R Notebook

1.Data importation and manipulation

First, we read in all datasets given, both traning and testing.

```r
#read in data
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.1
```

```
## -- Attaching packages ---------------------------------------------------------
------------ tidyverse 1.2.1 --
```

```
## √ ggplot2 3.0.0     √ purrr   0.2.5
## √ tibble  1.4.2     √ dplyr   0.7.8
## √ tidyr   0.8.2     √ stringr 1.3.1
## √ readr   1.1.1     √ forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

```
## Warning: package 'tibble' was built under R version 3.5.1
```

```
## Warning: package 'tidyr' was built under R version 3.5.1
```

```
## Warning: package 'readr' was built under R version 3.5.1
```

```
## Warning: package 'purrr' was built under R version 3.5.1
```

```
## Warning: package 'dplyr' was built under R version 3.5.1
```

```
## Warning: package 'stringr' was built under R version 3.5.1
```

```
## Warning: package 'forcats' was built under R version 3.5.1
```

```
## -- Conflicts ------------------------------------------------------------------
------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(readr)

location <- read_csv("C:/Users/zhang/Desktop/6240_r/hw4/Location.csv")
```

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

```r
#pair comparison see if duplicate
itemPairsTest <- read_csv("C:/Users/zhang/Desktop/6240_r/hw4/ItemPairs_test.csv")
```

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

```r
itemPairsTrain <- read_csv("C:/Users/zhang/Desktop/6240_r/hw4/ItemPairs_train.csv")
```

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

```r
#info
itemInfoTest <- read_csv("C:/Users/zhang/Desktop/6240_r/hw4/ItemInfo_test.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()
## )
```

```
itemInfoTrain <- read_csv("C:/Users/zhang/Desktop/6240_r/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()
## )
```

Log of price is not necessarily useful for the model.

```
#trail:when id is either id1 or id2, present the data
#id1 <- itemPairsTrain[6,1]
#id2 <- itemPairsTrain[6,2]
#itemInfoTrain %>% filter(itemID %in% c(id1,id2))

#change price into log form and get the density plot for log price to see if the dis
tribution is normal.
#itemInfoTrain %>%
#  mutate(logprice=log(price)) %>%
#  ggplot + geom_density(aes(x=logprice))
```

After reading in the data, we want to combine the two traning datasets itemPairsTrain and itemInfoTrain according to the item-ids these ads are assigned to. Also, we merge the location data into this dataset as well. So we will have all the information for the two products in the same row.

```
#Join training data#

#maerge location to the item info training and testing datasets
str(itemInfoTrain)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    3344613 obs. of  11 variables:
##  $ itemID      : int  1 3 4 7 8 9 12 15 16 19 ...
##  $ categoryID  : int  81 14 84 84 39 39 9 32 27 88 ...
##  $ title       : chr  "Продам Камаз 6520" "Yamaha r6" "iPhone 3gs 8gb" "Xiaomi Mi
4 3гб RAM + 16гб ROM белый" ...
##  $ description : chr  "Продам Камаз 6520 20 тонн" "Весь в тюнинге." "Телефон в хо
рошем состоянии, трещин и сколов нет, за все время менялся только аккумулятор(поэтом
у заряд держит "| __truncated__ "Отличный подарок на новый год от \"китайской apple
\"\nНовый в упаковке. Коробку вместе вскроем)\nЭкран: 5 дюймо"| __truncated__ ...
##  $ images_array: chr  "1064094, 5252822, 6645873, 6960145, 9230265" "11919573, 14
412228, 3204180, 6646877" "14384831, 6102021" NA ...
##  $ attrsJSON   : chr  "{\"Вид техники\":\"Грузовики\"}" "{\"Вид техники\":\"Мотоц
иклы\", \"Вид мотоцикла\":\"Спортивные\"}" "{\"Вид телефона\":\"iPhone\"}" "{\"Вид т
елефона\":\"Другие марки\"}" ...
##  $ price       : num  300000 300000 3500 13500 500 7000 445000 1600 1000 5000 ...
##  $ locationID  : int  648140 639040 640650 662210 624360 644200 631060 662810 657
600 637640 ...
##  $ metroID     : num  NA NA NA NA NA ...
##  $ lat         : num  64.7 55.7 56.2 55.8 55.8 ...
##  $ lon         : num  30.8 37.3 43.5 37.6 37.6 ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 11
##   .. ..$ itemID      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ categoryID  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ title       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ description : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ images_array: list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ attrsJSON   : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ price       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ locationID  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ metroID     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ lat         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ lon         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

