Recommender system for movies in R

Ingrid Brizotti

The primary goal of this project is to build different recommender systems for movies of users that rated movies by Twitter account. Also, the idea is to compare the recommender systems using:

- a binary rating (watched or not) and the rating gave by the user (from 0 to 10)
- using similarity between users (user-based) and items (item-based)
- different approaches to calculating the similarity: Euclidean distance, cosine distance, Pearson correlation and Jaccard index.

Dataset: MovieTweetings (https://github.com/sidooms/MovieTweetings)

Approach: item-based and user-based Collaborative Filtering

Techniques used to measure similarity: Euclidean distance, cosine distance, Jaccard index and Pearson correlation

Package: recommenderlab calculates the similarity and predicts the rating using regression (more info: https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf)

Steps:

1) Prepare the data (check duplicity, missing, clean data, apply transformations, exploratory analysis)

2) Divide 70% to train and 30% test

3) Build the recommender system using binary variable

4) Compare different approaches for binary systems measuring the accuracy using confusion matrix and ROC curve for BINARY systems

5) Build the recommender system using rating

6) Compare different approaches for rating systems measuring accuracy using confusion matrix and ROC curve for RATING systems

7) Conclusions:

- What is better? Systems using binary or rating?
- What is more accurate: item-based or user-based?
- Identify the best approach to calculate similarity

1) Prepare the data

In this phase, the datasets were load in R and checked for duplications and missing values. Also, it was performed some descriptive statistics like checking rating distribution, how many movies per user, and how many ratings per movie.

The data set was filtered and just movies with more than 500 ratings stayed (this parameter was checked and compared on literature, [1]) and for binary systems it was kept 10 movies/user and for rating 90 movies/user because they demonstrated a better performance (please, see the results section on the final report document).

```r
########### Load packages ##############
library(data.table)
library(ggplot2)
library(recommenderlab)
```

```
## Loading required package: Matrix

## Loading required package: arules

##
## Attaching package: 'arules'

## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: proxy

##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry
```

```r
library(scales)

########### Load data ##############
#setwd("/Users/ingridbrizotti/Desktop/Ryerson/3.Data_Analytics_Capstone/MovieTweetings/latest")

# MOVIES #
#mov <- readLines("movies.dat")
#head(mov)
#mov <- gsub("::", "*", mov)
#movies <- read.table(text=mov, sep="*", header=FALSE, stringsAsFactor=TRUE, na.strings = "EMPTY",
#                      fileEncoding="UTF-8",fill = TRUE, quote = "")
#colnames(movies) <- c("movie_id","movie_title_year","genre")
#head(movies)

# RATINGS #
```

```r
#rat <- readLines("ratings.dat")
#head(rat)
#rat <- gsub("::", "*", rat)
#ratings <- read.table(text=rat, sep="*", header=FALSE, stringsAsFactor=TRUE, na.strings = "EMPTY",
#                      fileEncoding="UTF-8",fill = TRUE, quote = "")
#head(ratings)
#colnames(ratings) <- c("user_id","movie_id","rating","rating_timestamp")
#head(ratings)

load("~/Desktop/Ryerson/3.Data_Analytics_Capstone/MovieTweetings/latest/ratings.Rda")
load("~/Desktop/Ryerson/3.Data_Analytics_Capstone/MovieTweetings/latest/movies.Rda")

########### Check duplicity and missing ##############

# Movies #

# Transform in matrix (it was a factor)
movies<- as.data.frame.matrix(movies)
head(movies)
```

```
##   movie_id                              movie_title_year
## 1  0000008 Edison Kinetoscopic Record of a Sneeze (1894)
## 2  0000010           La sortie des usines Lumière (1895)
## 3  0000012                 The Arrival of a Train (1896)
## 4  0000091                    Le manoir du diable (1896)
## 5  0000417                 Le voyage dans la lune (1902)
## 6  0000439                The Great Train Robbery (1903)
##                       genre
## 1        Documentary|Short
## 2        Documentary|Short
## 3        Documentary|Short
## 4             Short|Horror
## 5 Short|Adventure|Fantasy
## 6       Short|Action|Crime
```

```r
str(movies)
```

```
## 'data.frame':    25810 obs. of  3 variables:
##  $ movie_id        : chr  "0000008" "0000010" "0000012" "0000091" ...
##  $ movie_title_year: chr  "Edison Kinetoscopic Record of a Sneeze (1894)" "La sortie des usines Lumi
##  $ genre           : chr  "Documentary|Short" "Documentary|Short" "Documentary|Short" "Short|Horror"
```

```r
# Tranform movie_id in numeric
movies2 <- data.frame(movies, movie_id_n = as.numeric(movies$movie_id))
```

```
## Warning in data.frame(movies, movie_id_n = as.numeric(movies$movie_id)):
## NAs introduzidos por coerção
```

```r
summary(movies2$movie_id_n)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       8  103000  492000 1236000 2094000 6014000       7
```

```r
# Check duplicity of variable movie_id
dupli_m <- movies2[duplicated(movies2$movie_id_n),]
nrow(dupli_m)
```

```
## [1] 6
```

```r
# remove duplicate observations
movies2 <- movies2[!duplicated(movies2$movie_id_n),]


# Check for NAs in movie_id (movie_id is the primary key)
missing <- movies2[is.na(movies2$movie_id_n),]
nrow(missing)
```

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

```r
# Ratings #
head(ratings)
```

```
##   user_id movie_id rating rating_timestamp
## 1       1    68646     10       1381620027
## 2       1   113277     10       1379466669
## 3       2   422720      8       1412178746
## 4       2   454876      8       1394818630
## 5       2   790636      7       1389963947
## 6       2   816711      8       1379963769
```

```r
# Check NA on user_id
missing <- ratings[is.na(ratings$user_id),]
nrow(missing)
```

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

```r
# Check NA on movie_id
missing <- ratings[is.na(ratings$movie_id),]
nrow(missing)
```

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

```r
dim(movies2)
```

```
## [1] 25804     4
```

