HW3-Fangzhou Song

Fangzhou Song

Package loading

```
rm(list=ls())
library(tidyverse)
```

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

## v ggplot2 3.1.0     v purrr   0.3.0
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0

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

```
library(MASS)
```

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

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

```
library(e1071)
library(ROCR)
```

```
## Loading required package: gplots

##
## Attaching package: 'gplots'

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

```
library(gplots)
library(tree)
backup_options=options()
options(scipen=200)
set.seed(233)
```

Q1. Clickthrough Rate Analysis

1. Get the dataset and create a new dataset with all of the search events (**searchResultPage**) in the original dataset. The new dataset should also include **ALL** of the variables for these **searchResultPage** items.

2. To your new dataset, add a new variable called **clickthrough** which should be **TRUE** if the user clicked on a link immediately after this search.

Read data

```
data_origin=read_csv("C:/Users/ArkSong/Desktop/GWU/Stat 6240-Statistical Data Mining/Assignments/HW3/eve
```

Process data

```
(data1=data_origin %>%
  filter(action=="searchResultPage"|action=="visitPage") %>%
  arrange(session_id,timestamp) %>%
  group_by(session_id) %>%
  mutate(
    action_lead=lead(action)
  ) %>%
  filter(action=="searchResultPage") %>%
  mutate(
    clickthrough=if_else(action_lead!="visitPage" | is.na(action_lead)==TRUE,"FALSE","TRUE")
  ) %>%
  mutate(
    hour=substr(timestamp,9,10),
    min=substr(timestamp,11,12)
  ) %>%
    ungroup() %>%
   dplyr::select(clickthrough,group,n_results,hour,min) %>%
   mutate(
     clickthrough=as.factor(clickthrough),
     group=as.factor(group),
     hour=as.integer(hour),
     min=as.integer(min)
   )
 )
```

```
## # A tibble: 136,234 x 5
##    clickthrough group n_results  hour   min
##    <fct>        <fct>     <dbl> <int> <int>
##  1 FALSE        b            20    15    20
##  2 TRUE         b            18     8    49
##  3 TRUE         b            20     9    24
##  4 FALSE        a            20    16    19
##  5 FALSE        a            20    16    19
##  6 FALSE        a            20    16    20
##  7 FALSE        a            20    16    20
##  8 FALSE        a            20    16    20
##  9 FALSE        a            20    16    20
## 10 FALSE        b             1     5    33
## # ... with 136,224 more rows
```

Here, I include events "searchResultPage" and "visitPage" firstly and arrange all data in *session_id* and *timestamp* order. Then, in terms of each session, I apply lead function to create variable *action_lead*. The "searchResultPage" record whose *action_lead* is "visitPage" is the one that followed by "visitPage" sequentially, which means user clicked on a link immediately after **this** search. Therefore, those records shoulbe regard as "TRUE" in *clickthrough*

4. Randomly sample 10% of your dataset and store it as a new dataset. This will be your training data. Store the other 90% of the data in a separate data frame, and this will be the testing data.

```
train_index=sample(dim(data1)[1],round(dim(data1)[1]*0.1))
data1_train=data1[train_index,]
data1_test=data1[-(train_index),]
```

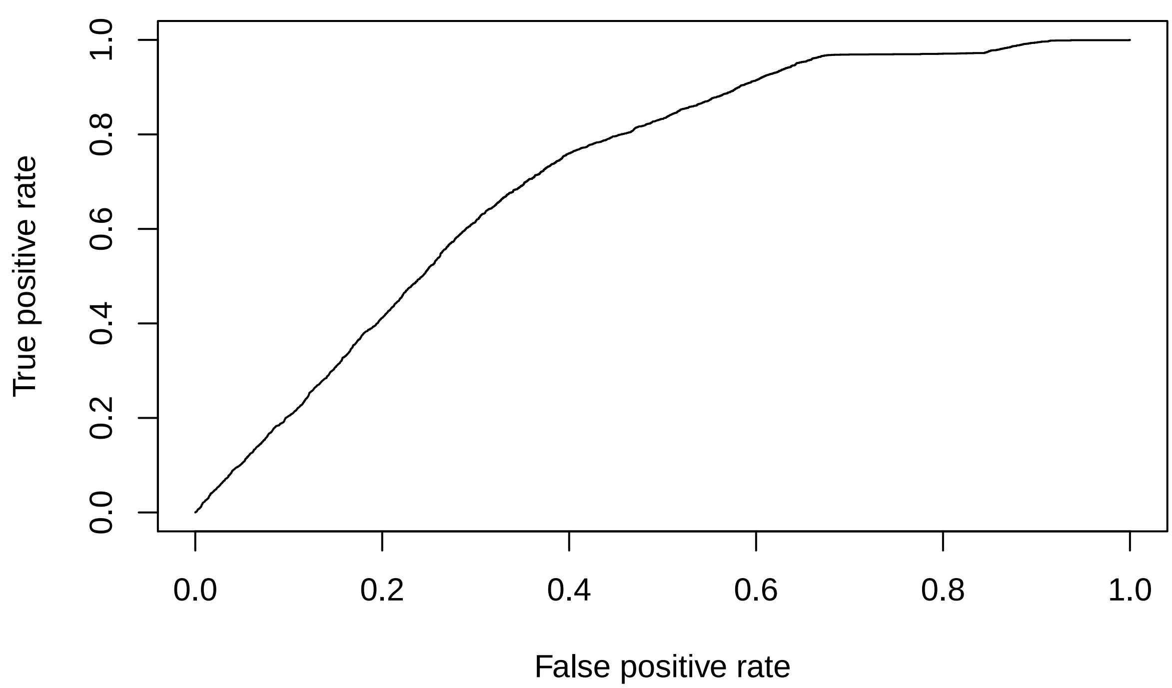
Naive Bayes

```
options(backup_options)
nb.fit1=naiveBayes(x=data1_train[,-1],y=data1_train$clickthrough)

nb.pred1_train=prediction(predict(nb.fit1,newdata=data1_train,type="raw")[,2],
                          data1_train$clickthrough)

nb.pred1_test=prediction(predict(nb.fit1,newdata=data1_test,type="raw")[,2],
                         data1_test$clickthrough)
```

Training set

resulting ROC curves

```
nb.ROC1_train=performance(nb.pred1_train,"tpr","fpr")
plot(nb.ROC1_train)
```



AUC

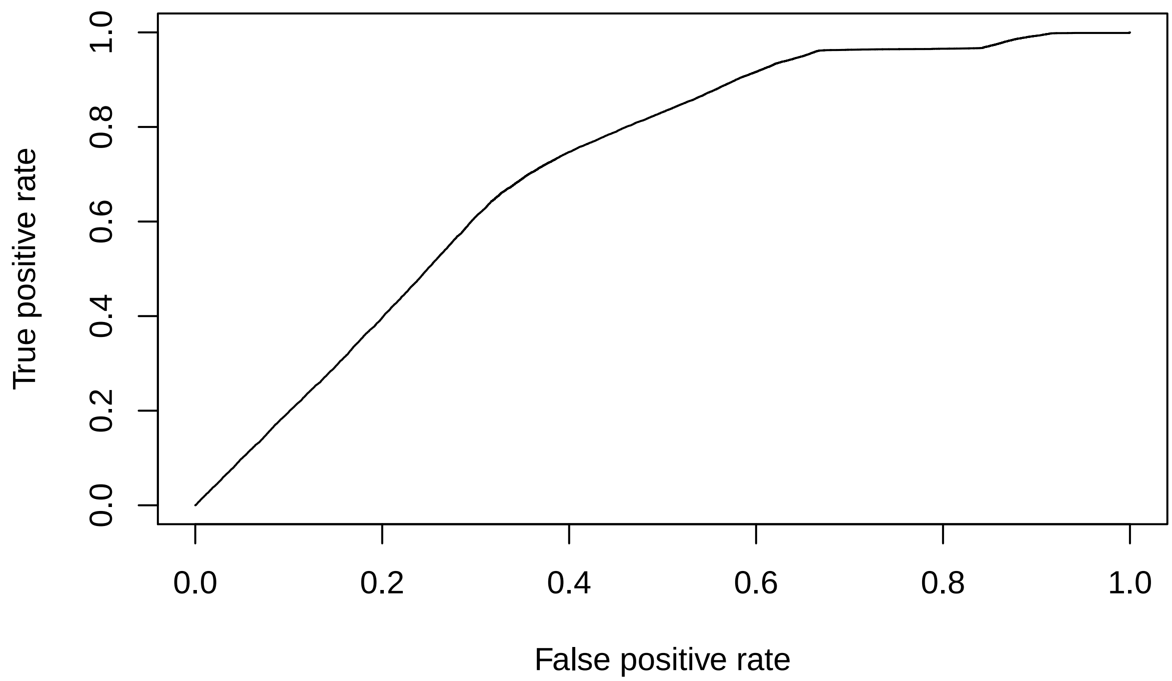
```r
nb.auc1_train=performance(nb.pred1_train,"auc")@y.values[[1]]
nb.auc1_train
```

```
## [1] 0.7191346
```

Test set

resulting ROC curves

```r
nb.ROC1_test=performance(nb.pred1_test,"tpr","fpr")
plot(nb.ROC1_test)
```



AUC

```r
nb.auc1_test=performance(nb.pred1_test,"auc")@y.values[[1]]
nb.auc1_test
```

```
## [1] 0.7132434
```

LDA

```r
lda.fit1=lda(clickthrough~.,data=data1_train)
lda.fit1
```

```
## Call:
## lda(clickthrough ~ ., data = data1_train)
##
```

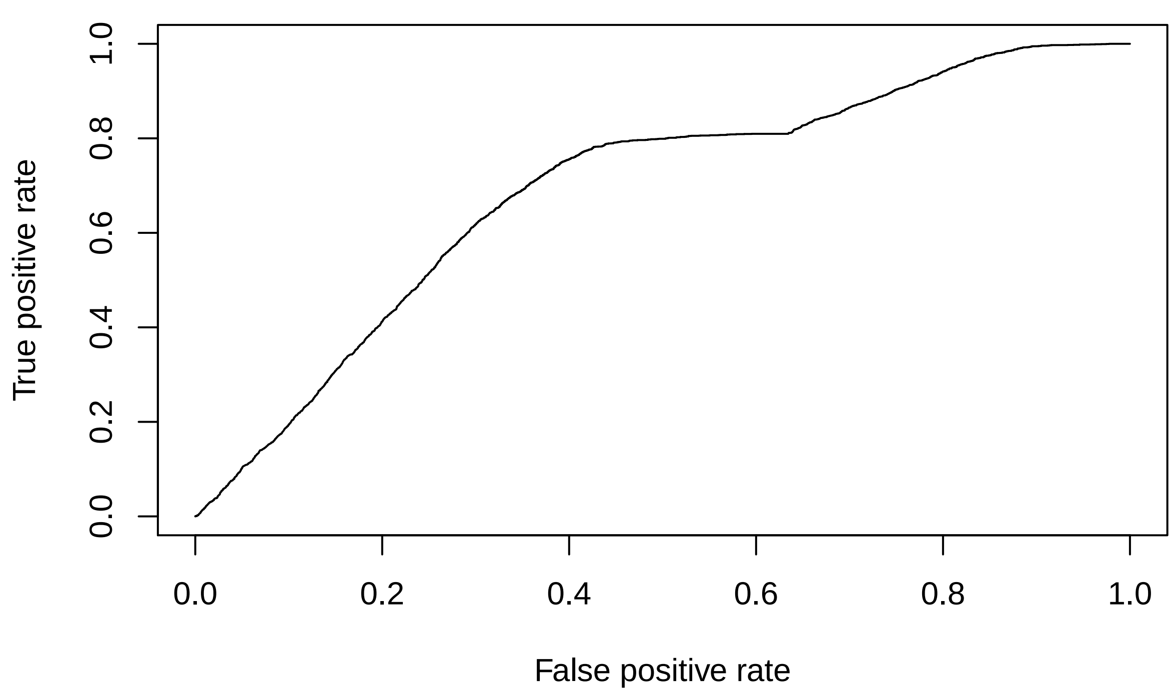
```
## Prior probabilities of groups:
##     FALSE      TRUE
## 0.7548998 0.2451002
##
## Group means:
##          groupb n_results     hour      min
## FALSE 0.3659082  11.89761 12.58479 29.50272
## TRUE  0.1904762  17.60078 12.75651 29.50165
##
## Coefficients of linear discriminants:
##                     LD1
## groupb    -1.417287e+00
## n_results  5.830396e-02
## hour       7.946424e-03
## min       -6.771843e-05
```