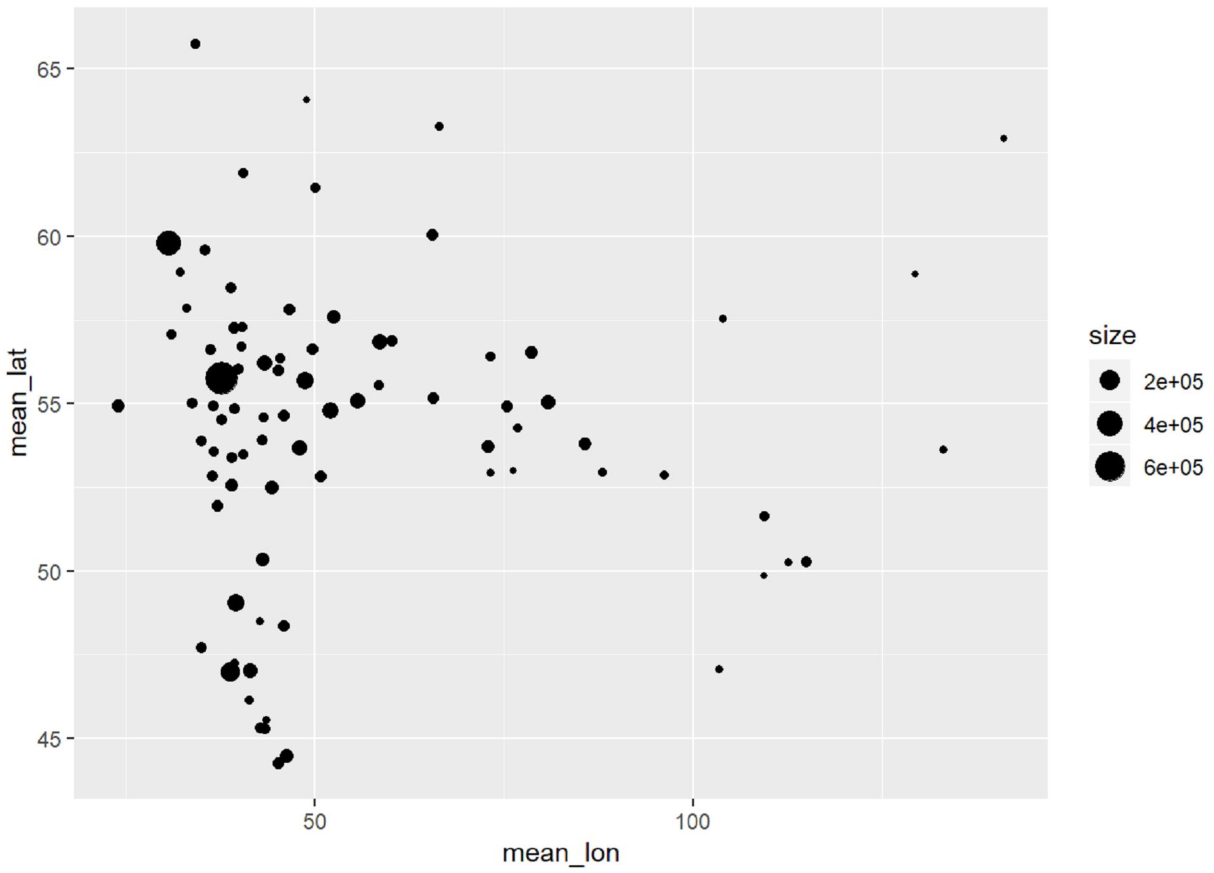
```
itemInfoTrain <- itemInfoTrain %>%
  left_join(location)
```

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

```
itemInfoTest <- itemInfoTest %>%
  left_join(location)
```

