MRS-Testing-git

shahin

11/03/2020

## Capstone Project-Movie Recommendation Systems

## Establish connection between R Project with github Repo

if(!require(tidyverse))   
 install.packages("tidyverse", repos = "http://cran.us.r-project.org") #[it loads ggplot2, tibble, tidyr, readr, purrr, and dplyr packages]

## Loading required package: tidyverse

## -- Attaching packages ------------------------------------------------------------------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1 v purrr 0.3.3  
## v tibble 3.0.4 v dplyr 1.0.2  
## v tidyr 1.1.0 v stringr 1.4.0  
## v readr 1.4.0 v forcats 0.5.0

## -- Conflicts ---------------------------------------------------------------------------------------------------------- tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

if(!require(caret))   
 install.packages("caret", repos = "http://cran.us.r-project.org")

## Loading required package: caret

## Loading required package: lattice

##   
## Attaching package: 'caret'

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

if(!require(data.table))   
 install.packages("data.table", repos = "http://cran.us.r-project.org")

## Loading required package: 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

if(!require(knitr))   
 install.packages("knitr", repos = "http://cran.us.r-project.org")

## Loading required package: knitr

if(!require(recommenderlab))   
 install.packages("recommenderlab", repos = "http://cran.us.r-project.org")

## Loading required package: recommenderlab

## Loading required package: Matrix

##   
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack

## Loading required package: arules

##   
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':  
##   
## recode

## 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

## Registered S3 methods overwritten by 'registry':  
## method from   
## print.registry\_field proxy  
## print.registry\_entry proxy

##   
## Attaching package: 'recommenderlab'

## The following objects are masked from 'package:caret':  
##   
## MAE, RMSE

if(!require(reshape2))   
 install.packages("reshape2", repos = "http://cran.us.r-project.org")

## Loading required package: reshape2

##   
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':  
##   
## dcast, melt

## The following object is masked from 'package:tidyr':  
##   
## smiths

#if(!require(dplyr))   
# install.packages("dplyr", repos = "http://cran.us.r-project.org")  
#if(!require(ggplot2))   
# install.packages("ggplot2", repos = "http://cran.us.r-project.org")  
#library(dplyr)  
#library(ggplot2)  
#library(knitr)  
#install.packages("recommenderlab")  
#library(recommenderlab)  
#install.packages("reshape2") # may already be installed  
#library(reshape2)  
#library (readr)

## Project overview:

The goal of this project is to give a better understanding of User Based Collaborative Filter (UBCF) and Item Based Collaborative Filter (IBCF) models for hybrid recommender systems by answering the following question:

With what level of performance can collaborative filtering using UBCF and IBCF models produce movie recommendations based on movies and user’s ratings data?

## Datasets

The dataset I choose for this project is from GroupLens research lab in the University of Minnesota and available in the MovieLens website which contains four files such as movies.csv, ratings.csv, links.csv and tags.csv. To build a recommendar system I used only movies.csv and ratings.csv data files.

movies\_url="https://raw.githubusercontent.com/sahmed07/MRS-Testing/main/movies.csv"  
movies<-read\_csv(url(movies\_url))

##   
## -- Column specification ------------------------------------------------------------------------------------------------------------------------  
## cols(  
## movieId = col\_double(),  
## title = col\_character(),  
## genres = col\_character()  
## )

movies <- as\_tibble(movies)  
movies

## # A tibble: 9,742 x 3  
## movieId title genres   
## <dbl> <chr> <chr>   
## 1 1 Toy Story (1995) Adventure|Animation|Children|Comed~  
## 2 2 Jumanji (1995) Adventure|Children|Fantasy   
## 3 3 Grumpier Old Men (1995) Comedy|Romance   
## 4 4 Waiting to Exhale (1995) Comedy|Drama|Romance   
## 5 5 Father of the Bride Part II~ Comedy   
## 6 6 Heat (1995) Action|Crime|Thriller   
## 7 7 Sabrina (1995) Comedy|Romance   
## 8 8 Tom and Huck (1995) Adventure|Children   
## 9 9 Sudden Death (1995) Action   
## 10 10 GoldenEye (1995) Action|Adventure|Thriller   
## # ... with 9,732 more rows

#Exploring the movies table and variables:

str(movies)

## tibble [9,742 x 3] (S3: tbl\_df/tbl/data.frame)  
## $ movieId: num [1:9742] 1 2 3 4 5 6 7 8 9 10 ...  
## $ title : chr [1:9742] "Toy Story (1995)" "Jumanji (1995)" "Grumpier Old Men (1995)" "Waiting to Exhale (1995)" ...  
## $ genres : chr [1:9742] "Adventure|Animation|Children|Comedy|Fantasy" "Adventure|Children|Fantasy" "Comedy|Romance" "Comedy|Drama|Romance" ...  
## - attr(\*, "spec")=  
## .. cols(  
## .. movieId = col\_double(),  
## .. title = col\_character(),  
## .. genres = col\_character()  
## .. )

glimpse(movies)