```
lda.pred1_train=prediction(predict(lda.fit1,newdata =data1_train)$posterior[,2],
                   data1_train$clickthrough)
lda.pred1_test=prediction(predict(lda.fit1,newdata = data1_test)$posterior[,2],
                   data1_test$clickthrough)
```

Training set

resulting ROC curves

```
lda.ROC1_train=performance(lda.pred1_train,"tpr","fpr")
plot(lda.ROC1_train)
```



AUC

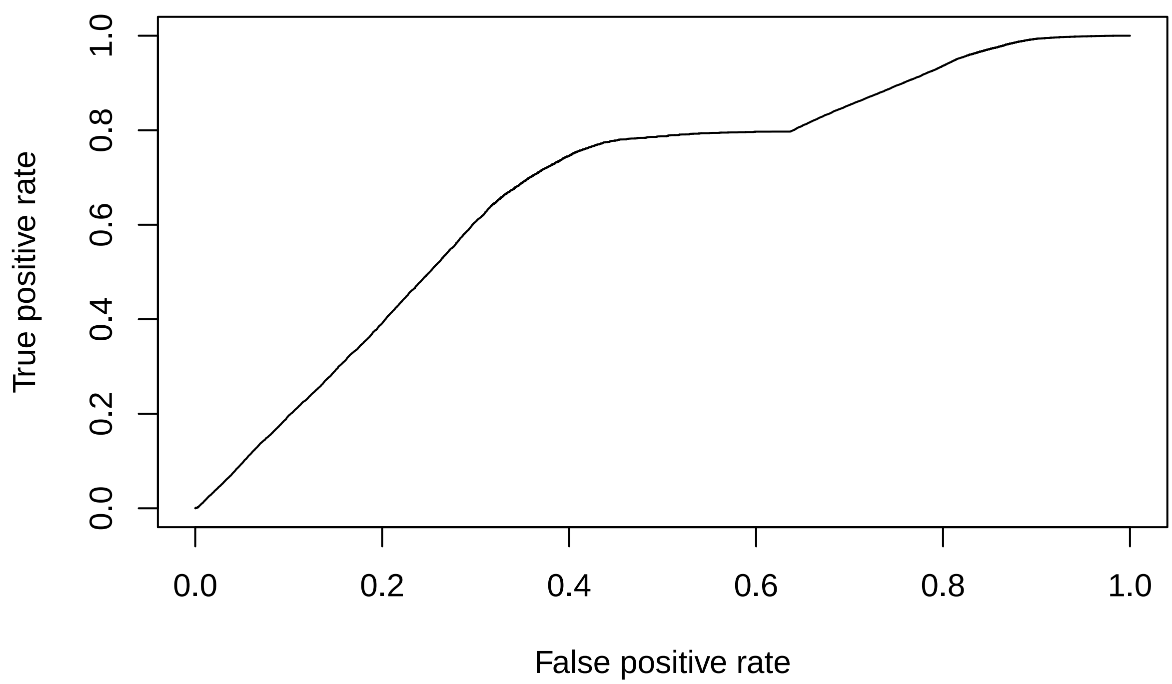
```
lda.auc1_train=performance(lda.pred1_train,"auc")@y.values[[1]]
lda.auc1_train
```

```
## [1] 0.6901326
```

Test set

resulting ROC curves

```
lda.ROC1_test=performance(lda.pred1_test,"tpr","fpr")
plot(lda.ROC1_test)
```



AUC

```
lda.auc1_test=performance(lda.pred1_test,"auc")@y.values[[1]]
lda.auc1_test
```

```
## [1] 0.6817521
```

QDA

```
qda.fit1=qda(clickthrough~.,data=data1_train)
qda.fit1
```

```
## Call:
```

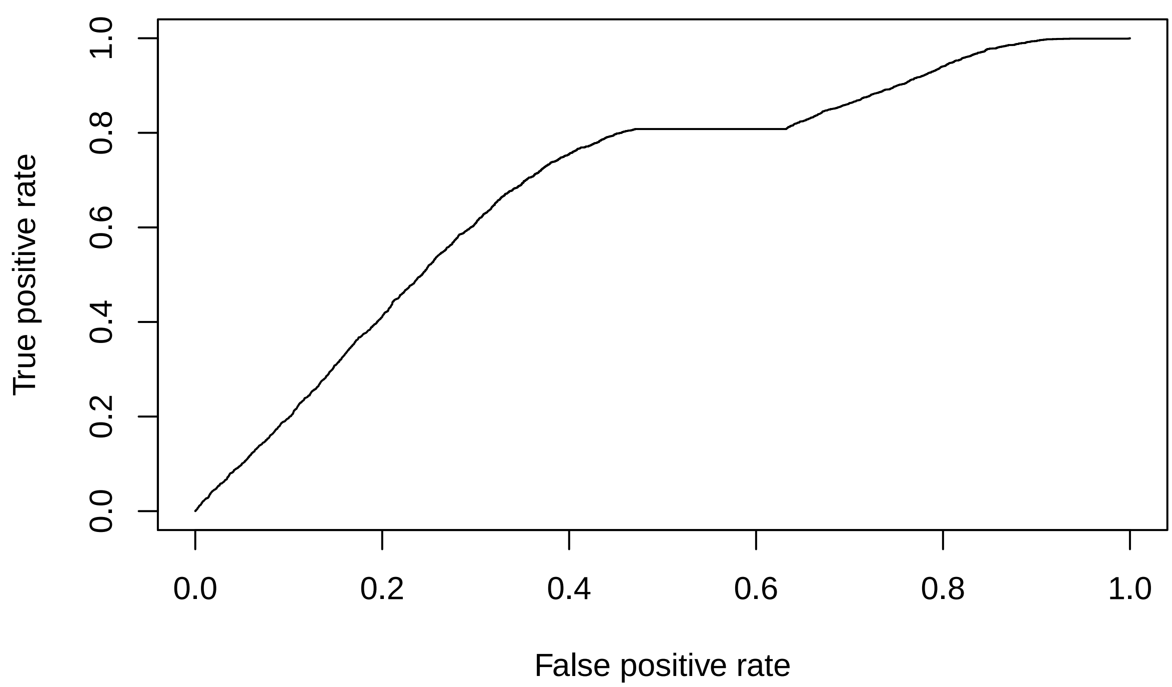
```
## qda(clickthrough ~ ., data = data1_train)
##
## Prior probabilities of groups:
##     FALSE      TRUE
## 0.7548998 0.2451002
##
## Group means:
##          groupb n_results     hour      min
## FALSE 0.3659082  11.89761 12.58479 29.50272
## TRUE  0.1904762  17.60078 12.75651 29.50165
```

```
qda.pred1_train=prediction(predict(qda.fit1,newdata =data1_train)$posterior[,2],
                          data1_train$clickthrough)
qda.pred1_test=prediction(predict(qda.fit1,newdata = data1_test)$posterior[,2],
                          data1_test$clickthrough)
```

Training set

resulting ROC curves

```
qda.ROC1_train=performance(qda.pred1_train,"tpr","fpr")
plot(qda.ROC1_train)
```



AUC

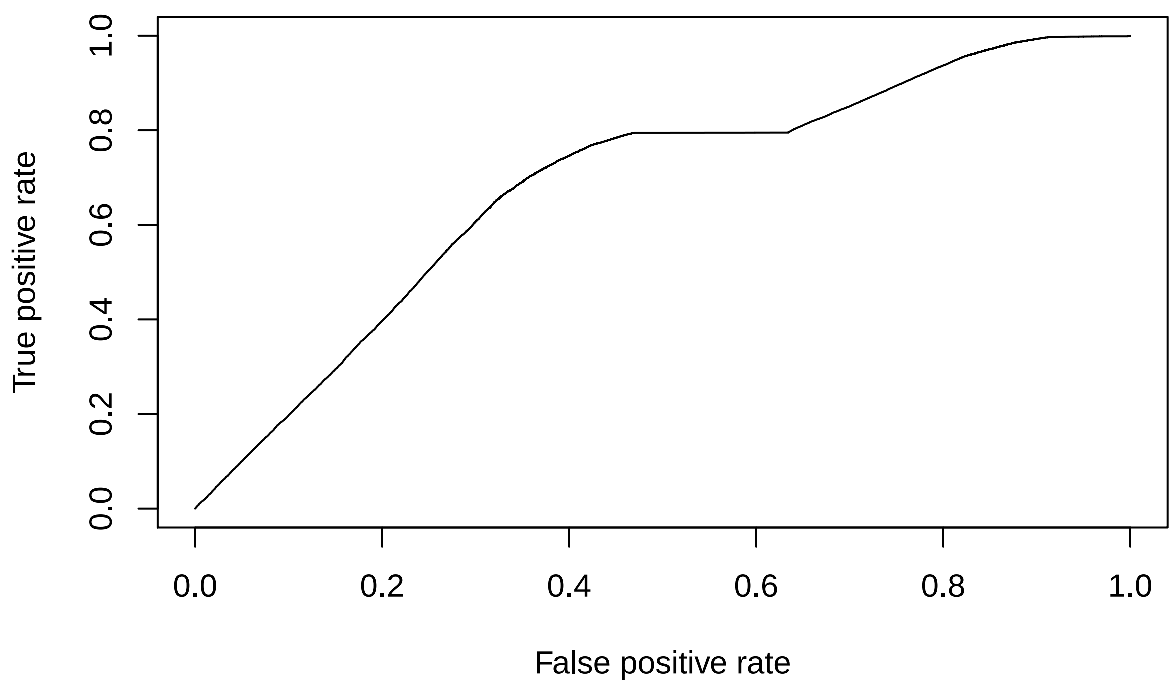
```
qda.auc1_train=performance(qda.pred1_train,"auc")@y.values[[1]]
qda.auc1_train
```

```
## [1] 0.6914456
```

Test set

resulting ROC curves

```r
qda.ROC1_test=performance(qda.pred1_test,"tpr","fpr")
plot(qda.ROC1_test)
```



AUC

```r
qda.auc1_test=performance(qda.pred1_test,"auc")@y.values[[1]]
qda.auc1_test
```

```
## [1] 0.6836562
```

Logistic Regression

```r
lr.fit1=glm(clickthrough~.,data=data1_train,family = binomial)
```

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