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

```
itemInfoTrain %>% group_by(regionID) %>%
  summarise(mean_lat=mean(lat),
            mean_lon=mean(lon),
            size = n()
  ) %>%
  ggplot(aes(y=mean_lat,x=mean_lon,size=size))+
  geom_point()
```



```
##Join training data##

train <- itemPairsTrain %>% left_join(itemInfoTrain,
                                       by = c("itemID_1" = "itemID"))
colnames(train)[5:15] <- paste0(colnames(train)[5:15],"_1")

train <- train %>% left_join(itemInfoTrain,
                             by = c("itemID_2" = "itemID"))
colnames(train)[16:26] <- paste0(colnames(train)[16:26],"_2")



##Join testing data##

test <- itemPairsTest %>% left_join(itemInfoTest,
                                     by = c("itemID_1" = "itemID"))
colnames(test)[4:14] <- paste0(colnames(test)[4:14],"_1")

test <- test %>% left_join(itemInfoTest,
                           by = c("itemID_2" = "itemID"))
colnames(test)[15:25] <- paste0(colnames(test)[15:25],"_2")
```

The metro location data has too much missing data (nearly 60% of the data is missing) thus it cannot be a good predictor for the model.

```
#see how much missing data are there
metroID_1.total <- length(train$metroID_1)
metroID_2.total <- length(train$metroID_2)
metroID_1.missing <- sum(is.na(train$metroID_1))
metroID_2.missing <- sum(is.na(train$metroID_2))

metroID_1.missing/metroID_1.total#metroID_1.missing.percentage
```

```
## [1] 0.6604839
```

```
metroID_2.missing/metroID_2.total##metroID_2.missing.percentage
```

```
## [1] 0.6606187
```

Since the training dataset is too large for us to run, we randomly chose 10% of the dataset to build the model.

```
set.seed(1)
train_val_id <- sample(nrow(train),
                       size = floor(nrow(train)*.10),replace = FALSE)
train_val <- train[train_val_id,]
train <- train[-train_val_id,]
```

Since similar product should have similar prices, the price different between item1 and item2 should be critical for identification of duplicity. Therefore, add a new variable to both training and testing datasets. The variable should equal to 1 if the result of the subtraction is small 0 for otherwise. For regionID, categoryID, and title, create new categorical variable. If two items have the same value for these variables, the result is 1, 0 otherwise.

```
#make adjustement to the datasets
quasi_creator_1 <- function(x){
  x %>% mutate(
         distance = sqrt((lat_1-lat_2)^2+(lon_1-lon_2)^2),
         price_min=pmin(log(price_1),log(price_2)),
         price_max=pmax(log(price_1),log(price_2)),
         price.diff=abs(price_1-price_2),
         price.diffpct=1*(abs(price_1-price_2)/pmin(price_1,price_2)<0.20), #if the
difference in price is higher than 15% of the cheaper item, the price is assumed to
be significantly different
         description_same=1*(description_1==description_2),
         location_same=1*(locationID_1==locationID_2),
         region_same=1*(regionID_1==regionID_2),
         category_same=1*(categoryID_1==categoryID_2),
         title_same=1*(title_1==title_2))
}

#change to the datasets using the function created above
train <- train %>% quasi_creator_1
```

```
## Warning: package 'bindrcpp' was built under R version 3.5.1
```

```
train_val <- train_val%>% quasi_creator_1
test <- test %>% quasi_creator_1
```