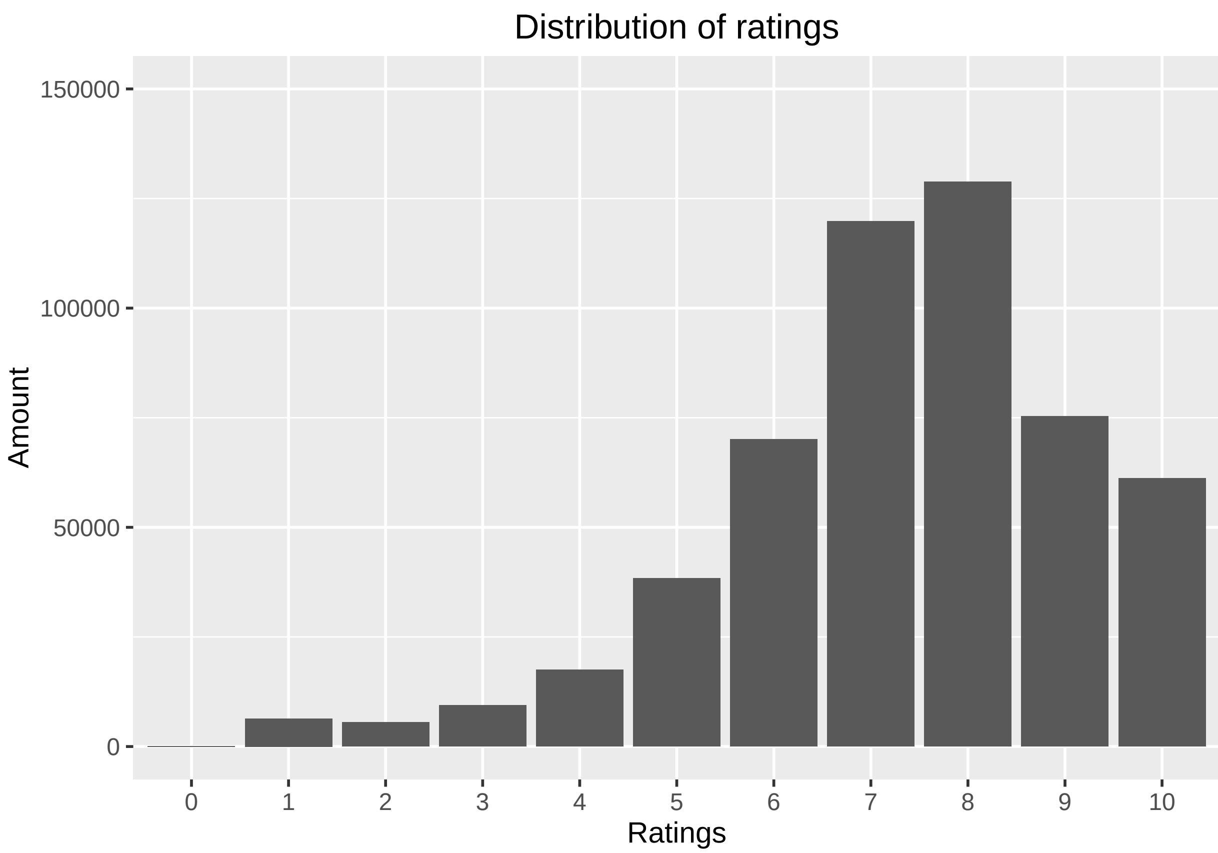
```r
dim(ratings)
```

```
## [1] 532608      4
```

```
########### Exploratory analysis ##############

# Analize the distribution of ratings
vector_ratings <- as.vector(ratings$rating)
unique(vector_ratings)
```

```
##  [1] 10  8  7  5  9  6  1  3  4  2  0
```

```
vector_ratings <- factor(vector_ratings)
qplot(vector_ratings, ylim = c(0,150000), xlab="Ratings", ylab="Amount") + ggtitle("Distribution of rat
```




```
# count the number of ratings per movie
t_m <- aggregate(cbind(count = rating) ~ movie_id,
                 data = ratings,
                 FUN = function(x){NROW(x)})

# merge this count on ratings data set
r <- merge(x=ratings, y=t_m ,by="movie_id", all.x=TRUE)


########### Filters ##############

# select movies with more than 500 ratings (checked on literature)
r <- r[r$count >= 500,]
```

```r
# count movies per user
t_m_u <- aggregate(cbind(count_movie = movie_id) ~ user_id,
                   data = r,
                   FUN = function(x){NROW(x)})
# merge
r2 <- merge(x=r, y=t_m_u ,by="user_id", all.x=TRUE)


# select users with more than 10 movies for BINARY systems
r_binary <- r2[r2$count_movie >= 10,]

# select users with more than 90 movies for RATING systems
r_rating <- r2[r2$count_movie >= 90,]


# delete columns that won't be used
r_binary <- subset(r_binary, , -c(rating_timestamp,count,count_movie,rating))
r_rating <- subset(r_rating, , -c(rating_timestamp,count,count_movie))

# convert it into a data table
r_binary <- data.table(r_binary)
r_rating <- data.table(r_rating)


######### Prepare the data for binary systems (watched 1, didn't watch 0) ########

# reshape (movies will be columns)
r_binary[, value := 1]
r_binary_wide <- reshape(data = r_binary,
                         direction = "wide",
                         idvar = "user_id",
                         timevar = "movie_id",
                         v.names = "value",
                         drop = NULL)

# keep only the columns containing ratings
# the user name will be the matrix row names, so we need to store them in the vector_users vector
vector_users <- r_binary_wide[, user_id]
r_binary_wide <- r_binary_wide[ ,user_id := NULL]

# have the column names equal to the item names
setnames(x = r_binary_wide,
         old = names(r_binary_wide),
         new = substring(names(r_binary_wide), 7))

# store the rating matrix within a recommenderlab object:
# 1) convert r_wide in a matrix
# 2) set the row names equal to the user names
matrix_binary_wide <- as.matrix(r_binary_wide)
rownames(matrix_binary_wide) <- vector_users
head(matrix_binary_wide[, 1:6])
```

```
##    454876 3079380 1398426 1091191 2294629 1454468
```

```
## 2       1       1       1       1       1       1
## 18     NA      NA      NA       1      NA      NA
## 26     NA      NA      NA      NA      NA      NA
## 27     NA       1      NA      NA      NA      NA
## 48     NA      NA      NA      NA      NA      NA
## 49     NA       1      NA      NA      NA      NA
```

```
# replace NA for zero
matrix_binary_wide[is.na(matrix_binary_wide)] <- 0
head(matrix_binary_wide[, 1:6])
```

```
##    454876 3079380 1398426 1091191 2294629 1454468
## 2       1       1       1       1       1       1
## 18      0       0       0       1       0       0
## 26      0       0       0       0       0       0
## 27      0       1       0       0       0       0
## 48      0       0       0       0       0       0
## 49      0       1       0       0       0       0
```

```
# 816711 2726560 3079380 1091191 2381249 1398426
# 2       1       1       1       1       1       1
# 18      0       0       0       1       0       0
# 26      1       0       0       0       0       0
# 38      0       0       0       0       0       0
# 48      0       0       0       0       0       0
# 49      0       0       1       0       0       0


# coercing matrix_wide into a binary rating matrix
binary_matrix <- as(matrix_binary_wide, "binaryRatingMatrix")
binary_matrix
```

```
## 4859 x 177 rating matrix of class 'binaryRatingMatrix' with 124501 ratings.
```

```
######## Prepare the data for rating systems #########
rating_wide <- reshape(data = r_rating,
                       direction = "wide",
                       idvar = "user_id",
                       timevar = "movie_id",
                       drop = NULL)

vector_users <- rating_wide[, user_id]
r_wide <- rating_wide[ ,user_id := NULL]

setnames(x = rating_wide,old = names(rating_wide),new = substring(names(rating_wide), 7))

matrix_rating_wide <- as.matrix(rating_wide)
rownames(matrix_rating_wide) <- vector_users
head(matrix_rating_wide[, 1:6])
```

```
##      .1800241 .3682448 .1210819 .2381249 .1951261 .1872194
## 665         6        8        6        4        6        5
```

```
## 1359        8       NA       NA        7        8       NA
## 1513        8       NA        5       NA        5        6
## 2056       NA        8       NA        6        8        8
## 2339        7       NA        5        7        3       NA
## 2834        7       NA        4       NA        6       10
```

```
ratings_matrix <- as(matrix_rating_wide, "realRatingMatrix")
ratings_matrix
```

```
## 62 x 177 rating matrix of class 'realRatingMatrix' with 6523 ratings.
```

```
# delete non necessary data
rm(r,r_binary,r_binary_wide,r_wide,r_rating,rating_wide,r2,ratings,t_m,t_m_u,missing,dupli_m)
```