```r
summary(lr.fit1)
```

```
##
## Call:
## glm(formula = clickthrough ~ ., family = binomial, data = data1_train)
```

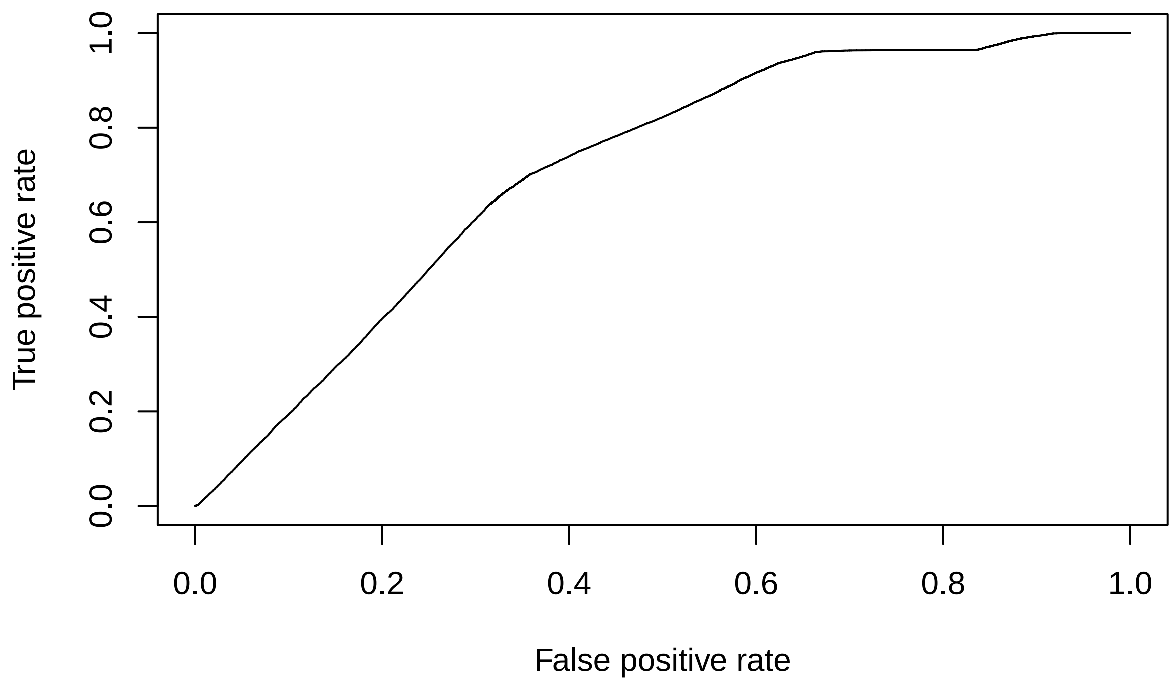
```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.8420  -0.5469  -0.3403   2.3753
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.9304156  0.0740440 -26.071   <2e-16 ***
## groupb      -0.9045517  0.0496294 -18.226   <2e-16 ***
## n_results    0.0691041  0.0027240  25.369   <2e-16 ***
## hour         0.0035756  0.0033448   1.069    0.285
## min          0.0002162  0.0012057   0.179    0.858
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 15173  on 13622  degrees of freedom
## Residual deviance: 14027  on 13618  degrees of freedom
## AIC: 14037
##
## Number of Fisher Scoring iterations: 6
```

```r
lr.pred1_train=prediction(predict(lr.fit1,newdata = data1_test),
                   data1_test$clickthrough)
lr.pred1_test=prediction(predict(lr.fit1,newdata = data1_train),
                   data1_train$clickthrough)
```

Training

resulting ROC curves

```r
lr.ROC1_train=performance(lr.pred1_train,"tpr","fpr")
plot(lr.ROC1_train)
```



AUC

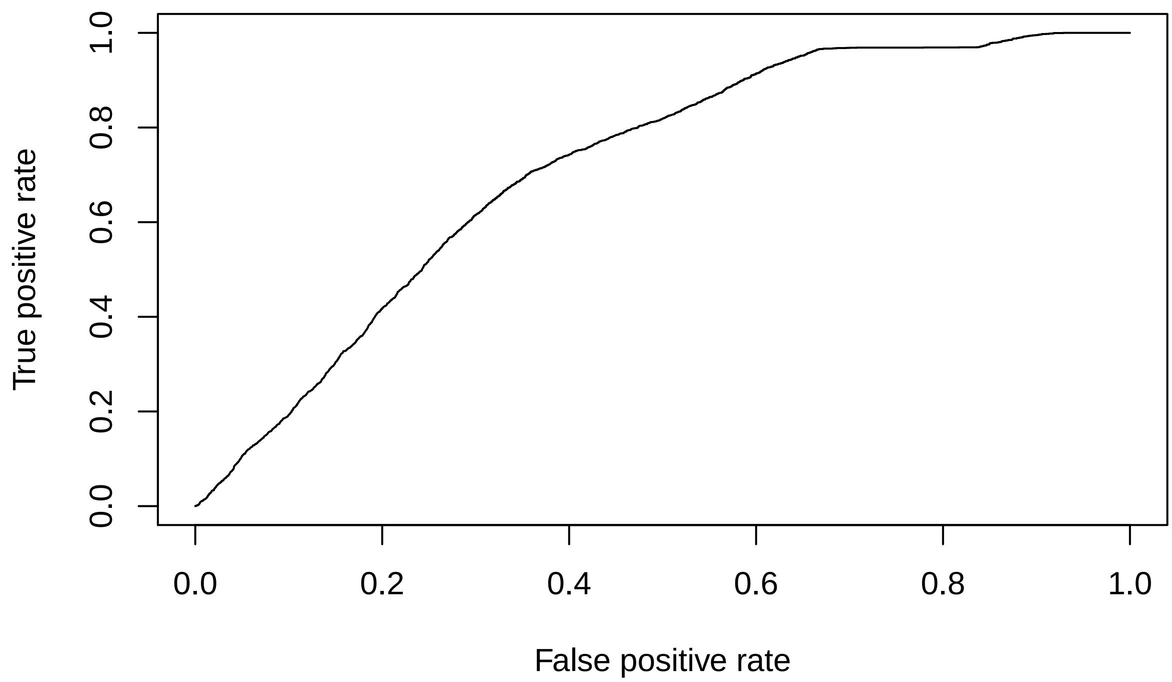
```
lr.auc1_train=performance(lr.pred1_train,"auc")@y.values[[1]]
lr.auc1_train
```

```
## [1] 0.7108408
```

Test set

resulting ROC curves

```
lr.ROC1_test=performance(lr.pred1_test,"tpr","fpr")
plot(lr.ROC1_test)
```



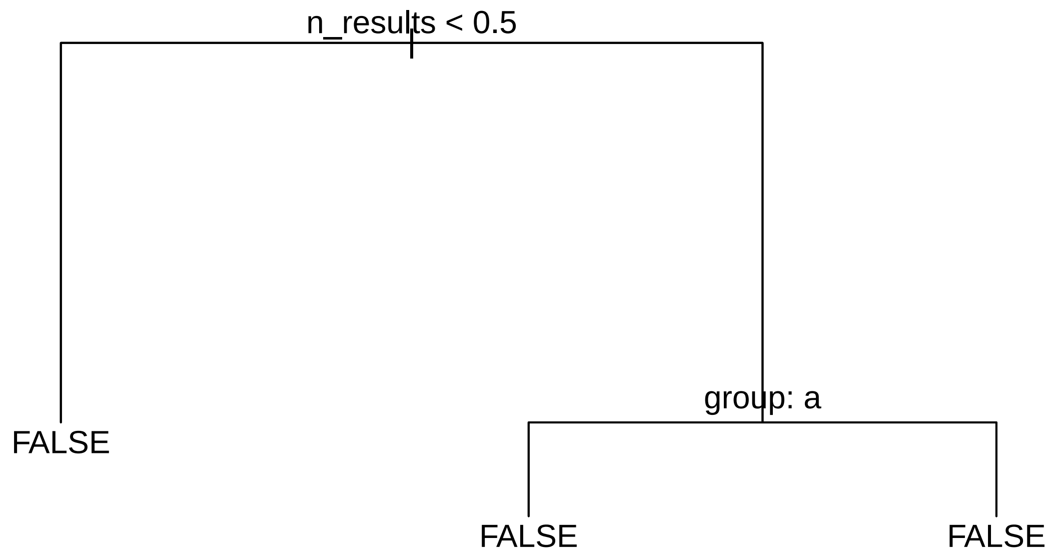
AUC

```
lr.auc1_test=performance(lr.pred1_test,"auc")@y.values[[1]]
lr.auc1_test
```

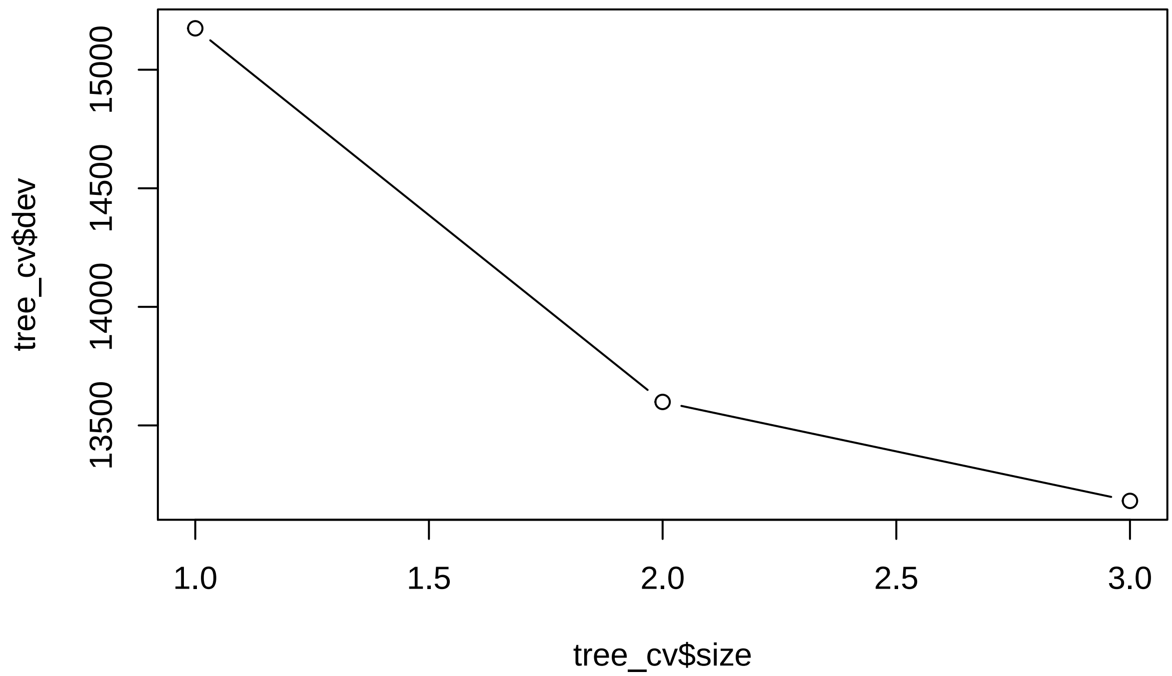
```
## [1] 0.7149299
```

Decision Trees

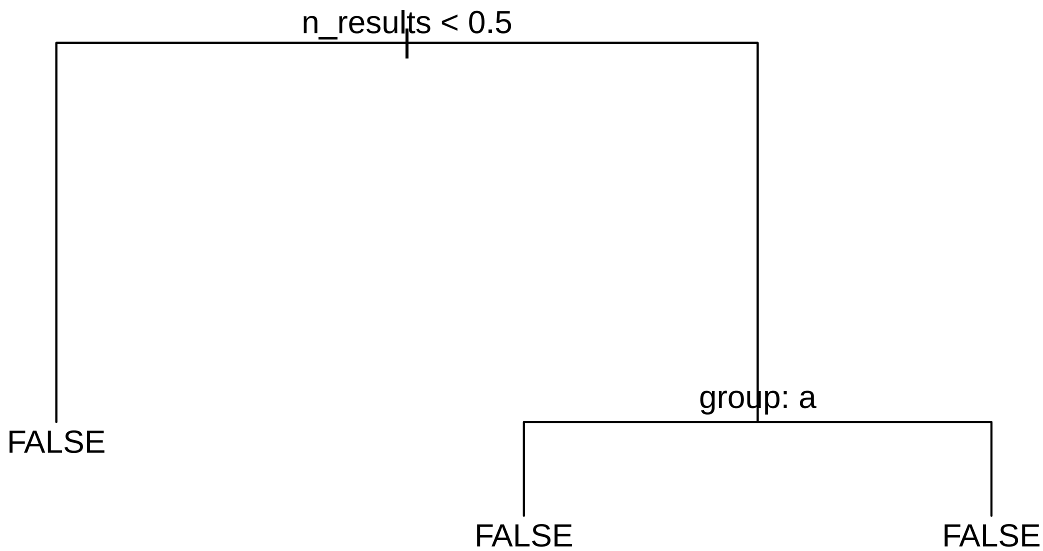
```
tree_1=tree(clickthrough~.,data=data1_train)
plot(tree_1)
text(tree_1,pretty=0)
```




