/pmax(nchar(title1), nchar(title2),
                                                          na.rm=TRUE)),
      titleCharDiff=pmax(nchar(title1)/nchar(title2),
                         nchar(title2)/nchar(title1)),
      titleCharMin = pmin(nchar(title1), nchar(title2), na.rm=TRUE),
      titleCharMax = pmax(nchar(title1), nchar(title2), na.rm=TRUE),
      titleMatch=matchPair(title1,title2),
      descriptionMatch=matchPair(description1,description2),
      descriptionCharDiff = pmax(nchar(description1)/nchar(description2),
                                 nchar(description2)/ nchar(description1)),
     descriptionCharMin = pmin( nchar(description1),  nchar(description2), na.rm=TRUE),
     descriptionCharMax = pmax( nchar(description1),  nchar(description2), na.rm=TRUE),

     # title-discrition distance
      title_discription_Dist_jw_1_1 = stringdist(title1, description1, method = "jw"),
      title_discription_Dist_jw_2_2 = stringdist(title2, description2, method = "jw"),
      title_discription_Dist_ja_1_1 = stringdist(title1, description1, method = "jaccard"),
      title_discription_Dist_ja_2_2 = stringdist(title2, description2, method = "jaccard"),
      title_discription_Dist_co_1_1 = stringdist(title1, description1, method = "cosine"),
      title_discription_Dist_co_2_2 = stringdist(title2, description2, method = "cosine")

    )
}

train1<- train%>% feature_creator1
train5<- train%>% feature_creator5

train1[is.na(train1)] <- -9999
train1[train1==Inf] <- -9999
train5[is.na(train5)] <- -9999
train5[train5==Inf] <- -9999
```

5. Randomly subsample and split

sample train with feature1

```
set.seed(123)
subtrain1 <- sample_frac(train1,0.03)

subtrain1 <- subtrain1  %>% mutate(isDuplicate=factor(isDuplicate))
subtrain1 <- subtrain1 %>% select(isDuplicate,distance:descriptionCharMax)

#Split the data into train, test and validation
spec1<- c(sample_train1 = 1/3, sample_test1  = 1/3, sample_valid1 = 1/3)

split1 <- sample(cut(
  seq(nrow(subtrain1)),
  nrow(subtrain1)*cumsum(c(0,spec1)),
  labels = names(spec1)
))

res1 <- split(subtrain1, split1)

sample_train1 <- res1$sample_train1
sample_test1 <-  res1$sample_test1
sample_valid1  <- res1$sample_valid1
```

sample train with feature5

```
set.seed(123)
subtrain5 <- sample_frac(train5,0.03)

subtrain5 <- subtrain5  %>% mutate(isDuplicate=factor(isDuplicate))
subtrain5 <- subtrain5 %>% select(isDuplicate,distance:descriptionCharMax)

#Split the data into train, test and validation
spec5<- c(sample_train5 = 1/3, sample_test5  = 1/3, sample_valid5= 1/3)

split5 <- sample(cut(
  seq(nrow(subtrain5)),
  nrow(subtrain5)*cumsum(c(0,spec5)),
  labels = names(spec5)
))

res5 <- split(subtrain5, split5)

sample_train5 <- res5$sample_train5
sample_test5 <-  res5$sample_test5
sample_valid5  <- res5$sample_valid5
```

6. Fit 10 different models on the training data

Based on HW4 I choose top 5 models : XGboost, RandomForest,logistic, LDA, gbm

6.1 h2o randomForest