Looking at the new variables created, see how much of the data is NA. Since in all cases the percentages of missing data are not high, we can still use these variables. In addition, we can use mean imputation for these missing data.

```
#see the percentages of data missing
sum(is.na(train$price.diffpct))/length(train$price.diffpct) #0.1133116
```

```
## [1] 0.1133116
```

```
sum(is.na(train$region_same))/length(train$region_same) #0
```

```
## [1] 0
```

```
sum(is.na(train$title_same))/length(train$title_same)#3.714356e-07
```

```
## [1] 3.714356e-07
```

```
sum(is.na(train$description_same))/length(train$description_same)#.528638e-05
```

```
## [1] 3.528638e-05
```

```
sum(is.na(train$location_same))/length(train$location_same)#0
```

```
## [1] 0
```

```
#use imputation to fill up the missing spots for train
train$price_min <- ifelse(is.na(train$price_min), mean(train$price_min, na.rm=TRU
E), train$price_min)

train$price_max <- ifelse(is.na(train$price_max), mean(train$price_max, na.rm=TRU
E), train$price_max)

train$distance <- ifelse(is.na(train$distance), mean(train$distance, na.rm=TRUE), tr
ain$distance)

train$price.diffpct <- ifelse(is.na(train$price.diffpct), mean(train$price.diffpct,
na.rm=TRUE), train$price.diffpct)

train$region_same <- ifelse(is.na(train$region_same), mean(train$region_same, na.rm=
TRUE), train$region_same)

train$title_same <- ifelse(is.na(train$title_same), mean(train$title_same, na.rm=TRU
E), train$title_same)

train$description_same <- ifelse(is.na(train$description_same), mean(train$descripti
on_same, na.rm=TRUE), train$description_same)

train$location_same <- ifelse(is.na(train$location_same), mean(train$location_same,
na.rm=TRUE), train$location_same)

train$price_max <- ifelse(is.na(train$price_max), mean(train$price_max, na.rm=TRU
E), train$price_max)

train$price_min <- ifelse(is.na(train$price_min), mean(train$price_min, na.rm=TRU
E), train$price_min)


#train_val
train_val$price_min <- ifelse(is.na(train_val$price_min), mean(train_val$price_min,
na.rm=TRUE), train_val$price_min)

train_val$price_max <- ifelse(is.na(train_val$price_max), mean(train_val$price_max,
na.rm=TRUE), train_val$price_max)

train_val$distance <- ifelse(is.na(train_val$distance), mean(train_val$distance, na.
rm=TRUE), train_val$distance)

train_val$price.diffpct <- ifelse(is.na(train_val$price.diffpct), mean(train_val$pri
ce.diffpct, na.rm=TRUE), train_val$price.diffpct)

train_val$region_same <- ifelse(is.na(train_val$region_same), mean(train_val$region_
same, na.rm=TRUE), train_val$region_same)

train_val$title_same <- ifelse(is.na(train_val$title_same), mean(train_val$title_sam
e, na.rm=TRUE), train_val$title_same)

train_val$description_same <- ifelse(is.na(train_val$description_same), mean(train_v
```

```r
al$description_same, na.rm=TRUE), train_val$description_same)

train_val$location_same <- ifelse(is.na(train_val$location_same), mean(train_val$loc
ation_same, na.rm=TRUE), train_val$location_same)

train_val$price_max <- ifelse(is.na(train_val$price_max), mean(train_val$price_max,
na.rm=TRUE), train_val$price_max)

train_val$price_min <- ifelse(is.na(train_val$price_min), mean(train_val$price_min,
na.rm=TRUE), train_val$price_min)

#test
test$price_min <- ifelse(is.na(test$price_min), mean(test$price_min, na.rm=TRUE), te
st$price_min)

test$price_max <- ifelse(is.na(test$price_max), mean(test$price_max, na.rm=TRUE), te
st$price_max)

test$distance <- ifelse(is.na(test$distance), mean(test$distance, na.rm=TRUE), test
$distance)

test$price.diffpct <- ifelse(is.na(test$price.diffpct), mean(test$price.diffpct, na.
rm=TRUE), test$price.diffpct)

test$region_same <- ifelse(is.na(test$region_same), mean(test$region_same, na.rm=TRU
E), test$region_same)

test$title_same <- ifelse(is.na(test$title_same), mean(test$title_same, na.rm=TRU
E), test$title_same)

test$description_same <- ifelse(is.na(test$description_same), mean(test$description_
same, na.rm=TRUE), test$description_same)

test$location_same <- ifelse(is.na(test$location_same), mean(test$location_same, na.
rm=TRUE), test$location_same)

test$price_max <- ifelse(is.na(test$price_max), mean(test$price_max, na.rm=TRUE), te
st$price_max)

test$price_min <- ifelse(is.na(test$price_min), mean(test$price_min, na.rm=TRUE), te
st$price_min)
```

2. Models

Try out different classifiers and choose the one with highest AUC. Improve the model by variable transformation. Finally, choose the model with the hightest AUC score. Here, LDA, QDA, Logistic regression, SVM, Decision Trees, Random Forests, various forms of Boosting are used.