2) Divide 70% to train and 30% test

In this step, the data set is divided in 70% for training and 30% for testing. Also, some parameters are defined:

- In order to measure performance binary equal 1 (watched the movie) is considered good and for rating systems greater or equal than 8 is good
- It is considered the 30 nearest neighbors to calculate the similarity [1] It will be recommended 30 movies, starting from 1 to 30 by 5

```
# split the data into the training and the test set
# Binary #
which_binary_train <- sample(x = c(TRUE, FALSE), size = nrow(binary_matrix),
                             replace = TRUE, prob = c(0.7, 0.3))
recc_binary_train <- binary_matrix[which_binary_train, ]
recc_binary_test <- binary_matrix[!which_binary_train, ]


# Rating #
which_rating_train <- sample(x = c(TRUE, FALSE), size = nrow(ratings_matrix),
                             replace = TRUE, prob = c(0.7, 0.3))
recc_rating_train <- ratings_matrix[which_rating_train, ]
recc_rating_test <- ratings_matrix[!which_rating_train, ]


# Defining some parameters
percentage_training <- 0.7
binary_threshold <- 1 # 1 is good
rating_threshold <- 8 # over or equal 8 is good
n_eval <- 1 # how many times we want to run the evaluation
number_neighbors <- 30 # nearest neighbors
n_recommendations <- c(1, 5, seq(10, 30, 5)) #number of recommendations
n_recommended <- 6 # 6 movies
```

3) Build arecommender system using binary variable

In this phase, is built a binary system on train data set using item-based collaborative filtering (IBCF) and Jaccard index. Also is extracted a data set with the first 6 recommendations per user.

```r
# Build a system using item-based and Jaccard index to measure the similarity
recc_binary_model <- Recommender(data = recc_binary_train,
                          method = "IBCF",
                          parameter = list(method = "Jaccard"))

# apply the recommender system on test set
recc_binary_predicted <- predict(object = recc_binary_model, newdata = recc_binary_test, n = n_recommen
recc_binary_predicted
```

```
## Recommendations as 'topNList' with n = 6 for 1484 users.
```

```r
# check configuration
class(recc_binary_predicted)
```

```
## [1] "topNList"
## attr(,"package")
## [1] "recommenderlab"
```

```r
# these are the recommendations for the first user:
recc_binary_predicted@items[[1]]
```

```
## [1]  94  50 124  59 125  81
```

```r
# these are the recommendations for the second user
recc_binary_predicted@items[[2]]
```

```
## [1] 114  77  78 118  60 105
```

```r
# define a matrix with the recommendations for each user:
recc_binary_matrix <- sapply(recc_binary_predicted@items, function(x){colnames(binary_matrix)[x]})
recc_binary_users <- as.data.table(recc_binary_matrix)
recc_binary_users_final <- t(recc_binary_users)
colnames(recc_binary_users_final) <- c("rec1","rec2","rec3","rec4","rec5","rec6")
head(recc_binary_users_final)
```

```
##     rec1      rec2      rec3      rec4      rec5      rec6
## 2   "2975590" "2379713" "3498820" "1663202" "2488496" "478970"
## 18  "1877832" "2872718" "2084970" "1981115" "2582802" "1631867"
## 48  "1431045" "993846"  "1800241" "2179136" "1895587" "2802144"
## 84  "1392190" "3460252" "2802144" "1877832" "3682448" "1843866"
## 102 "1408101" "478970"  "1663662" "3498820" "1392190" "2582802"
## 133 "1800241" "1300854" "816711"  "1981115" "1535108" "993846"
```

4) Compare different approaches for binary systems measuring the accuracy using confusion matrix and ROC curve for BINARY systems

In this step, is compared the performance of the following binary systems using confusion matrix, ROC curve and precision/recall graphs:

- Item-based Collaborative Filtering using Euclidean distance
- Item-based Collaborative Filtering using cosine distance
- Item-based Collaborative Filtering using Pearson correlation
- Item-based Collaborative Filtering using Jaccard index
- User-based Collaborative Filtering using Jaccard index
- User-based Collaborative Filtering using cosine distance