## Rows: 9,742  
## Columns: 3  
## $ movieId <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...  
## $ title <chr> "Toy Story (1995)", "Jumanji (1995)", "Grumpier Old Me...  
## $ genres <chr> "Adventure|Animation|Children|Comedy|Fantasy", "Advent...

summary(movies)

## movieId title genres   
## Min. : 1 Length:9742 Length:9742   
## 1st Qu.: 3248 Class :character Class :character   
## Median : 7300 Mode :character Mode :character   
## Mean : 42200   
## 3rd Qu.: 76232   
## Max. :193609

ratings\_url="https://raw.githubusercontent.com/sahmed07/MRS-Testing/main/ratings.csv"  
ratings<-read\_csv(url(ratings\_url))

##   
## -- Column specification ------------------------------------------------------------------------------------------------------------------------  
## cols(  
## userId = col\_double(),  
## movieId = col\_double(),  
## rating = col\_double(),  
## timestamp = col\_double()  
## )

#remove the timestamp as it may not be required for this project  
ratings <- subset(ratings, select = -c(timestamp) )  
ratings <- as\_tibble(ratings)  
ratings

## # A tibble: 100,836 x 3  
## userId movieId rating  
## <dbl> <dbl> <dbl>  
## 1 1 1 4  
## 2 1 3 4  
## 3 1 6 4  
## 4 1 47 5  
## 5 1 50 5  
## 6 1 70 3  
## 7 1 101 5  
## 8 1 110 4  
## 9 1 151 5  
## 10 1 157 5  
## # ... with 100,826 more rows

#Exploring the ratings table and variables:

str(ratings)

## tibble [100,836 x 3] (S3: tbl\_df/tbl/data.frame)  
## $ userId : num [1:100836] 1 1 1 1 1 1 1 1 1 1 ...  
## $ movieId: num [1:100836] 1 3 6 47 50 70 101 110 151 157 ...  
## $ rating : num [1:100836] 4 4 4 5 5 3 5 4 5 5 ...

glimpse(ratings)

## Rows: 100,836  
## Columns: 3  
## $ userId <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...  
## $ movieId <dbl> 1, 3, 6, 47, 50, 70, 101, 110, 151, 157, 163, 216, 223...  
## $ rating <dbl> 4, 4, 4, 5, 5, 3, 5, 4, 5, 5, 5, 5, 3, 5, 4, 5, 3, 3, ...

summary(ratings)

## userId movieId rating   
## Min. : 1.0 Min. : 1 Min. :0.500   
## 1st Qu.:177.0 1st Qu.: 1199 1st Qu.:3.000   
## Median :325.0 Median : 2991 Median :3.500   
## Mean :326.1 Mean : 19435 Mean :3.502   
## 3rd Qu.:477.0 3rd Qu.: 8122 3rd Qu.:4.000   
## Max. :610.0 Max. :193609 Max. :5.000

## Data Exploration and pre-processing

# most popular movie genres

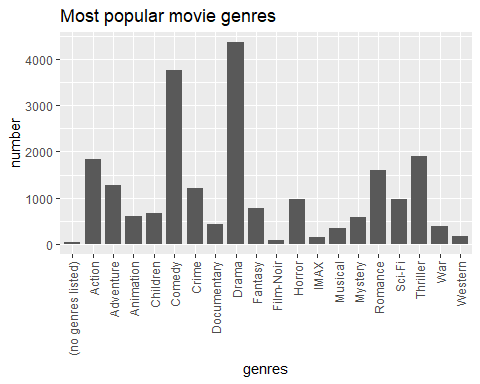
genres\_df <- movies %>%  
 separate\_rows(genres, sep = "\\|") %>%  
 group\_by(genres) %>%  
 summarise(number = n()) %>%  
 arrange(desc(number))

## `summarise()` ungrouping output (override with `.groups` argument)

genres\_df

## # A tibble: 20 x 2  
## genres number  
## <chr> <int>  
## 1 Drama 4361  
## 2 Comedy 3756  
## 3 Thriller 1894  
## 4 Action 1828  
## 5 Romance 1596  
## 6 Adventure 1263  
## 7 Crime 1199  
## 8 Sci-Fi 980  
## 9 Horror 978  
## 10 Fantasy 779  
## 11 Children 664  
## 12 Animation 611  
## 13 Mystery 573  
## 14 Documentary 440  
## 15 War 382  
## 16 Musical 334  
## 17 Western 167  
## 18 IMAX 158  
## 19 Film-Noir 87  
## 20 (no genres listed) 34

ggplot(data=genres\_df, aes(x=genres, y=number)) +  
 geom\_bar(stat="identity", width=.8)+  
 theme(axis.text.x=element\_text(angle=90,hjust=1,vjust=0.5))+  
 ggtitle("Most popular movie genres")