```
tree_cv=cv.tree(tree_1)
plot(tree_cv$size,tree_cv$dev,"b")
```




```
tree_1=prune.tree(tree_1,best=3)
plot(tree_1)
text(tree_1,pretty=0)
```

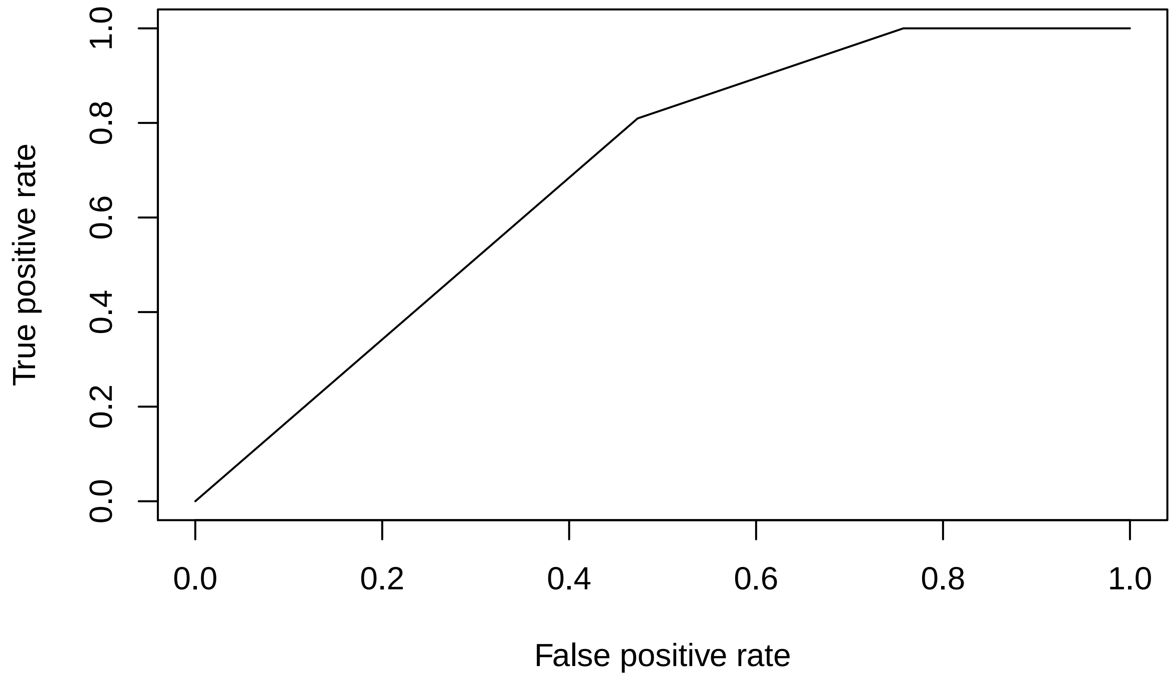



```
tree.pred1_train=prediction(predict(tree_1,newdata = data1_train)[,2],
                   data1_train$clickthrough)
tree.pred1_test=prediction(predict(tree_1,newdata = data1_test)[,2],
                   data1_test$clickthrough)
```

Training set

resulting ROC curves

```
tree.ROC1_train=performance(tree.pred1_train,"tpr","fpr")
plot(tree.ROC1_train)
```



AUC

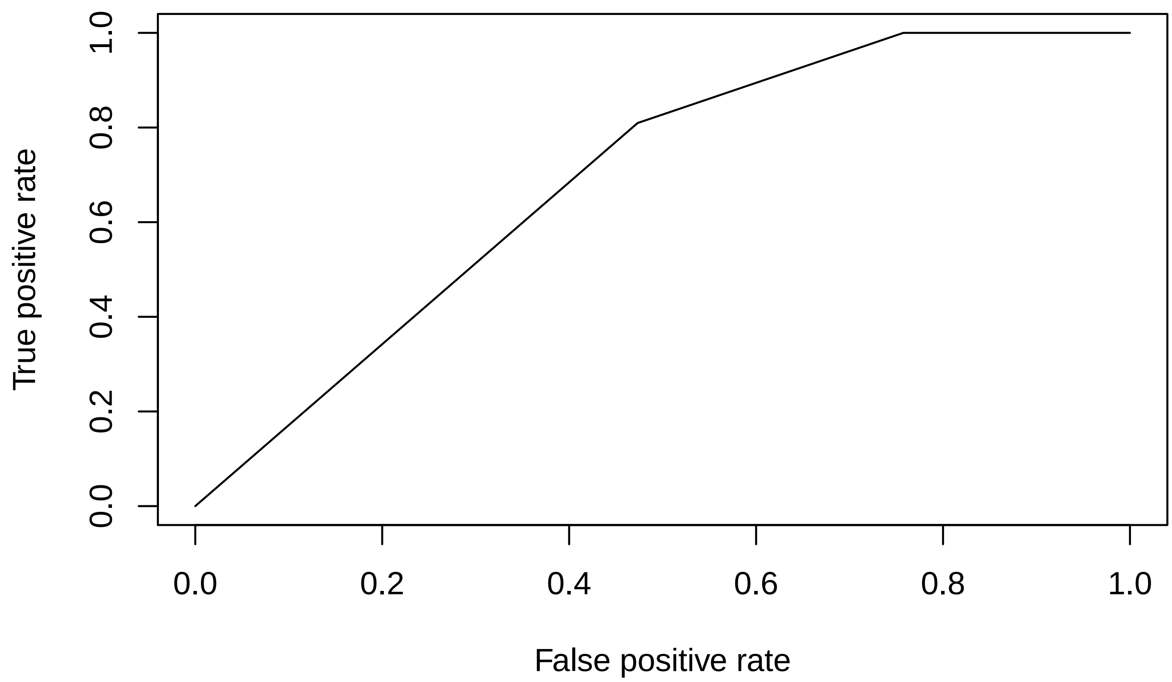
```
tree.auc1_train=performance(tree.pred1_train,"auc")@y.values[[1]]
tree.auc1_train
```

```
## [1] 0.6912194
```

Test set

resulting ROC curves

```
tree.ROC1_test=performance(tree.pred1_test,"tpr","fpr")
plot(tree.ROC1_train)
```



AUC

```
tree.auc1_test=performance(tree.pred1_test,"auc")@y.values[[1]]
tree.auc1_test
```

```
## [1] 0.68707
```

Summary

```
result=matrix(c(nb.auc1_test,lda.auc1_test,qda.auc1_test,lr.auc1_test,
                tree.auc1_test),nrow = 1,ncol = 5)
colnames(result)=c("Naive Bayes","LDA","QDA","Logistic Regression",
                   "Decision Tree")
result
```

```
##      Naive Bayes       LDA       QDA Logistic Regression Decision Tree
## [1,]   0.7132434 0.6817521 0.6836562           0.7149299       0.68707
```

From the result, we can see that Logistic Regression has the best performance and Naive Bayes has better one than rest of three. LDA, QDA and Decision Tree are relatively not doing well.

The Naive Bayes does well than LDA and QDA tells that it is reasonable to assume that 4 variables *group*,*n_results*,*hour*,*min* are independent rather than correlate, which is intuitive as well.

The Logistic Regression does well than Decision tree tells that the boundary of class of *TRUE* and *FALSE* is not well defined. Moreover, all of the region in decison tree are defined as *FALSE*, which also indicats that decision tree is not doing well here.

The variable *group* and *n_results* are sigificant in logistic regression and they are also the label of all internal nodes. This shows that these two variables are found to be important.

Q2. 30 Second Check-in

Process data

```
(data_temp=data_origin %>%
  arrange(session_id,timestamp) %>%
  filter(action=="searchResultPage"|action=="visitPage"|
           (action=="checkin"& checkin>=30)) %>%
  group_by(session_id) %>%
  mutate(
    lead_action=lead(action)
  )%>%
  filter(action=="searchResultPage" |
           (action=="visitPage" & lead_action=="checkin"))
 )
```

```
## # A tibble: 159,503 x 10
## # Groups:   session_id [68,003]
##     uuid  timestamp session_id group action checkin page_id n_results
##     <chr>     <dbl> <chr>      <chr> <chr>    <dbl> <chr>       <dbl>
##   1 e6f9~   2.02e13 0000cbcb6~ b     searc~      NA fdeeb9~        20
##   2 5b39~   2.02e13 0001382e0~ b     searc~      NA 7aa28c~        18
##   3 cae3~   2.02e13 0001382e0~ b     visit~      NA f88793~        NA
##   4 5138~   2.02e13 0001e8bb9~ b     searc~      NA 6b7f88~        20
##   5 948f~   2.02e13 0001e8bb9~ b     visit~      NA 35ee99~        NA
##   6 dd69~   2.02e13 000216cf1~ a     searc~      NA 08cebd~        20
##   7 7759~   2.02e13 000216cf1~ a     searc~      NA a9e6d0~        20
##   8 708e~   2.02e13 000216cf1~ a     searc~      NA fdce0b~        20
##   9 56de~   2.02e13 000216cf1~ a     searc~      NA 073960~        20
##  10 f69c~   2.02e13 000216cf1~ a     searc~      NA 044c95~        20
## # ... with 159,493 more rows, and 2 more variables: result_position <dbl>,
## #   lead_action <chr>
```

It is easy to see that the action *visitPage* whose *lead_action* == *checkin* is the visit at this page for at least 30 seconds or more. Just filter all *searchResultPage* and those special *visitPage*

```
(data2=data_temp %>%
  group_by(session_id) %>%
  mutate(
    lead_action_2=lead(action)
  ) %>%
  filter(action=="searchResultPage") %>%
  mutate(
    check30=if_else(is.na(lead_action_2)==TRUE | lead_action_2!="visitPage","FALSE","TRUE")
  ) %>%
   mutate(
    hour=substr(timestamp,9,10),
    min=substr(timestamp,11,12)
  ) %>%
    ungroup() %>%
   dplyr::select(check30,group,n_results,hour,min) %>%
   mutate(
```

```
      check30=as.factor(check30),
      group=as.factor(group),
      hour=as.integer(hour),
      min=as.integer(min)
    )
)
```

```
## # A tibble: 136,234 x 5
##    check30 group n_results  hour   min
##    <fct>   <fct>     <dbl> <int> <int>
##  1 FALSE   b            20    15    20
##  2 TRUE    b            18     8    49
##  3 TRUE    b            20     9    24
##  4 FALSE   a            20    16    19
##  5 FALSE   a            20    16    19
##  6 FALSE   a            20    16    20
##  7 FALSE   a            20    16    20
##  8 FALSE   a            20    16    20
##  9 FALSE   a            20    16    20
## 10 FALSE   b             1     5    33
## # ... with 136,224 more rows
```

Use lead function again. With same logic, if a *searchResultPage*'s *lead_action_2* is "visitPage", then user clicked on a link after this search, and then checked in at this page for at least 30 seconds or more. The *checkin* for this condition should be "TRUE"

Divide data into training set and test set

```
train_index2=sample(dim(data2)[1],round(dim(data2)[1]*0.1))
data2_train=data2[train_index2,]
data2_test=data2[-(train_index2),]
```

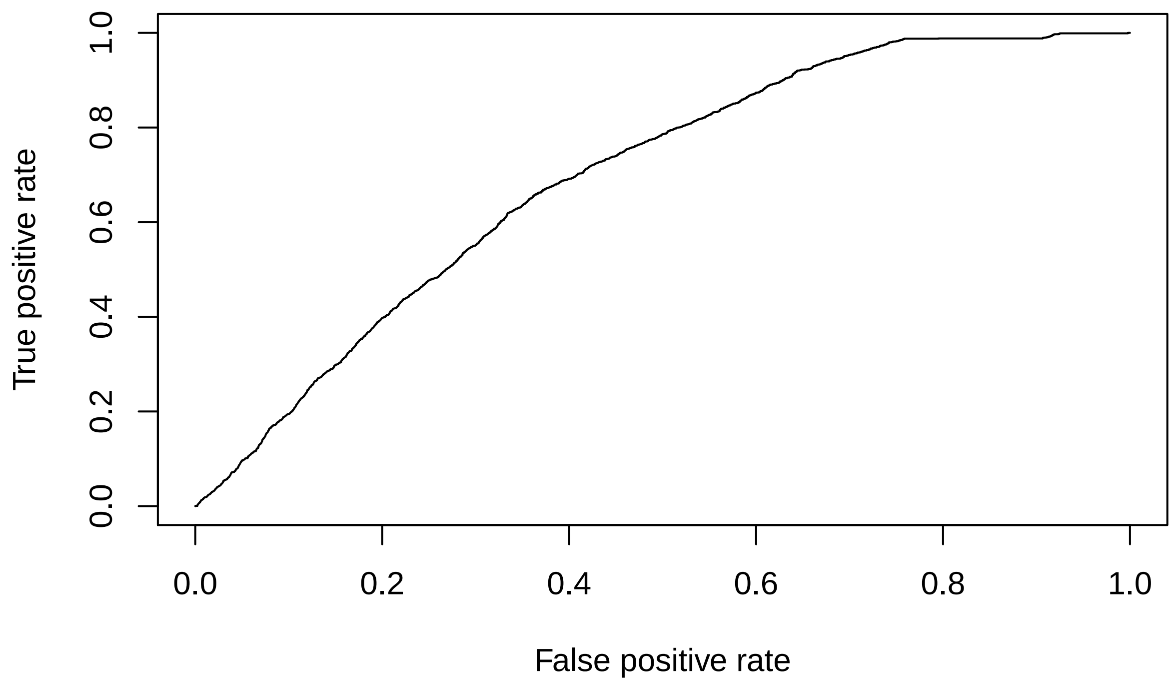
Naive Bayes