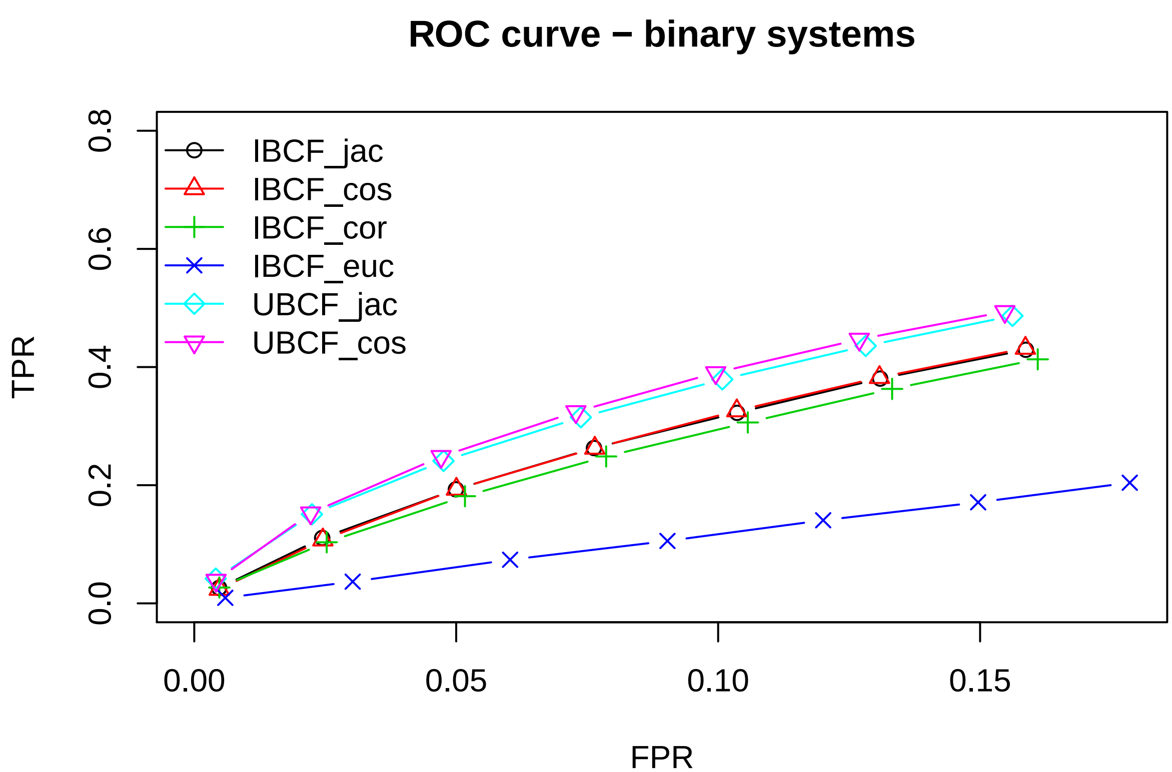
```
items_to_keep <- 10  # filter: 10 movies/user

# run evaluation on test data set
eval_binary_sets <- evaluationScheme(data = binary_matrix,
                              method = "split",
                              train = percentage_training,
                              given = items_to_keep,
                              goodRating =binary_threshold,
                              k = n_eval)
eval_binary_sets
```

```
## Evaluation scheme with 10 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.700
## Good ratings: >=1.000000
## Data set: 4859 x 177 rating matrix of class 'binaryRatingMatrix' with 124501 ratings.
```