By looking at the results of this model, the AUC score is 0.6411356 Since a random guess would generate an AUC of 0.5, the AUC score here is not amazing for LDA.

```
#Linear discriminent analysis#
#fit the LDA model based on train
# Classification: ROC Curve and AUC: https://developers.google.com/machine-learning/
crash-course/classification/roc-and-auc
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.5.1
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda.fit<-lda(isDuplicate~price_min+price_max+distance+location_same+description_same
+region_same+title_same+price.diffpct,data=train)


lda.fit
```

```
## Call:
## lda(isDuplicate ~ price_min + price_max + distance + location_same +
##     description_same + region_same + title_same + price.diffpct,
##     data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.5773561 0.4226439
##
## Group means:
##   price_min price_max  distance location_same description_same region_same
## 0  7.902219  8.433014 0.8356506     0.9398890       0.04435732   0.9721216
## 1  8.612074  8.803250 2.0282860     0.9134617       0.08721441   0.9838223
##   title_same price.diffpct
## 0  0.1304204     0.4575571
## 1  0.3666108     0.7772682
##
## Coefficients of linear discriminants:
##                          LD1
## price_min         0.16782858
## price_max        -0.15472525
## distance          0.02702872
## location_same    -0.78095918
## description_same  0.52046890
## region_same       1.21679918
## title_same        1.24193944
## price.diffpct     1.47138860
```

```
lda.class <- predict(lda.fit)$class #give prediction
table(train$isDuplicate,lda.class) #compare with the real value using confusion matr
ix
```

```
##    lda.class
##           0       1
##   0 1328958  225433
##   1  648253  489613
```

```
#AUC
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.5.2
```

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

```
## Warning: package 'gplots' was built under R version 3.5.1
```

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

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

```
#get the prediction value
lda.predictions <-predict(lda.fit,train_val)$class
#turn the prediction values into data frame
lda.pred <- as.data.frame(sapply(lda.predictions, as.numeric))
#isDuplicate of train_val
real <-  as.data.frame(sapply(train_val$isDuplicate, as.numeric))
#create prediction and real set
lda.pred <- prediction(lda.pred,labels=real)#label means the true value,test_val is
the other 90% of the training data

#calculate AUC value
performance(lda.pred,"auc")@y.values[[1]] #AUC value for train_val
```

```
## [1] 0.6411356
```

```
#ROC (receiver operating characteristic curve) plot

#performance(lda.pred,"tpr","fpr") #ROC
#plot(lda.ROC)#ROC
#A perfect predictor gives an AUC-ROC score of 1, a predictor which makes random gue
sses has an AUC-ROC score of 0.5.
```

QDA generates AUC score of 0.6363608 which is even worse than LDA.

```
#Quadratic discriminent analysis#
qda.fit <- qda(isDuplicate~price_min+price_max+distance+location_same+description_sa
me+region_same+title_same+price.diffpct,data=train)

qda.fit
```

```
## Call:
## qda(isDuplicate ~ price_min + price_max + distance + location_same +
##     description_same + region_same + title_same + price.diffpct,
##     data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.5773561 0.4226439
##
## Group means:
##   price_min price_max  distance location_same description_same region_same
## 0  7.902219  8.433014 0.8356506     0.9398890       0.04435732   0.9721216
## 1  8.612074  8.803250 2.0282860     0.9134617       0.08721441   0.9838223
##   title_same price.diffpct
## 0  0.1304204     0.4575571
## 1  0.3666108     0.7772682
```

```
qda.predictions <-predict(qda.fit,train_val)$class

qda.pred <- as.data.frame(sapply(qda.predictions, as.numeric))

real <-  as.data.frame(sapply(train_val$isDuplicate, as.numeric))

qda.pred <- prediction(qda.pred,labels=real)

performance(qda.pred,"auc")@y.values[[1]] #AUC
```

```
## [1] 0.6363608
```

```
#performance(qda.pred,"tpr","fpr") #ROC
```

The AUC score for logistic regression is 0.7455485 which is higher than those of QDA and LDA.

```
#Logistic regression#
log.fit <- glm(isDuplicate ~price_min+price_max+distance+ location_same+description_
same+region_same+category_same+title_same+price.diffpct,data=train,family="binomia
l")

log.fit %>% summary
```

```
##
## Call:
## glm(formula = isDuplicate ~ price_min + price_max + distance +
##     location_same + description_same + region_same + category_same +
##     title_same + price.diffpct, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.0423  -1.0294  -0.6063   1.0644   3.6958
##
## Coefficients: (1 not defined because of singularities)
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -1.8324094  0.0116105 -157.82   <2e-16 ***
## price_min         0.2679170  0.0033545   79.87   <2e-16 ***
## price_max        -0.2567920  0.0033596  -76.44   <2e-16 ***
## distance          0.0269050  0.0002494  107.90   <2e-16 ***
## location_same    -0.6981399  0.0063768 -109.48   <2e-16 ***
## description_same  0.4416040  0.0054920   80.41   <2e-16 ***
## region_same       1.1286448  0.0116061   97.25   <2e-16 ***
## category_same            NA         NA      NA       NA
## title_same        1.0251519  0.0033215  308.64   <2e-16 ***
## price.diffpct     1.1999500  0.0039588  303.11   <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: 3667560  on 2692256  degrees of freedom
## Residual deviance: 3179444  on 2692248  degrees of freedom
## AIC: 3179462
##
## Number of Fisher Scoring iterations: 4
```

```
log.predictions<-predict(log.fit, newdata = train_val,type = "response")#Obtains pre
dictions from a fitted generalized linear model.
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
log.pred <- as.data.frame(sapply(log.predictions, as.numeric))

real <-  as.data.frame(sapply(train_val$isDuplicate, as.numeric))

log.predict <- prediction(log.pred,labels=real)

performance(log.predict,"auc")@y.values[[1]]
```

```
## [1] 0.7455485
```