```
options(backup_options)
nb.fit2=naiveBayes(x=data2_train[,-1],y=data2_train$check30)

nb.pred2_train=prediction(predict(nb.fit2,newdata=data2_train,type="raw")[,2],
                          data2_train$check30)

nb.pred2_test=prediction(predict(nb.fit2,newdata=data2_test,type="raw")[,2],
                         data2_test$check30)
```

Training set

resulting ROC curves

```
nb.ROC2_train=performance(nb.pred2_train,"tpr","fpr")
plot(nb.ROC2_train)
```



AUC

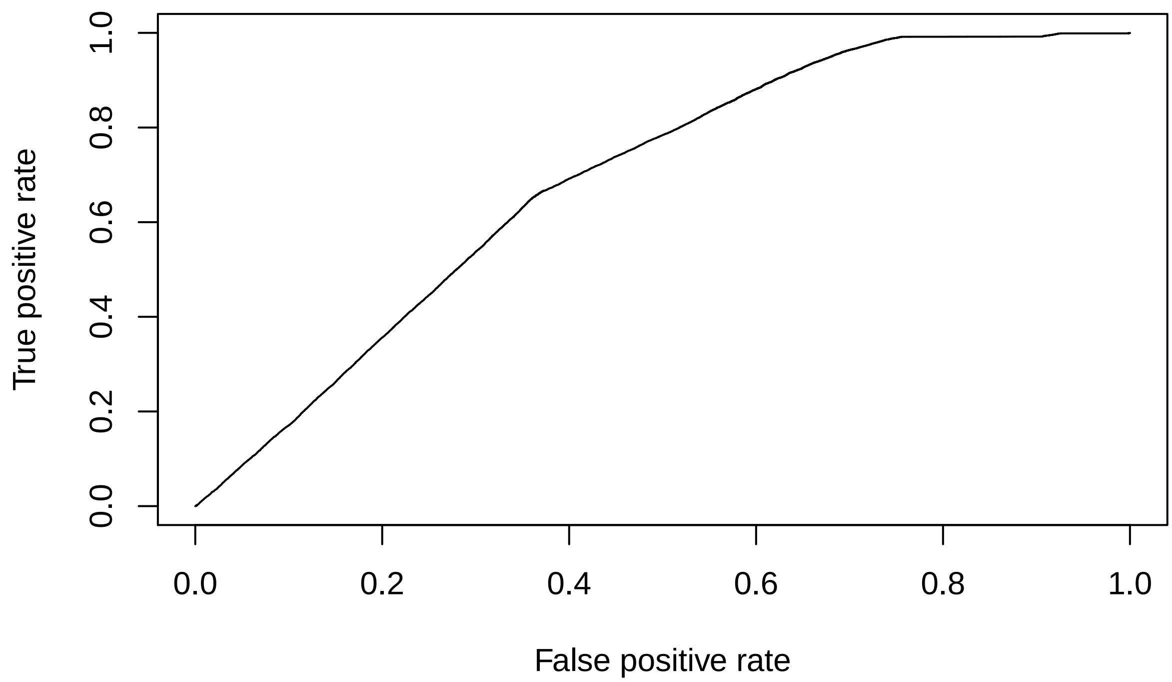
```
nb.auc2_train=performance(nb.pred2_train,"auc")@y.values[[1]]
nb.auc2_train
```

```
## [1] 0.6949286
```

Test set

resulting ROC curves

```
nb.ROC2_test=performance(nb.pred2_test,"tpr","fpr")
plot(nb.ROC2_test)
```



AUC

```
nb.auc2_test=performance(nb.pred2_test,"auc")@y.values[[1]]
nb.auc2_test
```

```
## [1] 0.6889426
```

LDA

```
lda.fit2=lda(check30~.,data=data2_train)
lda.fit2
```

```
## Call:
## lda(check30 ~ ., data = data2_train)
##
## Prior probabilities of groups:
##    FALSE     TRUE
## 0.844234 0.155766
##
## Group means:
##          groupb n_results     hour      min
## FALSE 0.3445787  12.52048 12.58899 29.69577
## TRUE  0.2087653  17.42790 12.71112 29.64892
##
## Coefficients of linear discriminants:
##                      LD1
```

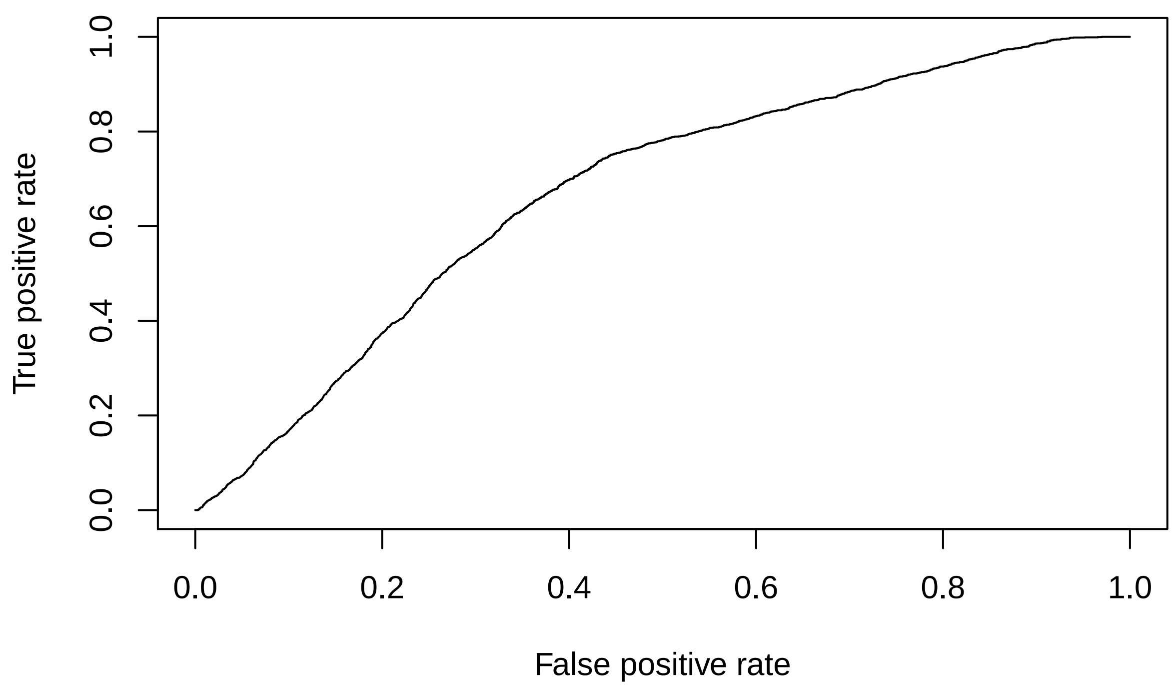
```
## groupb    -1.2721231398
## n_results  0.0633216461
## hour       0.0053651500
## min       -0.0008822715
```

```
lda.pred2_train=prediction(predict(lda.fit2,newdata =data2_train)$posterior[,2],
                  data2_train$check30)
lda.pred2_test=prediction(predict(lda.fit2,newdata = data2_test)$posterior[,2],
                  data2_test$check30)
```

Training set

resulting ROC curves

```
lda.ROC2_train=performance(lda.pred2_train,"tpr","fpr")
plot(lda.ROC2_train)
```



AUC

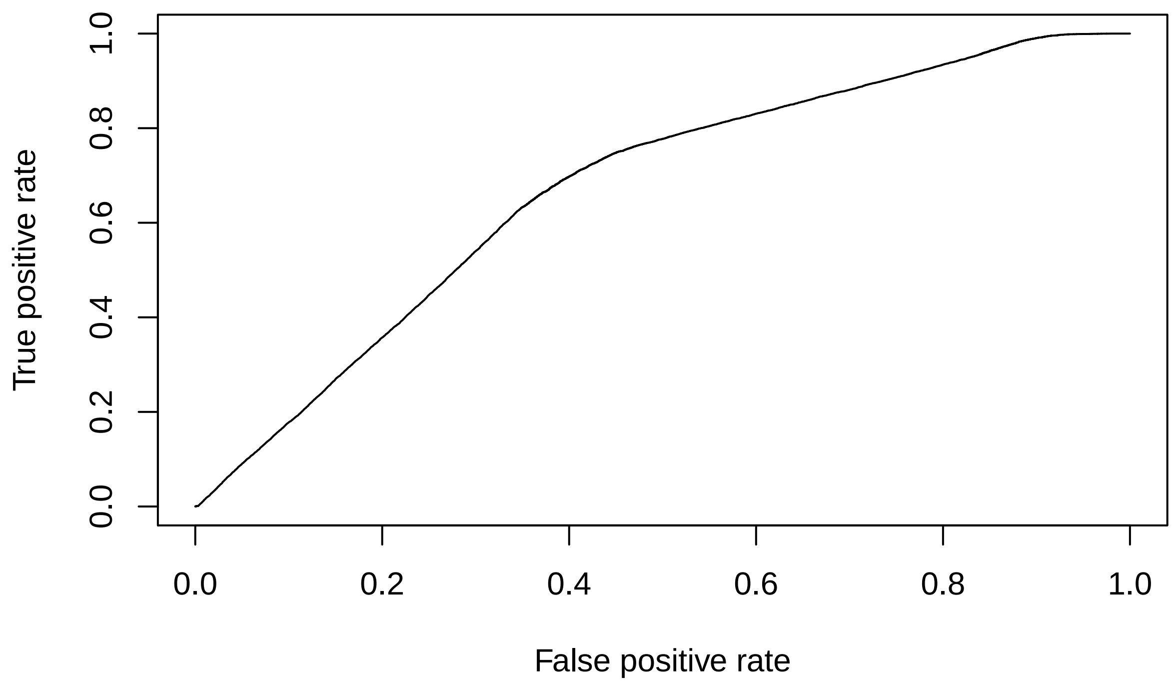
```
lda.auc2_train=performance(lda.pred2_train,"auc")@y.values[[1]]
lda.auc2_train
```

```
## [1] 0.6727161
```

Test set

resulting ROC curves

```r
lda.ROC2_test=performance(lda.pred2_test,"tpr","fpr")
plot(lda.ROC2_test)
```



AUC

```r
lda.auc2_test=performance(lda.pred2_test,"auc")@y.values[[1]]
lda.auc2_test
```