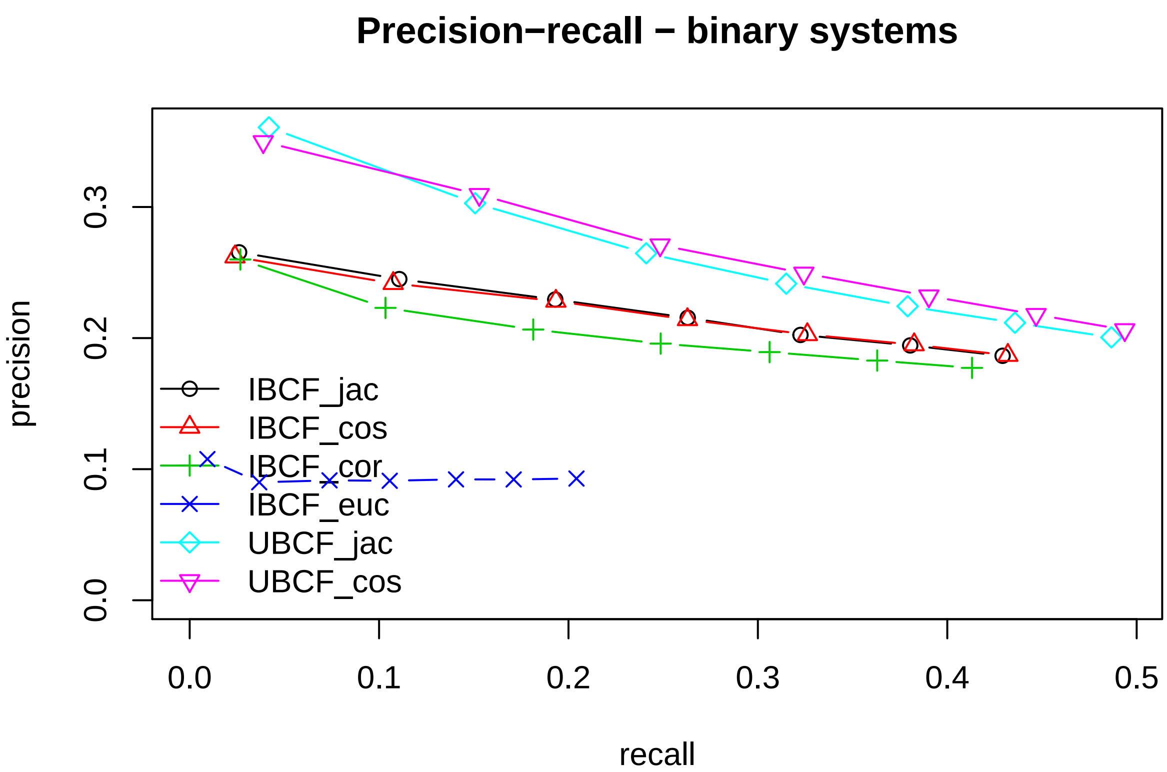
```
# list all the systems that will be compared
models_binary_evaluate <- list(
  IBCF_jac = list(name = "IBCF", param = list(method = "jaccard")),
  IBCF_cos = list(name = "IBCF", param = list(method = "cosine")),
  IBCF_cor = list(name = "IBCF", param = list(method = "pearson")),
  IBCF_euc = list(name = "IBCF", param = list(method = "euclidean")),

  UBCF_jac = list(name = "UBCF", param = list(method = "jaccard")),
  UBCF_cos = list(name = "UBCF", param = list(method = "cosine"))
)

list_binary_results <- evaluate(x = eval_binary_sets,
                         method = models_binary_evaluate,
                         n = n_recommendations)
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.436sec/0.643sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.519sec/0.587sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.243sec/0.511sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.713sec/0.489sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.001sec/18.051sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.002sec/16.477sec]
```

```r
# ROC curve
plot(list_binary_results, legend = "topleft",ylim = c(0, 0.80))
title("ROC curve - binary systems")
```




```r
# Precision/recall
plot(list_binary_results, "prec/rec", legend = "bottomleft")
title("Precision-recall - binary systems")
```




```r
# Confusion matrix
avg_binary_matrices <- lapply(list_binary_results, avg)


# Confusion matrix - IBCF cosine distance
head(avg_binary_matrices$IBCF_cos[,1:8])
```

```
##            TP         FP       FN       TN precision     recall        TPR
## 1  0.2620027  0.7379973 15.18038 150.8196 0.2620027 0.02394502 0.02394502
## 5  1.2085048  3.7914952 14.23388 147.7661 0.2417010 0.10746975 0.10746975
## 10 2.2818930  7.7181070 13.16049 143.8395 0.2281893 0.19335886 0.19335886
## 15 3.2119342 11.7880658 12.23045 139.7695 0.2141289 0.26282868 0.26282868
## 20 4.0541838 15.9458162 11.38820 135.6118 0.2027092 0.32612161 0.32612161
## 25 4.8744856 20.1255144 10.56790 131.4321 0.1949794 0.38251447 0.38251447
##            FPR
## 1  0.004768281
## 5  0.024549141
## 10 0.050039135
## 15 0.076456942
## 20 0.103558576
## 25 0.130795688
```

```r
# Confusion matrix - IBCF jaccard index
head(avg_binary_matrices$IBCF_jac[,1:8])
```

```
##            TP         FP       FN       TN precision     recall        TPR
## 1  0.2654321  0.7345679 15.17695 150.8230 0.2654321 0.02609642 0.02609642
## 5  1.2249657  3.7750343 14.21742 147.7826 0.2449931 0.11067075 0.11067075
```

```
## 10 2.2942387  7.7057613 13.14815 143.8519 0.2294239 0.19302185 0.19302185
## 15 3.2325103 11.7674897 12.20988 139.7901 0.2155007 0.26296268 0.26296268
## 20 4.0480110 15.9519890 11.39438 135.6056 0.2024005 0.32249769 0.32249769
## 25 4.8600823 20.1399177 10.58230 131.4177 0.1944033 0.38044380 0.38044380
##           FPR
## 1  0.00474707
## 5  0.02442681
## 10 0.04994309
## 15 0.07629907
## 20 0.10362005
## 25 0.13092636
```

```r
# Confusion matrix - IBCF euclidean distance
head(avg_binary_matrices$IBCF_euc[,1:8])
```

```
##           TP         FP       FN       TN  precision      recall
## 1  0.1076818  0.8923182 15.33471 150.6653 0.10768176 0.009372286
## 5  0.4499314  4.5500686 14.99246 147.0075 0.08998628 0.036735002
## 10 0.9142661  9.0829904 14.52812 142.4746 0.09146363 0.073840466
## 15 1.3641975 13.6186557 14.07819 137.9390 0.09114355 0.105634890
## 20 1.8374486 18.1056241 13.60494 133.4520 0.09221874 0.140631240
## 25 2.2894376 22.5706447 13.15295 128.9870 0.09215392 0.171069815
##            TPR         FPR
## 1  0.009372286 0.005923252
## 5  0.036735002 0.030238253
## 10 0.073840466 0.060290173
## 15 0.105634890 0.090317708
## 20 0.140631240 0.120046975
## 25 0.171069815 0.149631678
```

```r
# Confusion matrix - IBCF pearson correlation
head(avg_binary_matrices$IBCF_cor[,1:8])
```

```
##           TP         FP       FN       TN precision     recall        TPR
## 1  0.2599451  0.7400549 15.18244 150.8176 0.2599451 0.02677006 0.02677006
## 5  1.1152263  3.8847737 14.32716 147.6728 0.2230453 0.10340128 0.10340128
## 10 2.0651578  7.9348422 13.37723 143.6228 0.2065158 0.18140123 0.18140123
## 15 2.9368999 12.0631001 12.50549 139.4945 0.1957933 0.24869329 0.24869329
## 20 3.7860082 16.2139918 11.65638 135.3436 0.1893004 0.30620227 0.30620227
## 25 4.5713306 20.4286694 10.87106 131.1289 0.1828532 0.36301385 0.36301385
##            FPR
## 1  0.004787957
## 5  0.025280863
## 10 0.051671640
## 15 0.078621908
## 20 0.105659028
## 25 0.133213368
```

```r
# Confusion matrix - UBCF cosine distance
head(avg_binary_matrices$UBCF_cos[,1:8])
```

```
##           TP         FP         FN        TN precision     recall        TPR
```

```
## 1  0.3497942  0.6502058 15.092593 150.9074 0.3497942 0.03885343 0.03885343
## 5  1.5473251  3.4526749 13.895062 148.1049 0.3094650 0.15289579 0.15289579
## 10 2.7098765  7.2901235 12.732510 144.2675 0.2709877 0.24840376 0.24840376
## 15 3.7421125 11.2578875 11.700274 140.2997 0.2494742 0.32428861 0.32428861
## 20 4.6419753 15.3580247 10.800412 136.1996 0.2320988 0.39029102 0.39029102
## 25 5.4485597 19.5514403  9.993827 132.0062 0.2179424 0.44690602 0.44690602
##            FPR
## 1  0.004163549
## 5  0.022218749
## 10 0.047115353
## 15 0.072843644
## 20 0.099532989
## 25 0.126943057
```

```r
# Confusion matrix - UBCF jaccard index
head(avg_binary_matrices$UBCF_jac[,1:8])
```

```
##           TP         FP       FN       TN precision     recall        TPR
## 1  0.3607682  0.6392318 15.08162 150.9184 0.3607682 0.04184268 0.04184268
## 5  1.5144033  3.4855967 13.92798 148.0720 0.3028807 0.15075438 0.15075438
## 10 2.6467764  7.3532236 12.79561 144.2044 0.2646776 0.24107373 0.24107373
## 15 3.6227709 11.3772291 11.81962 140.1804 0.2415181 0.31499656 0.31499656
## 20 4.4855967 15.5144033 10.95679 136.0432 0.2242798 0.37910627 0.37910627
## 25 5.2928669 19.7071331 10.14952 131.8505 0.2117147 0.43575791 0.43575791
##            FPR
## 1  0.004098675
## 5  0.022469854
## 10 0.047591232
## 15 0.073796489
## 20 0.100802872
## 25 0.128171642
```