```
#h2o.shutdown()
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## -----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
h2o.init(nthreads=-1,max_mem_size='4G')
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /var/folders/c4/q3r3mnpx3ms9jjkg6pr960hw0000gn/T//RtmpBQ9Zw3/h2o_effyhou_started_from_r.out
##     /var/folders/c4/q3r3mnpx3ms9jjkg6pr960hw0000gn/T//RtmpBQ9Zw3/h2o_effyhou_started_from_r.err
##
##
## Starting H2O JVM and connecting: .. Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 seconds 47 milliseconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.22.1.1
##     H2O cluster version age:    3 months and 19 days !!!
##     H2O cluster name:           H2O_started_from_R_effyhou_mic847
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.56 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.3.2 (2016-10-31)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (3 months and 19 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

```
sample_trainHex1<-as.h2o(sample_train1)
```

```
##
  |
  |                                                                   |   0%
  |
  |=================================================================| 100%
```

```
features1<-colnames(sample_train1)[!(colnames(sample_train1) %in% c("isDuplicate"))]
validationHex1<-as.h2o(sample_valid1)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
testHex1<-as.h2o(sample_test1)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
rf1 <- h2o.randomForest(x=features1,
                             y="isDuplicate",
                             training_frame = sample_trainHex1,
                             validation_frame = validationHex1,
                             ntree=500,
                             seed = 123)
```

```
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   1%
  |
  |=                                                                |   2%
  |
  |==                                                               |   2%
  |
  |===                                                              |   5%
  |
  |=====                                                            |   8%
  |
  |=======                                                          |  11%
  |
  |=========                                                        |  14%
  |
  |===========                                                      |  17%
  |
  |============                                                     |  19%
  |
  |==============                                                   |  22%
  |
  |================                                                 |  25%
  |
  |==================                                               |  28%
  |
  |====================                                             |  30%
  |
  |======================                                           |  34%
  |
  |========================                                         |  37%
  |
  |===========================                                      |  41%
  |
  |============================                                     |  43%
  |
  |============================                                     |  44%
  |
  |===============================                                  |  47%
  |
  |=================================                                |  51%
  |
  |===================================                              |  54%
  |
  |=====================================                            |  58%
  |
  |========================================                         |  61%
  |
  |==========================================                       |  64%
  |
  |============================================                     |  68%
  |
  |==============================================                   |  71%
  |
  |=================================================                |  75%
  |
  |===================================================              |  78%
  |
  |=====================================================            |  82%
  |
  |=======================================================          |  85%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  90%
  |
  |============================================================     |  93%
  |
  |===============================================================  |  96%
  |
  |=================================================================| 100%
```

```
#predict validation
 rf1_pred_valid<-predict(rf1  ,validationHex1, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
 rf1_pred_valid<-as.vector(rf1_pred_valid)
#predict test
 rf1_pred_test<-predict(rf1  ,testHex1, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
 rf1_pred_test<-as.vector(rf1_pred_test)


sample_trainHex5<-as.h2o(sample_train5)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
features5<-colnames(sample_train5)[!(colnames(sample_train5) %in% c("isDuplicate"))]
validationHex5<-as.h2o(sample_valid5)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
testHex5<-as.h2o(sample_test5)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
rf5<- h2o.randomForest(x=features5,
                             y="isDuplicate",
                             training_frame = sample_trainHex5,
                             validation_frame = validationHex5,
                             ntree=500,
                             seed = 123)
```

```
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   1%
  |
  |=                                                                |   2%
  |
  |==                                                               |   3%
  |
  |====                                                             |   6%
  |
|======                                                           |   9%
|
|========                                                         |  12%
|
|==========                                                       |  15%
|
|============                                                     |  19%
|
|==============                                                   |  22%
|
|================                                                 |  25%
|
|===================                                              |  29%
|
|=====================                                            |  32%
|
|======================                                           |  34%
|
|========================                                         |  37%
|
|========================                                         |  38%
|
|===========================                                      |  41%
|
|============================                                     |  42%
|
|=============================                                    |  45%
|
|===============================                                  |  48%
|
|=================================                                |  51%
|
|====================================                             |  55%
|
|======================================                           |  58%
|
|========================================                         |  61%
|
|==========================================                       |  65%
|
|============================================                     |  68%
|
|==============================================                   |  71%
|
|=================================================                |  75%
|
|===================================================              |  78%
|
|=====================================================            |  82%
|
|=======================================================          |  85%
|
|=========================================================        |  88%
|
  |============================================================     |  92%
  |
  |==============================================================   |  95%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