```
## [1] 0.6702053
```

QDA

```r
qda.fit2=qda(check30~.,data=data2_train)
qda.fit2
```

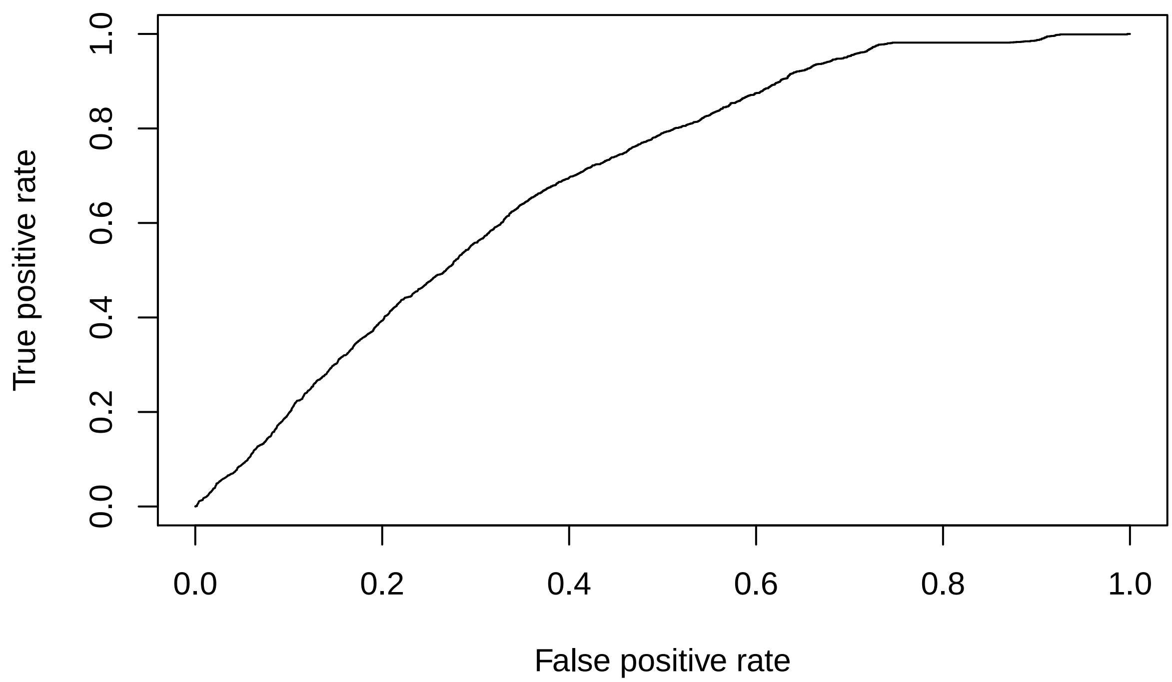
```
## Call:
## qda(check30 ~ ., data = data2_train)
##
## Prior probabilities of groups:
##    FALSE     TRUE
## 0.844234 0.155766
##
## Group means:
##          groupb n_results     hour      min
## FALSE 0.3445787  12.52048 12.58899 29.69577
## TRUE  0.2087653  17.42790 12.71112 29.64892
```

```r
qda.pred2_train=prediction(predict(qda.fit2,newdata =data2_train)$posterior[,2],
                           data2_train$check30)
qda.pred2_test=prediction(predict(qda.fit2,newdata = data2_test)$posterior[,2],
                           data2_test$check30)
```

Training set

resulting ROC curves

```r
qda.ROC2_train=performance(qda.pred2_train,"tpr","fpr")
plot(qda.ROC2_train)
```



AUC

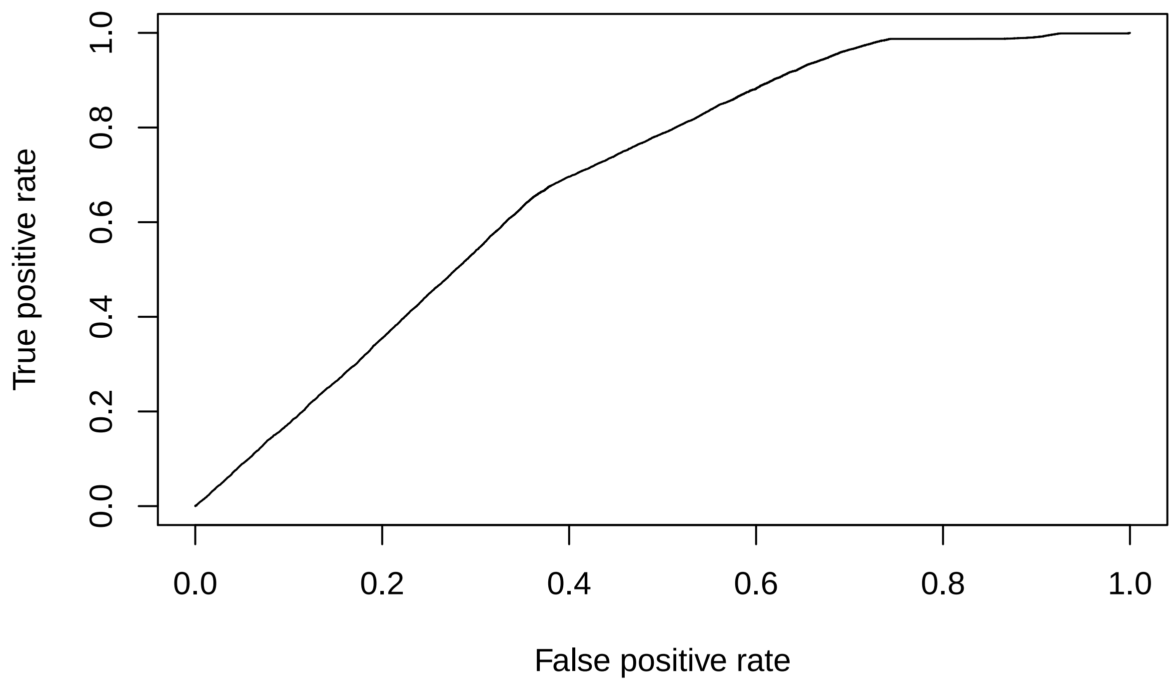
```r
qda.auc2_train=performance(qda.pred2_train,"auc")@y.values[[1]]
qda.auc2_train
```

```
## [1] 0.6951857
```

Test set

resulting ROC curves

```r
qda.ROC2_test=performance(qda.pred2_test,"tpr","fpr")
plot(qda.ROC2_test)
```



AUC

```
qda.auc2_test=performance(qda.pred2_test,"auc")@y.values[[1]]
qda.auc2_test
```

```
## [1] 0.6897433
```

Logistic Regression

```
lr.fit2=glm(check30~.,data=data2_train,family = binomial)
summary(lr.fit2)
```

```
##
## Call:
## glm(formula = check30 ~ ., family = binomial, data = data2_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.3968  -0.7020  -0.5157  -0.3658   2.3975
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.1817646  0.0836080 -26.095   <2e-16 ***
## groupb      -0.6702659  0.0573589 -11.685   <2e-16 ***
## n_results    0.0451919  0.0028611  15.795   <2e-16 ***
## hour         0.0026397  0.0038869   0.679    0.497
```

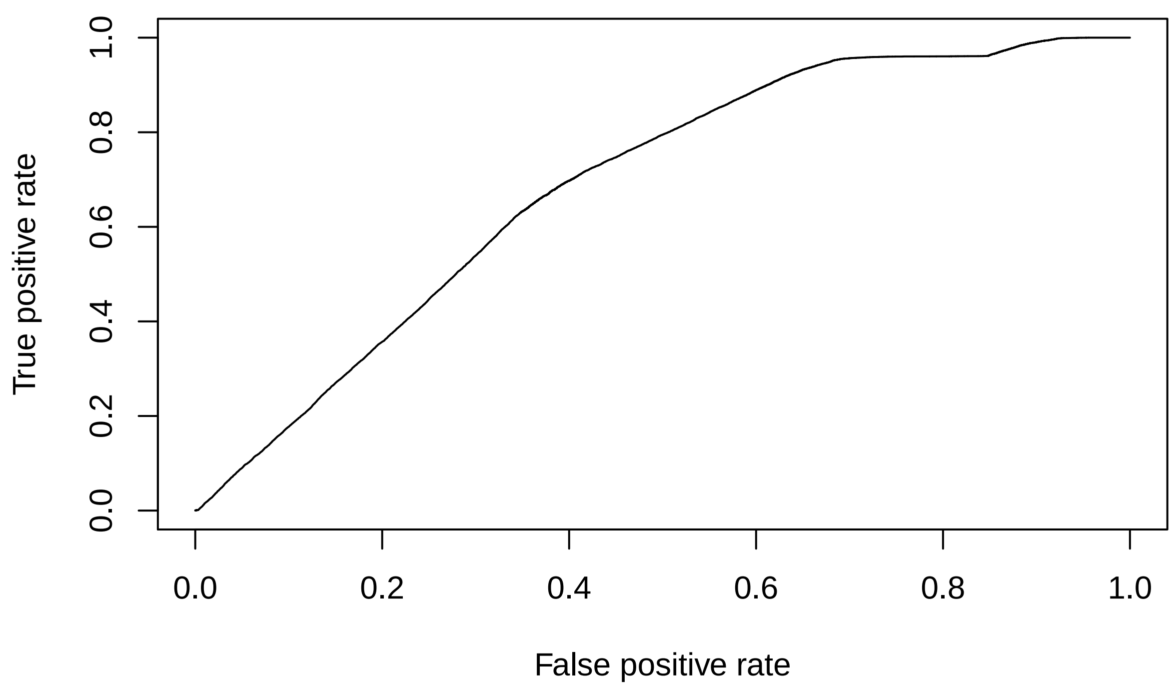
```
## min         -0.0003345  0.0013973  -0.239    0.811
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11786  on 13622  degrees of freedom
## Residual deviance: 11316  on 13618  degrees of freedom
## AIC: 11326
##
## Number of Fisher Scoring iterations: 5
```

```
lr.pred2_train=prediction(predict(lr.fit2,newdata = data2_test),
                   data2_test$check30)
lr.pred2_test=prediction(predict(lr.fit2,newdata = data2_train),
                   data2_train$check30)
```

Training

resulting ROC curves

```
lr.ROC2_train=performance(lr.pred2_train,"tpr","fpr")
plot(lr.ROC2_train)
```



AUC

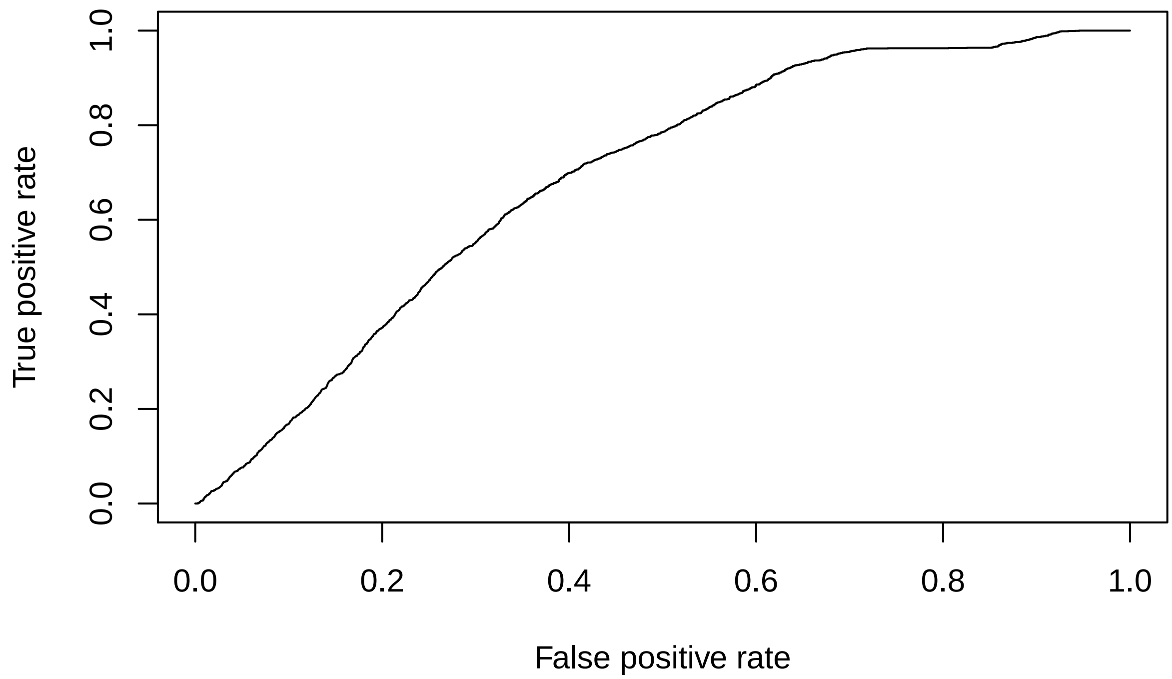
```
lr.auc2_train=performance(lr.pred2_train,"auc")@y.values[[1]]
lr.auc2_train
```