5) Build the recommender system using rating

In this phase, is built a rating system on train data set using item-based collaborative filtering (IBCF) and Jaccard index. Also is extracted a data set with the first 6 recommendations per user.

```r
# Build a system using item-based and Jaccard index to measure the similarity
recc_rating_model <- Recommender(data = recc_rating_train,
                                 method = "IBCF",
                                 parameter = list(method = "Jaccard"))

recc_rating_predicted <- predict(object = recc_rating_model, newdata = recc_rating_test, n = n_recommend
recc_rating_predicted
```

```
## Recommendations as 'topNList' with n = 6 for 21 users.
```

```r
class(recc_rating_predicted)
```

```
## [1] "topNList"
## attr(,"package")
## [1] "recommenderlab"
```

```r
# these are the recommendations for the first user:
recc_rating_predicted@items[[1]]
```

```
## [1] 177  12 126  86 150  83
```

```r
# second user
recc_rating_predicted@items[[2]]
```

```
## [1]  44 174 170  57 124  93
```

```r
# define a matrix with the recommendations for each user:
recc_rating_matrix <- sapply(recc_rating_predicted@items, function(x){colnames(ratings_matrix)[x]})
recc_rating_users <- as.data.table(recc_rating_matrix)
recc_rating_users_final <- t(recc_rating_users)
colnames(recc_rating_users_final) <- c("rec1","rec2","rec3","rec4","rec5","rec6")
head(recc_rating_users_final)
```

```
##       rec1       rec2       rec3       rec4       rec5       rec6
## 2056  ".1375666" ".2713180" ".1411250" ".1650554" ".790724"  ".455944"
## 2910  ".1386697" ".111161"  ".2488496" ".1179933" ".2395427" ".3498820"
## 8543  ".1895587" ".1392170" ".2948356" ".1853728" ".3682448" ".1045658"
## 8755  ".2080374" ".2096673" ".1398426" ".111161"  ".2357129" ".1392170"
## 10856 ".1631867" ".1453405" ".1980209" ".455944"  ".2080374" ".2140373"
## 11087 ".3460252" ".1853728" ".3076658" ".1895587" ".2872718" ".3682448"
```

6) Compare different approaches for rating systems measuring the accuracy using confusion matrix and ROC curve for RATING systems

In this step, is compared the performance of the following binary systems using confusion matrix, ROC curve and precision/recall graphs:

- Item-based Collaborative Filtering using Euclidean distance
- Item-based Collaborative Filtering using cosine distance
- Item-based Collaborative Filtering using Pearson correlation
- Item-based Collaborative Filtering using Jaccard index
- User-based Collaborative Filtering using Jaccard index
- User-based Collaborative Filtering using cosine distance
- User-based Collaborative Filtering using Pearson correlation
- User-based Collaborative Filtering using Euclidean distance

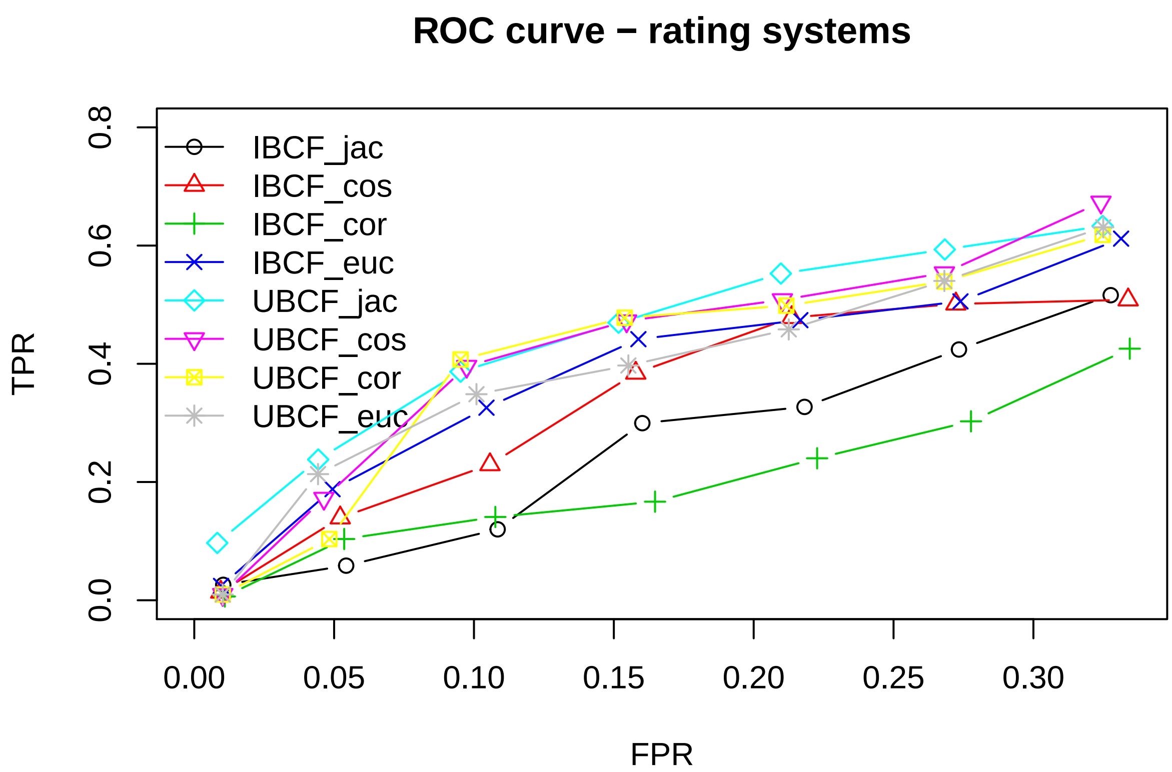
```r
# run evaluation on test data set
items_to_keep <- 90
eval_rating_sets <- evaluationScheme(data = ratings_matrix,
                                     method = "split",
                                     train = percentage_training,
                                     given = items_to_keep,
                                     goodRating =rating_threshold,
                                     k = n_eval)
eval_rating_sets
```

```
## Evaluation scheme with 90 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.700
## Good ratings: >=8.000000
## Data set: 62 x 177 rating matrix of class 'realRatingMatrix' with 6523 ratings.
```