I am not able to predict because the prediction by the “predict” function produce missing output. Mnay of the predictions are not generated. They are not NA, intead they are just missing. In other words, the number of predictions is not equal to the number if isDuplicate in the train_val. Thus, AUC score cannot be checked.

```
#install.packages("e1071")
#library(e1071)
#Support Vector Machine#
#train1 <- sample_frac(train,0.001)
#train1 <- data.frame(train1)
#train1$isDuplicate=as.factor(train1$isDuplicate)
#svm linear fit
#svm.fit1 <- svm(isDuplicate~price_min+price_max+distance+location_same+description_same+region_same+title_same+price.diffpct,data=train1,scale = TRUE,        method="C-classifcation",cost=10,kernel="linear")

#svm.fit1.pred <- svm.fit1 %>%
 # predict(train_val) %>%
 # prediction(labels=train_val$isDuplicate)

#performance(svm.fit1.pred,"auc")@y.values[[1]]
```

Decision Tree without bagging gives AUC score of 0.6134513 which is close to the reults of QDA and LDA.

```
#Decision Trees#
library (tree)
```

```
## Warning: package 'tree' was built under R version 3.5.2
```

```
library (ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.5.1
```

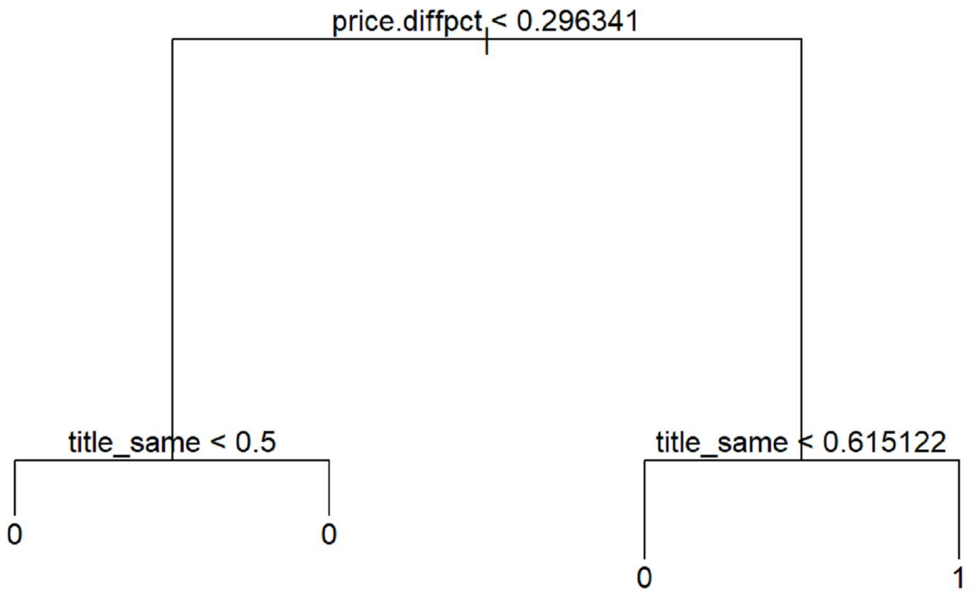
```
train.tree <- data.frame(train)
train.tree$isDuplicate <- as.factor(train.tree$isDuplicate)

tree.fit= tree( isDuplicate~price_min+price_max+distance+location_same+description_s
ame+region_same+title_same+price.diffpct,data=train.tree )

summary(tree.fit)
```

```
##
## Classification tree:
## tree(formula = isDuplicate ~ price_min + price_max + distance +
##     location_same + description_same + region_same + title_same +
##     price.diffpct, data = train.tree)
## Variables actually used in tree construction:
## [1] "price.diffpct" "title_same"
## Number of terminal nodes:  4
## Residual mean deviance:  1.191 = 3207000 / 2692000
## Misclassification error rate: 0.3412 = 918621 / 2692257
```

```
plot(tree.fit);text(tree.fit ,pretty =0)
```



```
tree.prediction<-predict(tree.fit,train_val)


modify <- function(x){
  x %>% mutate(
        prediction = 1*(tree.prediction[,1]<tree.prediction[,2]))
}

tree.prediction<- data.frame(tree.prediction)
tree.prediction<- tree.prediction %>% modify

tree.pred <- prediction(tree.prediction$prediction,labels=train_val$isDuplicate)
performance(tree.pred,"auc")@y.values[[1]]
```

```
## [1] 0.6134513
```