```
## [1] 0.6874952
```

Test set

resulting ROC curves

```
lr.ROC2_test=performance(lr.pred2_test,"tpr","fpr")
plot(lr.ROC2_test)
```



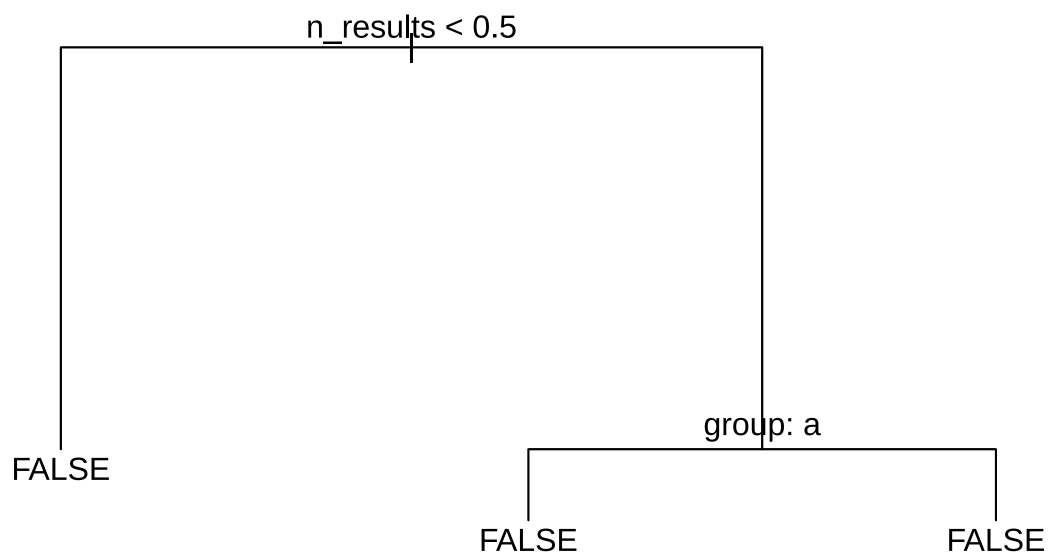
AUC

```
lr.auc2_test=performance(lr.pred2_test,"auc")@y.values[[1]]
lr.auc2_test
```

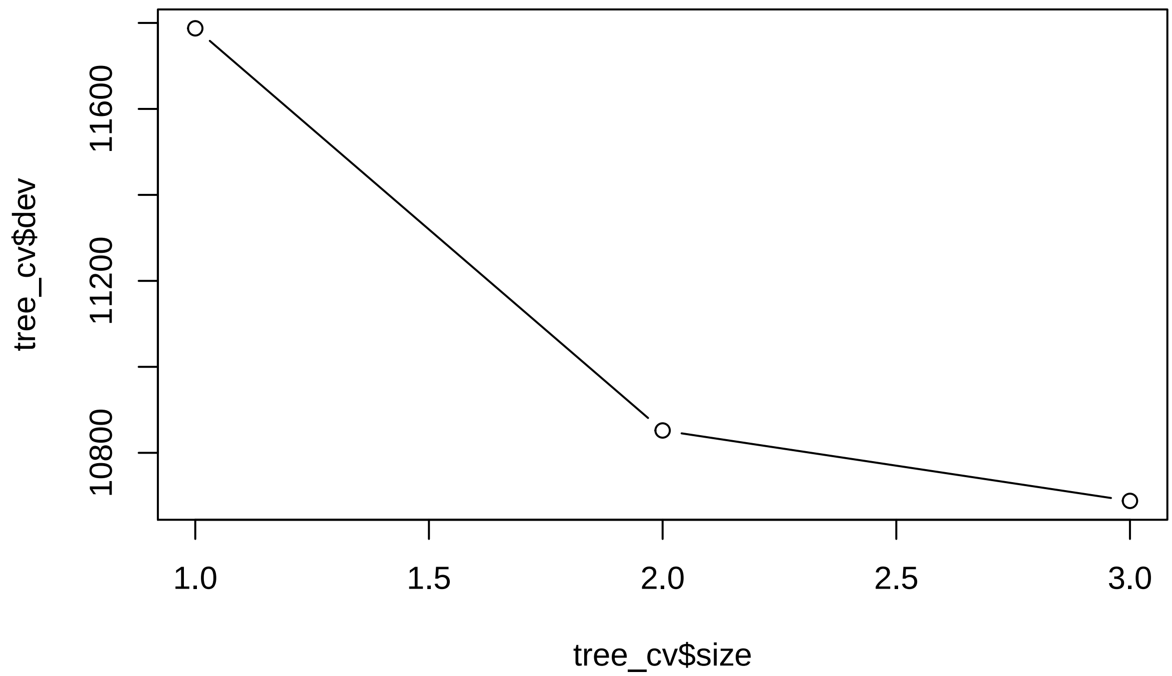
```
## [1] 0.6881127
```

Decision Trees

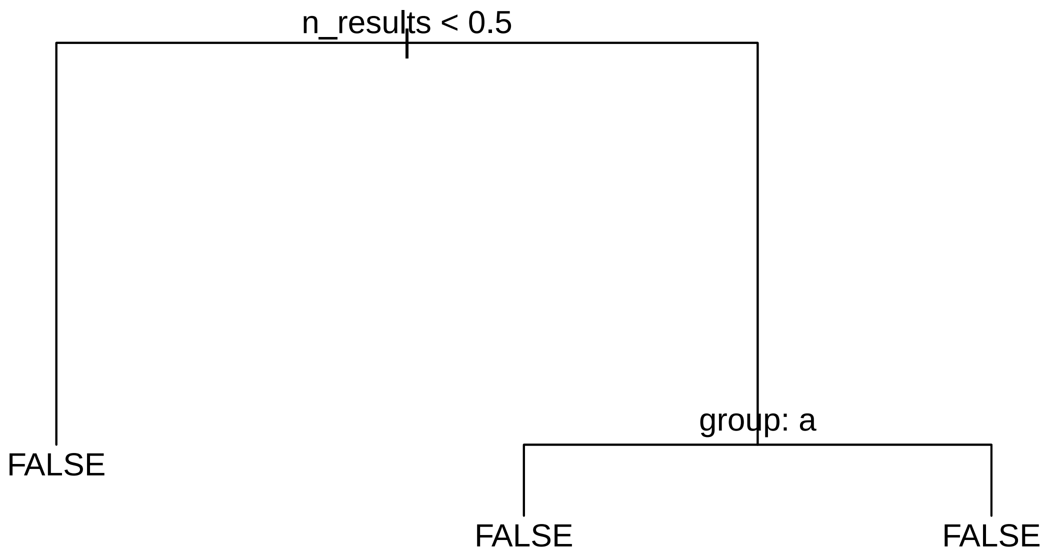
```
tree_2=tree(check30~.,data=data2_train)
plot(tree_2)
text(tree_2,pretty=0)
```




```
tree_cv=cv.tree(tree_2)
plot(tree_cv$size,tree_cv$dev,"b")
```




```r
tree_2=prune.tree(tree_2,best=3)
plot(tree_2)
text(tree_2,pretty=0)
```

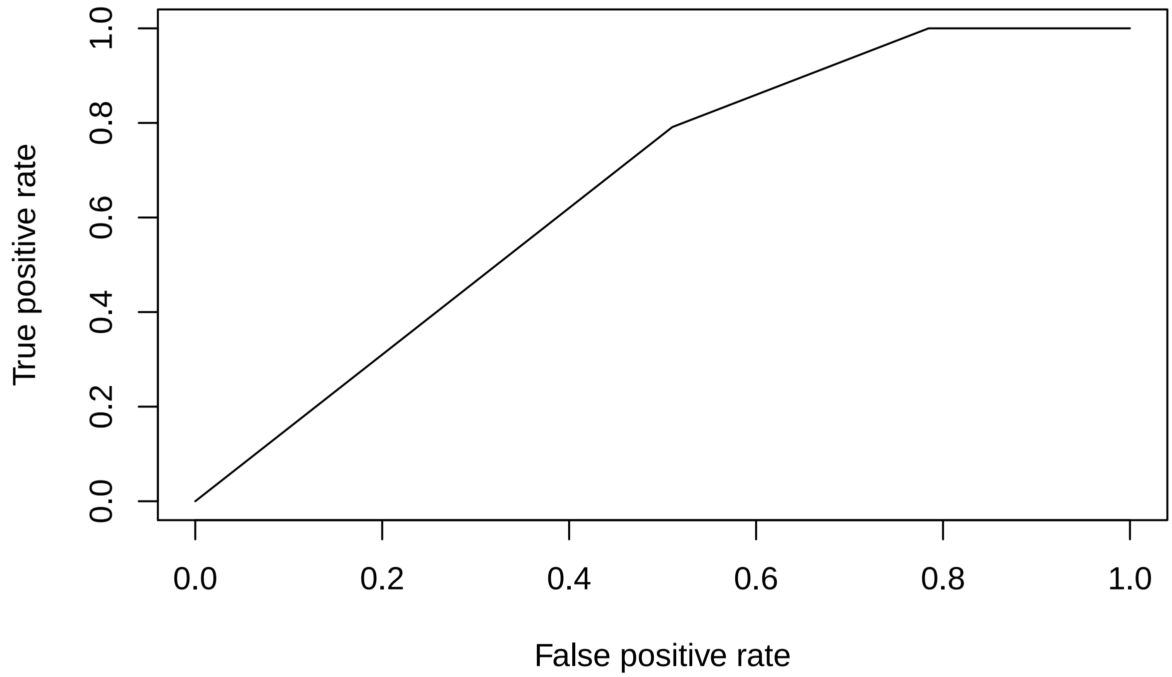



```
tree.pred2_train=prediction(predict(tree_2,newdata = data2_train)[,2],
                   data2_train$check30)
tree.pred2_test=prediction(predict(tree_2,newdata = data2_test)[,2],
                   data2_test$check30)
```

Training set

resulting ROC curves

```
tree.ROC2_train=performance(tree.pred2_train,"tpr","fpr")
plot(tree.ROC2_train)
```



AUC

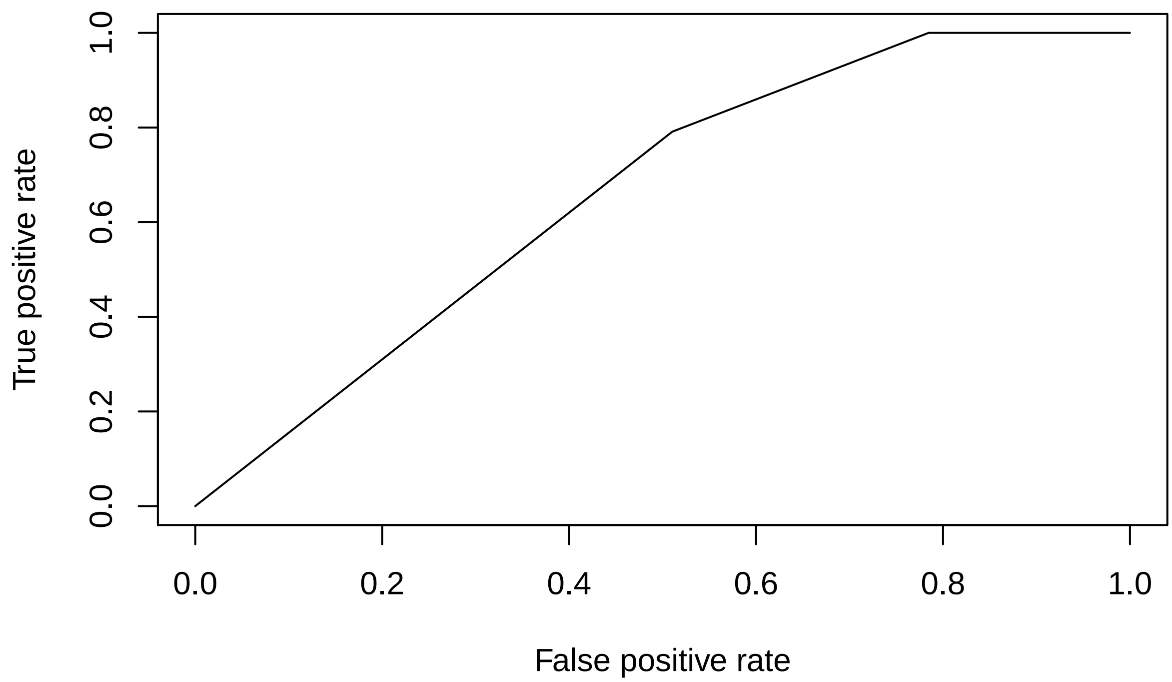
```
tree.auc2_train=performance(tree.pred2_train,"auc")@y.values[[1]]
tree.auc2_train
```