```
#predict validation
 rf5_pred_valid<-predict(rf5  ,validationHex5, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
 rf5_pred_valid<-as.vector(rf5_pred_valid)
#predict test
 rf5_pred_test<-predict( rf5  ,testHex5, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
 rf5_pred_test<-as.vector(rf5_pred_test)
```

6.2 xgboost

```
maxTrees <- 200
shrinkage <- 0.10
gamma <- 1
depth <- 10
minChildWeight <- 40
colSample <- 0.85
subSample <- 0.85
earlyStopRound <- 4

xg1_features<-colnames(sample_train1)[!(colnames(sample_train1) %in% c("isDuplicate"))]
d_train1 <- xgb.DMatrix(as.matrix(sample_train1[, xg1_features]), label=as.numeric(sample_train1$isDupl
d_validation1 <- sample_valid1%>%
  select(-isDuplicate) %>%
  as.matrix %>%
  xgb.DMatrix(label=as.numeric(sample_valid1$isDuplicate)-1)

test_p1<-sample_test1[,-1]
d_test1 <- xgb.DMatrix(as.matrix(test_p1))


xgb1 <- xgboost(params=list(max_depth=depth,
                            eta=shrinkage,
                            gamma=gamma,
                            colsample_bytree=colSample,
                                 min_child_weight=minChildWeight),
                     data=d_train1,
                     nrounds=90,
                     objective="binary:logistic",
                     eval_metric="auc")   #0.855005
```

```
## [1]  train-auc:0.803136
## [2]  train-auc:0.806879
## [3]  train-auc:0.809378
## [4]  train-auc:0.811136
## [5]  train-auc:0.813328
## [6]  train-auc:0.815260
## [7]  train-auc:0.816865
## [8]  train-auc:0.818603
## [9]  train-auc:0.819839
## [10] train-auc:0.820743
## [11] train-auc:0.821799
## [12] train-auc:0.822579
## [13] train-auc:0.823784
## [14] train-auc:0.824586
## [15] train-auc:0.825443
## [16] train-auc:0.826205
## [17] train-auc:0.827532
## [18] train-auc:0.828335
## [19] train-auc:0.828973
## [20] train-auc:0.829890
## [21] train-auc:0.830706
## [22] train-auc:0.831196
## [23] train-auc:0.832043
## [24] train-auc:0.832697
## [25] train-auc:0.833395
## [26] train-auc:0.833942
## [27] train-auc:0.834635
## [28] train-auc:0.835169
## [29] train-auc:0.835903
## [30] train-auc:0.836743
## [31] train-auc:0.837548
## [32] train-auc:0.837838
## [33] train-auc:0.838305
## [34] train-auc:0.838727
## [35] train-auc:0.839320
## [36] train-auc:0.839842
## [37] train-auc:0.840496
## [38] train-auc:0.841006
## [39] train-auc:0.841181
## [40] train-auc:0.841626
## [41] train-auc:0.841827
## [42] train-auc:0.842337
## [43] train-auc:0.842954
## [44] train-auc:0.843337
## [45] train-auc:0.843778
## [46] train-auc:0.844016
## [47] train-auc:0.844326
## [48] train-auc:0.844626
## [49] train-auc:0.844923
## [50] train-auc:0.845365
## [51] train-auc:0.845763
## [52] train-auc:0.846170
## [53] train-auc:0.846396
## [54] train-auc:0.846778
## [55] train-auc:0.847135
## [56] train-auc:0.847619
## [57] train-auc:0.847864
## [58] train-auc:0.848027
## [59] train-auc:0.848134
## [60] train-auc:0.848624
## [61] train-auc:0.848689
## [62] train-auc:0.848759
## [63] train-auc:0.849083
## [64] train-auc:0.849215
## [65] train-auc:0.849543
## [66] train-auc:0.849721
## [67] train-auc:0.849901
## [68] train-auc:0.850234
## [69] train-auc:0.850339
## [70] train-auc:0.850559
## [71] train-auc:0.850680
## [72] train-auc:0.851008
## [73] train-auc:0.851209
## [74] train-auc:0.851365
## [75] train-auc:0.851530
## [76] train-auc:0.851910
## [77] train-auc:0.852200
## [78] train-auc:0.852293
## [79] train-auc:0.852704
## [80] train-auc:0.853051
## [81] train-auc:0.853203
## [82] train-auc:0.853479
## [83] train-auc:0.853646
## [84] train-auc:0.853780
## [85] train-auc:0.853833
## [86] train-auc:0.853877
## [87] train-auc:0.853976
## [88] train-auc:0.854362
## [89] train-auc:0.854731
## [90] train-auc:0.855005
```