The AUC score for Random Forest is 0.6610911

```
#Random Forests#
#install.packages("randomForest")
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.3
```

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

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
randomforest.train <- sample_frac(train,0.001)
randomforest.train$isDuplicate <- as.factor(randomforest.train$isDuplicate)
randomforest.train<-data.frame(randomforest.train)

randomForest.fit <- randomForest(isDuplicate~price_min+price_max+distance+location_s
ame+description_same+region_same+title_same+price.diffpct,data=randomforest.train, i
mportance = TRUE, mtry=5,ntree=800)

randomForest.fit
```

```
##
## Call:
##  randomForest(formula = isDuplicate ~ price_min + price_max +      distance + loc
ation_same + description_same + region_same +      title_same + price.diffpct, data
= randomforest.train, importance = TRUE,      mtry = 5, ntree = 800)
##                Type of random forest: classification
##                      Number of trees: 800
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 33.73%
## Confusion matrix:
##      0   1 class.error
## 0 1177 385   0.2464789
## 1  523 607   0.4628319
```

```
randomForest.predictions<-predict(randomForest.fit, newdata = train_val)

randomForest.pred <- as.data.frame(sapply(randomForest.predictions, as.numeric))

real <-  as.data.frame(sapply(train_val$isDuplicate, as.numeric))

randomForest.predict <- prediction(randomForest.pred,labels=real)

performance(randomForest.predict,"auc")@y.values[[1]]
```

```
## [1] 0.6501238
```

The AUC score for Gradient Boosting Machines is 0.7253959

```
#install.packages("gbm")
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.5.3
```

```
## Loaded gbm 2.1.5
```

```
train1 <- sample_frac(train,0.001)

gbm.fit <-gbm(isDuplicate~price_min+price_max+distance+location_same+description_sam
e+region_same+title_same+price.diffpct,data=train1,
                distribution = "bernoulli",
                n.trees = 800,
                interaction.depth = 10)

gbm.predictions<-predict(gbm.fit, n.trees=800, newdata = train_val)

gbm.pred <- as.data.frame(sapply(gbm.predictions, as.numeric))

real <-  as.data.frame(sapply(train_val$isDuplicate, as.numeric))

gbm.predict <- prediction(gbm.pred,labels=real)


performance(gbm.predict,"auc")@y.values[[1]]
```

```
## [1] 0.7259613
```

The AUC score for adaboost0.658594

```
#install.packages('fastAdaboost')
library(fastAdaboost)
```

```
## Warning: package 'fastAdaboost' was built under R version 3.5.3
```

```
trainadaboost<-data.frame(train1)

adaboost.fit <- adaboost(isDuplicate~price_min+price_max+distance+location_same+desc
ription_same+region_same+title_same+price.diffpct,data=trainadaboost,8)

adaboost.predictions<-predict(adaboost.fit,newdata = train_val)

adaboost.pred <- adaboost.predictions[2]
adaboost.pred<- as.data.frame(adaboost.pred)
a<-adaboost.pred[,1]
b<-adaboost.pred[,2]

#for (i in 1:length(a)){
#  prediction[i] = 1*(a[i]<b[i])
#}
#real <-  as.data.frame(sapply(train_val$isDuplicate, as.numeric))

#adaboost.predict <- prediction(prediction,labels=real)

#performance(adaboost.predict,"auc")@y.values[[1]]
```

Therefore, logistic regression generates the highest AUC score which is 0.7455485. Thus, we use logistic regression to predict using the test data.

```
log.predictions.test<-predict(log.fit, newdata = test,type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
#write.csv(log.predictions.test, file = "MyData.csv")
```