```
## [1] 0.6629124
```

Test set

resulting ROC curves

```
tree.ROC2_test=performance(tree.pred2_test,"tpr","fpr")
plot(tree.ROC2_train)
```



AUC

```
tree.auc2_test=performance(tree.pred2_test,"auc")@y.values[[1]]
tree.auc2_test
```

```
## [1] 0.6613817
```

Q3. Intro to Avito Duplicate Ads Detection

Read data

```
setwd("E:/Avito Duplicate Ads Detection/avito-duplicate-ads-detection")

library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

```r
library(readr)
library(lattice)
library(caret)
```

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

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

```r
library(stringdist)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

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

```r
location=fread("input/Location.csv")
itemPairsTest=fread("input/ItemPairs_test.csv")
itemPairsTrain=fread("input/ItemPairs_train.csv")
itemInfoTest=read_csv("input/ItemInfo_test.csv")
```

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

```r
itemInfoTrain=read_csv("input/ItemInfo_train.csv")
```

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

```r
itemInfoTest=data.table(itemInfoTest)
itemInfoTrain=data.table(itemInfoTrain)

setkey(location, locationID)
setkey(itemInfoTrain, itemID)
setkey(itemInfoTest, itemID)
```

Drop unused factors

```r
dropAndNumChar = function(itemInfo){
  itemInfo[, ':=' (ncharTitle = nchar(title),
                   ncharDescription = nchar(description),
                   description = NULL,
                   images_array = NULL,
                   attrsJSON = NULL)]
}

dropAndNumChar(itemInfoTest)
dropAndNumChar(itemInfoTrain)
```

Merge ItemPairs and ItemInfo

```r
mergeInfo = function(itemPairs, itemInfo){
  # merge on itemID_1
  setkey(itemPairs, itemID_1)
  itemPairs = itemInfo[itemPairs]
  setnames(itemPairs, names(itemInfo), paste0(names(itemInfo), "_1"))
  # merge on itemID_2
  setkey(itemPairs, itemID_2)
  itemPairs = itemInfo[itemPairs]
  setnames(itemPairs, names(itemInfo), paste0(names(itemInfo), "_2"))
  # merge on locationID_1
  setkey(itemPairs, locationID_1)
  itemPairs = location[itemPairs]
  setnames(itemPairs, names(location), paste0(names(location), "_1"))
  # merge on locationID_2
  setkey(itemPairs, locationID_2)
  itemPairs = location[itemPairs]
  setnames(itemPairs, names(location), paste0(names(location), "_2"))
  return(itemPairs)
}

itemPairsTrain = mergeInfo(itemPairsTrain, itemInfoTrain)
itemPairsTest = mergeInfo(itemPairsTest, itemInfoTest)

rm(list=c("itemInfoTest", "itemInfoTrain", "location"))
```

Create 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)))
}

createFeatures = function(itemPairs){
  itemPairs[, ':=' (locationMatch = matchPair(locationID_1, locationID_2),
                    locationID_1 = NULL,
                    locationID_2 = NULL,
                    regionMatch = matchPair(regionID_1, regionID_2),
                    regionID_1 = NULL,
                    regionID_2 = NULL,
                    metroMatch = matchPair(metroID_1, metroID_2),
                    metroID_1 = NULL,
                    metroID_2 = NULL,
                    categoryID_1 = NULL,
                    categoryID_2 = NULL,
                    priceMatch = matchPair(price_1, price_2),
                    priceDiff = pmax(price_1/price_2, price_2/price_1),
                    priceMin = pmin(price_1, price_2, na.rm=TRUE),
                    priceMax = pmax(price_1, price_2, na.rm=TRUE),
                    price_1 = NULL,
                    price_2 = NULL,
                    titleStringDist = stringdist(title_1, title_2, method = "jw"),
                    titleStringDist2 = (stringdist(title_1, title_2, method = "lcs") /
                        pmax(ncharTitle_1, ncharTitle_2, na.rm=TRUE)),
                    title_1 = NULL,
                    title_2 = NULL,
                    titleCharDiff = pmax(ncharTitle_1/ncharTitle_2, ncharTitle_2/ncharTitle_1),
                    titleCharMin = pmin(ncharTitle_1, ncharTitle_2, na.rm=TRUE),
                    titleCharMax = pmax(ncharTitle_1, ncharTitle_2, na.rm=TRUE),
                    ncharTitle_1 = NULL,
                    ncharTitle_2 = NULL,
                    descriptionCharDiff = pmax(ncharDescription_1/ncharDescription_2, ncharDescription_
                    descriptionCharMin = pmin(ncharDescription_1, ncharDescription_2, na.rm=TRUE),
                    descriptionCharMax = pmax(ncharDescription_1, ncharDescription_2, na.rm=TRUE),
                    ncharDescription_1 = NULL,
                    ncharDescription_2 = NULL,
                    distance = sqrt((lat_1-lat_2)^2+(lon_1-lon_2)^2),
                    lat_1 = NULL,
                    lat_2 = NULL,
                    lon_1 = NULL,
                    lon_2 = NULL,
                    itemID_1 = NULL,
                    itemID_2 = NULL)]

  itemPairs[, ':=' (priceDiff = ifelse(is.na(priceDiff), 0, priceDiff),
                    priceMin = ifelse(is.na(priceMin), 0, priceMin),
                    priceMax = ifelse(is.na(priceMax), 0, priceMax),
                    titleStringDist = ifelse(is.na(titleStringDist), 0, titleStringDist),
                    titleStringDist2 = ifelse(is.na(titleStringDist2) | titleStringDist2 == Inf, 0, titl
}
```

```
createFeatures(itemPairsTest)
createFeatures(itemPairsTrain)
```

Train Model

```
maxTrees = 120
shrinkage = 0.07
gamma = 1
depth = 13
minChildWeight = 38
colSample = 0.4
subSample = 0.37
earlyStopRound = 4

modelVars = names(itemPairsTrain)[which(!(names(itemPairsTrain) %in% c("isDuplicate", "generationMethod

itemPairsTest = data.frame(itemPairsTest)
itemPairsTrain = data.frame(itemPairsTrain)
set.seed(0)

# Matrix
dtrain = xgb.DMatrix(as.matrix(itemPairsTrain[, modelVars]), label=itemPairsTrain$isDuplicate)
dtest = xgb.DMatrix(as.matrix(itemPairsTest[, modelVars]))

xgbResult = xgboost(params=list(max_depth=depth,
                                 eta=shrinkage,
                                 gamma=gamma,
                                 colsample_bytree=colSample,
                                 min_child_weight=minChildWeight),
                     data=dtrain,
                     nrounds=maxTrees,
                     objective="binary:logistic",
                     eval_metric="auc")
```

```
## [1]  train-auc:0.774599
## [2]  train-auc:0.800695
## [3]  train-auc:0.815681
## [4]  train-auc:0.825151
## [5]  train-auc:0.828282
## [6]  train-auc:0.830554
## [7]  train-auc:0.832529
## [8]  train-auc:0.833151
## [9]  train-auc:0.836119
## [10] train-auc:0.836991
## [11] train-auc:0.838799
## [12] train-auc:0.839131
## [13] train-auc:0.839337
## [14] train-auc:0.839335
## [15] train-auc:0.839199
## [16] train-auc:0.840443
## [17] train-auc:0.841063
## [18] train-auc:0.841456
```

```
## [19] train-auc:0.842496
## [20] train-auc:0.843048
## [21] train-auc:0.842974
## [22] train-auc:0.843084
## [23] train-auc:0.843764
## [24] train-auc:0.844491
## [25] train-auc:0.845001
## [26] train-auc:0.845064
## [27] train-auc:0.845312
## [28] train-auc:0.845977
## [29] train-auc:0.846380
## [30] train-auc:0.846833
## [31] train-auc:0.846915
## [32] train-auc:0.847285
## [33] train-auc:0.847981
## [34] train-auc:0.848559
## [35] train-auc:0.848982
## [36] train-auc:0.849348
## [37] train-auc:0.849797
## [38] train-auc:0.849941
## [39] train-auc:0.850262
## [40] train-auc:0.850563
## [41] train-auc:0.850861
## [42] train-auc:0.851460
## [43] train-auc:0.851589
## [44] train-auc:0.851909
## [45] train-auc:0.852157
## [46] train-auc:0.852406
## [47] train-auc:0.852841
## [48] train-auc:0.853145
## [49] train-auc:0.853516
## [50] train-auc:0.853884
## [51] train-auc:0.854265
## [52] train-auc:0.854550
## [53] train-auc:0.854847
## [54] train-auc:0.855197
## [55] train-auc:0.855509
## [56] train-auc:0.855727
## [57] train-auc:0.856129
## [58] train-auc:0.856385
## [59] train-auc:0.856681
## [60] train-auc:0.856902
## [61] train-auc:0.857013
## [62] train-auc:0.857242
## [63] train-auc:0.857572
## [64] train-auc:0.857853
## [65] train-auc:0.858085
## [66] train-auc:0.858367
## [67] train-auc:0.858640
## [68] train-auc:0.858901
## [69] train-auc:0.859248
## [70] train-auc:0.859483
## [71] train-auc:0.859659
## [72] train-auc:0.859972
```

```
## [73] train-auc:0.860175
## [74] train-auc:0.860443
## [75] train-auc:0.860691
## [76] train-auc:0.860912
## [77] train-auc:0.861132
## [78] train-auc:0.861499
## [79] train-auc:0.861653
## [80] train-auc:0.861819
## [81] train-auc:0.861968
## [82] train-auc:0.862246
## [83] train-auc:0.862363
## [84] train-auc:0.862591
## [85] train-auc:0.862793
## [86] train-auc:0.862978
## [87] train-auc:0.863098
## [88] train-auc:0.863433
## [89] train-auc:0.863663
## [90] train-auc:0.863890
## [91] train-auc:0.864104
## [92] train-auc:0.864275
## [93] train-auc:0.864398
## [94] train-auc:0.864571
## [95] train-auc:0.864736
## [96] train-auc:0.864857
## [97] train-auc:0.865047
## [98] train-auc:0.865183
## [99] train-auc:0.865373
## [100]    train-auc:0.865556
## [101]    train-auc:0.865685
## [102]    train-auc:0.865811
## [103]    train-auc:0.866007
## [104]    train-auc:0.866169
## [105]    train-auc:0.866280
## [106]    train-auc:0.866390
## [107]    train-auc:0.866529
## [108]    train-auc:0.866654
## [109]    train-auc:0.866778
## [110]    train-auc:0.866925
## [111]    train-auc:0.867014
## [112]    train-auc:0.867139
## [113]    train-auc:0.867207
## [114]    train-auc:0.867335
## [115]    train-auc:0.867501
## [116]    train-auc:0.867560
## [117]    train-auc:0.867716
## [118]    train-auc:0.867877
## [119]    train-auc:0.868063
## [120]    train-auc:0.868280
```

```
testPreds = predict(xgbResult, dtest)
```

Output

```r
submission = data.frame(id=itemPairsTest$id, probability=testPreds)
write.csv(submission, file="submission.csv",row.names=FALSE)
```

Result

```r
knitr::include_graphics("result.png")
```