```r
#predict validation:
 xgb1_pred_valid <- predict( xgb1, d_validation1)

#predict test:
xgb1_pred_test <- predict( xgb1, d_test1)


xg5_features<-colnames(sample_train5)[!(colnames(sample_train5) %in% c("isDuplicate"))]
d_train5 <- xgb.DMatrix(as.matrix(sample_train5[, xg5_features]), label=as.numeric(sample_train5$isDupli
d_validation5 <- sample_valid5%>%
  select(-isDuplicate) %>%
  as.matrix %>%
  xgb.DMatrix(label=as.numeric(sample_valid5$isDuplicate)-1)

test_p5<-sample_test5[,-1]
d_test5 <- xgb.DMatrix(as.matrix(test_p5))


xgb5 <- xgboost(params=list(max_depth=depth,
                                 eta=shrinkage,
                                 gamma=gamma,
                                 colsample_bytree=colSample,
                                 min_child_weight=minChildWeight),
                     data=d_train5,
                     nrounds=100,
                     objective="binary:logistic",
                     eval_metric="auc")   #0.857057
```

```
## [1]  train-auc:0.803136
## [2]  train-auc:0.806879
## [3]  train-auc:0.809378
## [4]  train-auc:0.811136
## [5]  train-auc:0.813328
## [6]  train-auc:0.815260
## [7]  train-auc:0.816865
## [8]  train-auc:0.818603
## [9]  train-auc:0.819839
## [10] train-auc:0.820743
## [11] train-auc:0.821799
## [12] train-auc:0.822579
## [13] train-auc:0.823784
## [14] train-auc:0.824586
## [15] train-auc:0.825443
## [16] train-auc:0.826205
## [17] train-auc:0.827532
## [18] train-auc:0.828335
## [19] train-auc:0.828973
## [20] train-auc:0.829890
## [21] train-auc:0.830706
## [22] train-auc:0.831196
## [23] train-auc:0.832043
## [24] train-auc:0.832697
## [25] train-auc:0.833395
## [26] train-auc:0.833942
## [27] train-auc:0.834635
## [28] train-auc:0.835169
## [29] train-auc:0.835903
## [30] train-auc:0.836743
## [31] train-auc:0.837548
## [32] train-auc:0.837838
## [33] train-auc:0.838305
## [34] train-auc:0.838727
## [35] train-auc:0.839320
## [36] train-auc:0.839842
## [37] train-auc:0.840496
## [38] train-auc:0.841006
## [39] train-auc:0.841181
## [40] train-auc:0.841626
## [41] train-auc:0.841827
## [42] train-auc:0.842337
## [43] train-auc:0.842954
## [44] train-auc:0.843337
## [45] train-auc:0.843778
## [46] train-auc:0.844016
## [47] train-auc:0.844326
## [48] train-auc:0.844626
## [49] train-auc:0.844923
## [50] train-auc:0.845365
## [51] train-auc:0.845763
## [52] train-auc:0.846170
## [53] train-auc:0.846396
## [54] train-auc:0.846778
## [55] train-auc:0.847135
## [56] train-auc:0.847619
## [57] train-auc:0.847864
## [58] train-auc:0.848027
## [59] train-auc:0.848134
## [60] train-auc:0.848624
## [61] train-auc:0.848689
## [62] train-auc:0.848759
## [63] train-auc:0.849083
## [64] train-auc:0.849215
## [65] train-auc:0.849543
## [66] train-auc:0.849721
## [67] train-auc:0.849901
## [68] train-auc:0.850234
## [69] train-auc:0.850339
## [70] train-auc:0.850559
## [71] train-auc:0.850680
## [72] train-auc:0.851008
## [73] train-auc:0.851209
## [74] train-auc:0.851365
## [75] train-auc:0.851530
## [76] train-auc:0.851910
## [77] train-auc:0.852200
## [78] train-auc:0.852293
## [79] train-auc:0.852704
## [80] train-auc:0.853051
## [81] train-auc:0.853203
## [82] train-auc:0.853479
## [83] train-auc:0.853646
## [84] train-auc:0.853780
## [85] train-auc:0.853833
## [86] train-auc:0.853877
## [87] train-auc:0.853976
## [88] train-auc:0.854362
## [89] train-auc:0.854731
## [90] train-auc:0.855005
## [91] train-auc:0.855339
## [92] train-auc:0.855377
## [93] train-auc:0.855482
## [94] train-auc:0.855837
## [95] train-auc:0.856207
## [96] train-auc:0.856639
## [97] train-auc:0.856724
## [98] train-auc:0.856912
## [99] train-auc:0.856941
## [100]    train-auc:0.857057
```