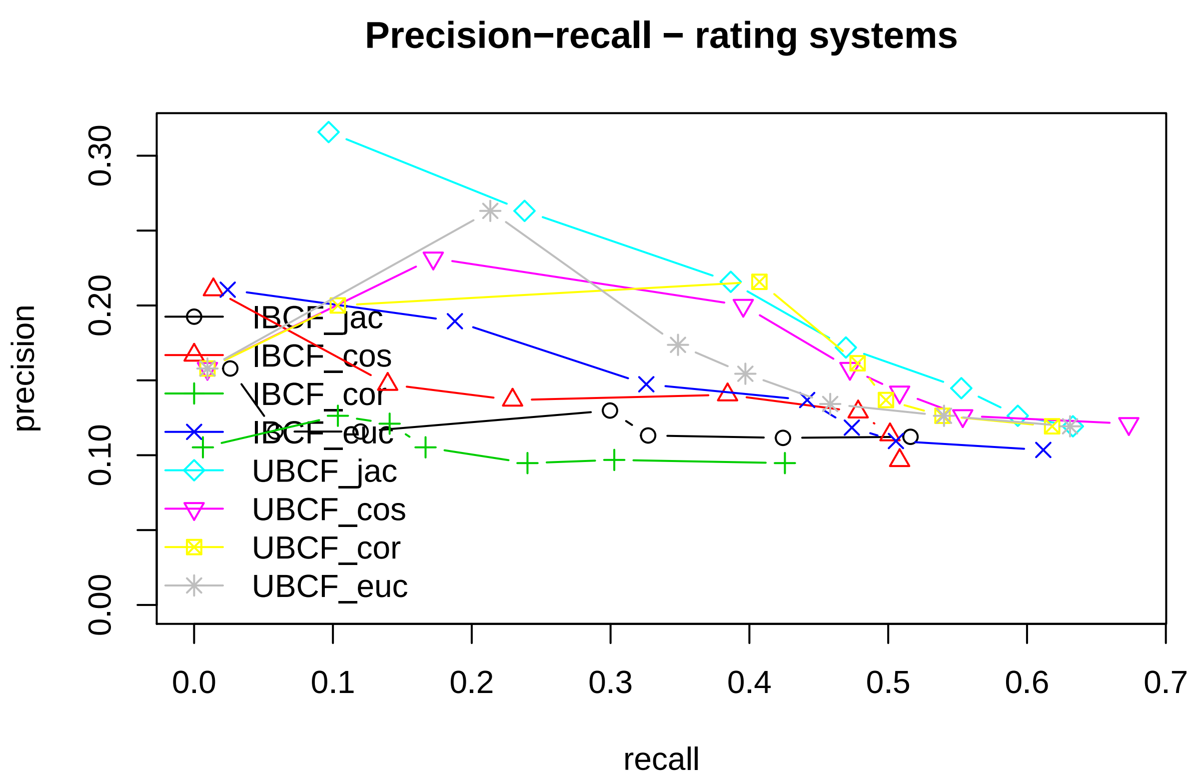
```r
models_rating_evaluate <- list(
  IBCF_jac = list(name = "IBCF", param = list(method = "jaccard")),
  IBCF_cos = list(name = "IBCF", param = list(method = "cosine")),
  IBCF_cor = list(name = "IBCF", param = list(method = "pearson")),
  IBCF_euc = list(name = "IBCF", param = list(method = "euclidean")),

  UBCF_jac = list(name = "UBCF", param = list(method = "jaccard")),
  UBCF_cos = list(name = "UBCF", param = list(method = "cosine")),
  UBCF_cor = list(name = "UBCF", param = list(method = "pearson")),
  UBCF_euc = list(name = "UBCF", param = list(method = "euclidean"))

)

list_rating_results <- evaluate(x = eval_rating_sets,
                                method = models_rating_evaluate,
                                n = n_recommendations)
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.092sec/0.021sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.094sec/0.019sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.15sec/0.019sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.09sec/0.018sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.003sec/0.043sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.003sec/0.04sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.003sec/0.038sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.003sec/0.036sec]
```

```r
plot(list_rating_results, legend = "topleft",ylim = c(0, 0.80))
title("ROC curve - rating systems")
```




```r
plot(list_rating_results, "prec/rec", legend = "bottomleft")
title("Precision-recall - rating systems")
```



```r
# Confusion matrix
avg_rating_matrices <- lapply(list_rating_results, avg)


# Confusion matrix - IBCF cosine distance
head(avg_rating_matrices$IBCF_cos[,1:8])
```

```
##           TP         FP       FN       TN precision     recall        TPR
## 1  0.2105263  0.7894737 5.947368 80.05263 0.2105263 0.01389587 0.01389587
## 5  0.7368421  4.2631579 5.421053 76.57895 0.1473684 0.13944752 0.13944752
## 10 1.3684211  8.6315789 4.789474 72.21053 0.1368421 0.22942573 0.22942573
## 15 2.1052632 12.8947368 4.052632 67.94737 0.1403509 0.38428490 0.38428490
## 20 2.5789474 17.4210526 3.578947 63.42105 0.1289474 0.47834808 0.47834808
## 25 2.8421053 22.1578947 3.315789 58.68421 0.1136842 0.50129442 0.50129442
##            FPR
## 1  0.009473128
## 5  0.052185566
## 10 0.105718550
## 15 0.157841277
## 20 0.213622179
## 25 0.272290148
```

```r
# Confusion matrix -  IBCF jaccard index
head(avg_rating_matrices$IBCF_jac[,1:8])
```

```
##           TP         FP       FN       TN precision     recall        TPR
## 1  0.1578947  0.8421053 6.000000 80.00000 0.1578947 0.02606000 0.02606000
## 5  0.5789474  4.4210526 5.578947 76.42105 0.1157895 0.05850501 0.05850501
## 10 1.1578947  8.8421053 5.000000 72.00000 0.1157895 0.11994212 0.11994212
## 15 1.9473684 13.0526316 4.210526 67.78947 0.1298246 0.29958911 0.29958911
## 20 2.2631579 17.7368421 3.894737 63.10526 0.1131579 0.32706523 0.32706523
## 25 2.7894737 22.2105263 3.368421 58.63158 0.1115789 0.42420286 0.42420286
##           FPR
## 1  0.01036122
## 5  0.05431290
## 10 0.10845737
## 15 0.16018922
## 20 0.21817815
## 25 0.27339470
```

```r
# Confusion matrix -  IBCF euclidean distance
head(avg_rating_matrices$IBCF_euc[,1:8])
```

```
##           TP         FP       FN       TN precision     recall        TPR
## 1  0.2105263  0.7894737 5.947368 80.05263 0.2105263 0.02421154 0.02421154
## 5  0.9473684  4.0526316 5.210526 76.78947 0.1894737 0.18786361 0.18786361
## 10 1.4736842  8.5263158 4.684211 72.31579 0.1473684 0.32576093 0.32576093
## 15 2.0526316 12.9473684 4.105263 67.89474 0.1368421 0.44163784 0.44163784
## 20 2.3684211 17.6315789 3.789474 63.21053 0.1184211 0.47379107 0.47379107
## 25 2.7368421 22.2631579 3.421053 58.57895 0.1094737 0.50556094 0.50556094
##           FPR
## 1  0.00958115
```

```
## 5  0.04943876
## 10 0.10447308
## 15 0.15878245
## 20 0.21669001
## 25 0.27392257
```

```r
# Confusion matrix -  IBCF pearson correlation
head(avg_rating_matrices$IBCF_cor[,1:8])
```

```