```
#predict validation:
 xgb5_pred_valid <- predict( xgb5, d_validation5)

#predict test:
xgb5_pred_test <- predict( xgb5, d_test5)


xgb5_prediction<- prediction(xgb5_pred_test,labels=sample_test1$isDuplicate)

performance(xgb5_prediction,"auc")@y.values[[1]]  #  0.8178197
```

```
## [1] 0.8178197
```

6.3 LDA

```
#LDA:

lda1<- lda(isDuplicate~.,data = sample_train1)
lda1
```

```
## Call:
## lda(isDuplicate ~ ., data = sample_train1)
##
## Prior probabilities of groups:
##         0         1
## 0.5770877 0.4229123
##
## Group means:
##    distance  sameLoc samemetro sameprice   priceDiff  priceMin priceMax
## 0 0.9250093 1.188380  2.293170  3.280890 376367.8016  127854.5 20985361
## 1 2.0356004 1.248044  2.553316  2.422259   -977.3178 7301817.6 14364313
##   titleStringDist titleStringDist2 titleCharDiff titleCharMin titleCharMax
## 0       0.3212639        0.7500445       1.44943     23.34200     30.02908
## 1       0.1849947        0.4133287       1.36139     21.29207     25.96435
##   titleMatch descriptionMatch descriptionCharDiff descriptionCharMin
## 0   3.610207         3.862596            1.812630           286.8211
## 1   2.896372         3.744447            1.199938           214.5414
##   descriptionCharMax
## 0           420.1962
## 1           284.4412
##
## Coefficients of linear discriminants:
##                               LD1
## distance             1.299021e-02
## sameLoc              8.816746e-02
## samemetro            2.214140e-01
## sameprice           -4.277468e-01
## priceDiff           -2.078212e-10
## priceMin             3.609131e-11
## priceMax            -7.811239e-12
## titleStringDist      4.990981e-01
## titleStringDist2    -1.357229e+00
## titleCharDiff        6.566496e-02
## titleCharMin        -5.814435e-03
## titleCharMax        -3.338133e-03
## titleMatch          -1.373988e-01
## descriptionMatch    -1.059220e-01
## descriptionCharDiff -6.615096e-06
## descriptionCharMin  -7.360458e-05
## descriptionCharMax  -3.820759e-04
```

```
#predict validation
lda1_pred_valid<- lda1 %>%
  predict(sample_valid1) %>%
  (function(x) x$posterior[,2])
lda1_pred_valid <-as.vector(lda1_pred_valid)

#predict test
lda1_pred_test <- lda1 %>%
  predict(sample_test1) %>%
  (function(x) x$posterior[,2])
lda1_pred_test<-as.vector(lda1_pred_test)

 ####Feature 5
lda5<- lda(isDuplicate~.,data = sample_train5)
lda5
```

```
## Call:
## lda(isDuplicate ~ ., data = sample_train5)
##
## Prior probabilities of groups:
##         0         1
## 0.5770877 0.4229123
##
## Group means:
##    distance  sameLoc samemetro sameprice   priceDiff  priceMin priceMax
## 0 0.9250093 1.188380  2.293170  3.280890 376367.8016  127854.5 20985361
## 1 2.0356004 1.248044  2.553316  2.422259   -977.3178 7301817.6 14364313
##   titleStringDist titleStringDist2 titleCharDiff titleCharMin titleCharMax
## 0       0.3212639        0.7500445       1.44943     23.34200     30.02908
## 1       0.1849947        0.4133287       1.36139     21.29207     25.96435
##   titleMatch descriptionMatch descriptionCharDiff descriptionCharMin
## 0   3.610207         3.862596            1.812630           286.8211
## 1   2.896372         3.744447            1.199938           214.5414
##   descriptionCharMax
## 0           420.1962
## 1           284.4412
##
## Coefficients of linear discriminants:
```

```
##                               LD1
## distance             1.299021e-02
## sameLoc              8.816746e-02
## samemetro            2.214140e-01
## sameprice           -4.277468e-01
## priceDiff           -2.078212e-10
## priceMin             3.609131e-11
## priceMax            -7.811239e-12
## titleStringDist      4.990981e-01
## titleStringDist2    -1.357229e+00
## titleCharDiff        6.566496e-02
## titleCharMin        -5.814435e-03
## titleCharMax        -3.338133e-03
## titleMatch          -1.373988e-01
## descriptionMatch    -1.059220e-01
## descriptionCharDiff -6.615096e-06
## descriptionCharMin  -7.360458e-05
## descriptionCharMax  -3.820759e-04
```

```
#predict validation
lda5_pred_valid<- lda5 %>%
  predict(sample_valid5) %>%
  (function(x) x$posterior[,2])
lda5_pred_valid <-as.vector(lda5_pred_valid)

#predict test
lda5_pred_test <- lda5 %>%
  predict(sample_test5) %>%
  (function(x) x$posterior[,2])
lda5_pred_test<-as.vector(lda5_pred_test)
```

6.4 GBM

```
sample_trainHex1<-as.h2o(sample_train1)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
validationHex1<-as.h2o(sample_valid1)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
testHex1<-as.h2o(sample_test1)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
gbm1 <- h2o.gbm(
  ## standard model parameters
  x = features1,
  y="isDuplicate",
  training_frame = sample_trainHex1,
  validation_frame = validationHex1,
  ntrees = 500,
  learn_rate=0.07,
  sample_rate = 0.8,
  col_sample_rate = 0.6,
  seed = 1234,
  max_depth=7
)
```

```
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   1%
  |
  |==                                                               |   3%
  |
  |===                                                              |   5%
  |
  |=====                                                            |   7%
  |
  |=========                                                        |  14%
  |
  |================                                                 |  24%
  |
  |=======================                                          |  35%
  |
  |=============================                                    |  45%
  |
  |====================================                             |  55%
  |
  |===============================================                  |  72%
  |
  |=====================================================            |  81%
  |
  |=================================================================| 100%
```