##           TP         FP       FN       TN  precision      recall
## 1  0.1052632  0.8947368 6.052632 79.94737 0.10526316 0.006354394
## 5  0.6315789  4.3684211 5.526316 76.47368 0.12631579 0.103583599
## 10 1.2105263  8.7894737 4.947368 72.05263 0.12105263 0.140874013
## 15 1.5789474 13.4210526 4.578947 67.42105 0.10526316 0.166737692
## 20 1.8947368 18.1052632 4.263158 62.73684 0.09473684 0.240157373
## 25 2.4210526 22.5789474 3.736842 58.26316 0.09684211 0.302671339
##            TPR        FPR
## 1  0.006354394 0.01093625
## 5  0.103583599 0.05360879
## 10 0.140874013 0.10764237
## 15 0.166737692 0.16473298
## 20 0.240157373 0.22267673
## 25 0.302671339 0.27769028
```

```r
# Confusion matrix -  UBCF cosine distance
head(avg_rating_matrices$UBCF_cos[,1:8])
```

```
##           TP         FP       FN       TN precision      recall
## 1  0.1578947  0.8421053 6.000000 80.00000 0.1578947 0.009622367
## 5  1.1578947  3.8421053 5.000000 77.00000 0.2315789 0.172287562
## 10 2.0000000  8.0000000 4.157895 72.84211 0.2000000 0.395599109
## 15 2.3684211 12.6315789 3.789474 68.21053 0.1578947 0.472410460
## 20 2.8421053 17.1578947 3.315789 63.68421 0.1421053 0.508226895
## 25 3.1578947 21.8421053 3.000000 59.00000 0.1263158 0.553687244
##            TPR        FPR
## 1  0.009622367 0.01018437
## 5  0.172287562 0.04635480
## 10 0.395599109 0.09738076
## 15 0.472410460 0.15457149
## 20 0.508226895 0.21026617
## 25 0.553687244 0.26819183
```

```r
# Confusion matrix -  UBCF jaccard index
head(avg_rating_matrices$UBCF_jac[,1:8])
```

```
##           TP         FP       FN       TN precision     recall        TPR
## 1  0.3157895  0.6842105 5.842105 80.15789 0.3157895 0.09692395 0.09692395
## 5  1.3157895  3.6842105 4.842105 77.15789 0.2631579 0.23805554 0.23805554
## 10 2.1578947  7.8421053 4.000000 73.00000 0.2157895 0.38659243 0.38659243
## 15 2.5789474 12.4210526 3.578947 68.42105 0.1719298 0.46952425 0.46952425
## 20 2.8947368 17.1052632 3.263158 63.73684 0.1447368 0.55268301 0.55268301
## 25 3.1578947 21.8421053 3.000000 59.00000 0.1263158 0.59335113 0.59335113
```

```
##             FPR
## 1  0.008232195
## 5  0.044324703
## 10 0.095203872
## 15 0.151698630
## 20 0.209718358
## 25 0.268335256
```

```r
# Confusion matrix -  UBCF pearson correlation
head(avg_rating_matrices$UBCF_cor[,1:8])
```

```
##           TP         FP       FN       TN precision      recall
## 1  0.1578947  0.8421053 6.000000 80.00000 0.1578947 0.009622367
## 5  1.0000000  4.0000000 5.157895 76.84211 0.2000000 0.103531726
## 10 2.1578947  7.8421053 4.000000 73.00000 0.2157895 0.407232537
## 15 2.4210526 12.5789474 3.736842 68.26316 0.1614035 0.477966015
## 20 2.7368421 17.2631579 3.421053 63.57895 0.1368421 0.498397835
## 25 3.1578947 21.8421053 3.000000 59.00000 0.1263158 0.539098075
##             TPR        FPR
## 1  0.009622367 0.01018437
## 5  0.103531726 0.04828565
## 10 0.407232537 0.09519551
## 15 0.477966015 0.15388796
## 20 0.498397835 0.21166093
## 25 0.539098075 0.26814601
```

```r
# Confusion matrix -  UBCF euclidean distance
head(avg_rating_matrices$UBCF_euc[,1:8])
```

```
##           TP         FP       FN       TN precision      recall
## 1  0.1578947  0.8421053 6.000000 80.00000 0.1578947 0.009622367
## 5  1.3157895  3.6842105 4.842105 77.15789 0.2631579 0.213364178
## 10 1.7368421  8.2631579 4.421053 72.57895 0.1736842 0.348583579
## 15 2.3157895 12.6842105 3.842105 68.15789 0.1543860 0.397101818
## 20 2.6842105 17.3157895 3.473684 63.52632 0.1342105 0.458219912
## 25 3.1578947 21.8421053 3.000000 59.00000 0.1263158 0.540198572
##             TPR        FPR
## 1  0.009622367 0.01018437
## 5  0.213364178 0.04425306
## 10 0.348583579 0.10088770
## 15 0.397101818 0.15519840
## 20 0.458219912 0.21253830
## 25 0.540198572 0.26816283
```

7) Conclusions:

As we can see below, rating systems are more accurate then binary ones, and user-based are superior than item-based. Also, Jaccard index and cosine distance are a better approach to calculate the similarity.

```r
library(rafalib)
mypar(1,2)
```

```r
plot(list_binary_results, legend = "topleft",ylim = c(0, 0.80))
title("ROC curve - binary systems")

plot(list_rating_results, legend = "topleft",ylim = c(0, 0.80))
title("ROC curve - rating systems")
```