```r
#predict validation
gbm_pred_valid1<-predict(gbm1 ,validationHex1, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
gbm_pred_valid1<-as.vector(gbm_pred_valid1)
#predict test
gbm_pred_test1<-predict(gbm1,testHex1, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
gbm_pred_test1<-as.vector(gbm_pred_test1)



sample_trainHex5<-as.h2o(sample_train5)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
validationHex5<-as.h2o(sample_valid5)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
testHex5<-as.h2o(sample_test5)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
gbm5 <- h2o.gbm(
  ## standard model parameters
  x = features5,
  y="isDuplicate",
  training_frame = sample_trainHex5,
  validation_frame = validationHex5,
  ntrees = 500,
  learn_rate=0.07,
  sample_rate = 0.8,
  col_sample_rate = 0.6,
  seed = 1234,
  max_depth=7
)
```

```
##
  |
  |                                                                 |   0%
  |
  |==                                                               |   3%
  |
  |====                                                             |   6%
  |
  |======                                                           |  10%
  |
  |========                                                         |  13%
  |
  |===============                                                  |  23%
  |
  |======================                                           |  34%
  |
  |=============================                                    |  45%
  |
  |====================================                             |  55%
  |
  |=========================================================        |  88%
  |
  |=================================================================| 100%
```

```
#predict validation
gbm_pred_valid5<-predict(gbm5 ,validationHex5, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
gbm_pred_valid5<-as.vector(gbm_pred_valid5)
#predict test
gbm_pred_test5<-predict(gbm5,testHex5, probability=TRUE)[3]
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
gbm_pred_test5<-as.vector(gbm_pred_test5)
```

6.5 Logistic regression

```
lg1 <- glm(isDuplicate ~ .,data=sample_train1,family="binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
lg1_pred_valid <- lg1 %>%
  predict(sample_valid1,type="response")
lg1_pred_valid<-as.vector(lg1_pred_valid)

lg1_pred_test <- lg1 %>%
  predict(sample_test1,type="response")
lg1_pred_test<-as.vector(lg1_pred_test)

lg5 <- glm(isDuplicate ~ .,data=sample_train5,family="binomial")
```

```
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
lg5_pred_valid <- lg5 %>%
  predict(sample_valid5,type="response")
lg5_pred_valid<-as.vector(lg5_pred_valid)


lg5_pred_test <- lg5 %>%
  predict(sample_test5,type="response")
lg5_pred_test<-as.vector(lg5_pred_test)
```

7. Stacking

7.1 stacking model (stacking 10 different models)

Use 10 Models:2 logistic regression, 2 xgboost, 2 gbm, 2 LDA, 2 RandomForest;

Use feature 1 and feature 5

```
stack_v<-cbind(lda1_pred_valid,lda5_pred_valid,
                gbm_pred_valid1,gbm_pred_valid5,
                lg1_pred_valid,lg5_pred_valid,
                xgb1_pred_valid,xgb5_pred_valid,
                rf1_pred_valid,rf5_pred_valid)

stack_t<-cbind(lda1_pred_test,lda5_pred_test,
                gbm_pred_test1,gbm_pred_test5,
                lg1_pred_test,lg5_pred_test,
                xgb1_pred_test,xgb5_pred_test,
                rf1_pred_test,rf5_pred_test)



stack_v_xg<- xgb.DMatrix(as.matrix(stack_v), label=as.numeric(sample_valid1$isDuplicate)-1)

modelStack<- xgboost(params=list(max_depth=depth,
                                 eta=shrinkage,
                                 gamma=gamma,
                                 colsample_bytree=colSample,
                                 min_child_weight=minChildWeight),
                     data=stack_v_xg,
                     nrounds=100,
                     objective="binary:logistic",
                     eval_metric="auc")    #0.843325
```

```
## [1]  train-auc:0.824238
## [2]  train-auc:0.826193
## [3]  train-auc:0.826974
## [4]  train-auc:0.827889
## [5]  train-auc:0.828571
## [6]  train-auc:0.829523
## [7]  train-auc:0.830061
## [8]  train-auc:0.830501
## [9]  train-auc:0.830977
## [10] train-auc:0.831344
## [11] train-auc:0.831721
## [12] train-auc:0.832048
## [13] train-auc:0.832310
## [14] train-auc:0.832662
## [15] train-auc:0.832851
## [16] train-auc:0.833139
## [17] train-auc:0.833596
## [18] train-auc:0.833935
## [19] train-auc:0.834204
## [20] train-auc:0.834476
## [21] train-auc:0.834782
## [22] train-auc:0.835042
## [23] train-auc:0.835327
## [24] train-auc:0.835538
## [25] train-auc:0.835863
## [26] train-auc:0.836041
## [27] train-auc:0.836282
## [28] train-auc:0.836601
## [29] train-auc:0.836863
## [30] train-auc:0.837138
## [31] train-auc:0.837264
## [32] train-auc:0.837545
## [33] train-auc:0.837644
## [34] train-auc:0.837820
## [35] train-auc:0.837970
## [36] train-auc:0.838293
## [37] train-auc:0.838405
## [38] train-auc:0.838578
## [39] train-auc:0.838649
## [40] train-auc:0.838882
## [41] train-auc:0.839007
## [42] train-auc:0.839161
## [43] train-auc:0.839283
## [44] train-auc:0.839496
## [45] train-auc:0.839654
## [46] train-auc:0.839755
## [47] train-auc:0.839872
## [48] train-auc:0.840046
## [49] train-auc:0.840142
## [50] train-auc:0.840184
## [51] train-auc:0.840257
## [52] train-auc:0.840441
## [53] train-auc:0.840505
## [54] train-auc:0.840634
## [55] train-auc:0.840698
## [56] train-auc:0.840748
## [57] train-auc:0.840861
## [58] train-auc:0.840905
## [59] train-auc:0.840951
## [60] train-auc:0.841007
## [61] train-auc:0.841047
## [62] train-auc:0.841103
## [63] train-auc:0.841256
## [64] train-auc:0.841336
## [65] train-auc:0.841361
## [66] train-auc:0.841387
## [67] train-auc:0.841439
## [68] train-auc:0.841690
## [69] train-auc:0.841719
## [70] train-auc:0.841825
## [71] train-auc:0.841847
## [72] train-auc:0.841875
## [73] train-auc:0.841998
## [74] train-auc:0.842034
## [75] train-auc:0.842057
## [76] train-auc:0.842085
## [77] train-auc:0.842123
## [78] train-auc:0.842148
## [79] train-auc:0.842184
## [80] train-auc:0.842212
## [81] train-auc:0.842262
## [82] train-auc:0.842294
## [83] train-auc:0.842313
## [84] train-auc:0.842379
## [85] train-auc:0.842676
## [86] train-auc:0.842692
## [87] train-auc:0.842714
## [88] train-auc:0.842786
## [89] train-auc:0.842856
## [90] train-auc:0.842908
## [91] train-auc:0.842930
## [92] train-auc:0.842969
## [93] train-auc:0.843163
## [94] train-auc:0.843181
## [95] train-auc:0.843212
## [96] train-auc:0.843240
## [97] train-auc:0.843260
## [98] train-auc:0.843275
## [99] train-auc:0.843300
## [100]    train-auc:0.843325
```

```r
modelStack_predict<- predict(modelStack,stack_t)


#AUC
stack_prediction<- prediction(modelStack_predict,labels=sample_test1$isDuplicate)

performance(stack_prediction,"auc")@y.values[[1]]  # 0.819282
```

```
## [1] 0.819282
```

Stacking model has higher AUC (0.819282) than the single XGboost model (which has the highest AUC score (0.8178197) among 10 different models).

7.2 Use Stacking model to obtain classifications

```
class_stack<-ifelse (modelStack_predict > 0.5,1,0)
```