 # Based on Ratings which is the best/top Movie?

# average rating for a movie  
avg\_rating <- ratings %>%  
 inner\_join(movies, by = "movieId") %>%  
 na.omit() %>%  
 select(title, rating) %>%  
 group\_by(title, rating) %>%  
 summarise(count = n(), mean = mean(rating), min = min(rating), max = max(rating)) %>%  
 ungroup() %>%  
 arrange(desc(mean))

## `summarise()` regrouping output by 'title' (override with `.groups` argument)

avg\_rating

## # A tibble: 30,413 x 6  
## title rating count mean min max  
## <chr> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 'burbs, The (1989) 5 2 5 5 5  
## 2 'Salem's Lot (2004) 5 1 5 5 5  
## 3 'Til There Was You (1997) 5 1 5 5 5  
## 4 (500) Days of Summer (2009) 5 5 5 5 5  
## 5 [REC] (2007) 5 3 5 5 5  
## 6 ¡Three Amigos! (1986) 5 2 5 5 5  
## 7 10 Cloverfield Lane (2016) 5 1 5 5 5  
## 8 10 Things I Hate About You (1999) 5 8 5 5 5  
## 9 101 Dalmatians (1996) 5 3 5 5 5  
## 10 101 Dalmatians (One Hundred and One Dalm~ 5 1 5 5 5  
## # ... with 30,403 more rows

# using IMDB weight rating function determine top rated movie

weighted\_rating <- function(R, v, m, C) {  
 return (v/(v+m))\*R + (m/(v+m))\*C  
}  
  
avg\_rating <- avg\_rating %>%  
 mutate(wr = weighted\_rating(mean, count, 500, mean(mean))) %>%  
 arrange(desc(wr)) %>%  
 select(title, rating, count, mean, wr)  
  
avg\_rating

## # A tibble: 30,413 x 5  
## title rating count mean wr  
## <chr> <dbl> <int> <dbl> <dbl>  
## 1 Shawshank Redemption, The (1994) 5 153 5 0.234  
## 2 Pulp Fiction (1994) 5 123 5 0.197  
## 3 Forrest Gump (1994) 5 116 5 0.188  
## 4 Matrix, The (1999) 5 109 5 0.179  
## 5 Star Wars: Episode IV - A New Hope (1977) 5 104 5 0.172  
## 6 Jurassic Park (1993) 4 97 4 0.162  
## 7 Silence of the Lambs, The (1991) 4 97 4 0.162  
## 8 Forrest Gump (1994) 4 94 4 0.158  
## 9 Schindler's List (1993) 5 92 5 0.155  
## 10 Silence of the Lambs, The (1991) 5 92 5 0.155  
## # ... with 30,403 more rows

# Split genre and create a search matrix

movie\_genre <- as.data.frame(movies$genres, stringsAsFactors = FALSE)  
movie\_genre\_2 <- as.data.frame(tstrsplit(movie\_genre[,1], "[|]", type.convert = TRUE),stringsAsFactors = FALSE)  
  
colnames(movie\_genre\_2) <- c(1:10)  
  
genre\_list <- c("Action", "Adventure", "Animation", "Children",   
 "Comedy", "Crime","Documentary", "Drama", "Fantasy",  
 "Film-Noir", "Horror", "Musical", "Mystery","Romance",  
 "Sci-Fi", "Thriller", "War", "Western")  
  
genre\_matx <- matrix(0,9743,18)  
genre\_matx[1,] <- genre\_list  
colnames(genre\_matx) <- genre\_list  
  
for (index in 1:nrow(movie\_genre\_2)){  
 for(col in 1:ncol(movie\_genre\_2)){  
 gen\_col = which(genre\_matx[1,] == movie\_genre\_2[index,col])  
 genre\_matx[index+1,gen\_col] <- 1  
 }  
}  
  
genre\_matx\_2 <- as.data.frame(genre\_matx[-1,], stringsAsFactors=FALSE)  
  
for (col in 1:ncol(genre\_matx\_2)) {  
 genre\_matx\_2[,col] <- as.integer(genre\_matx\_2[,col])  
}  
  
head(genre\_matx\_2)

## Action Adventure Animation Children Comedy Crime Documentary Drama  
## 1 0 1 1 1 1 0 0 0  
## 2 0 1 0 1 0 0 0 0  
## 3 0 0 0 0 1 0 0 0  
## 4 0 0 0 0 1 0 0 1  
## 5 0 0 0 0 1 0 0 0  
## 6 1 0 0 0 0 1 0 0  
## Fantasy Film-Noir Horror Musical Mystery Romance Sci-Fi Thriller War  
## 1 1 0 0 0 0 0 0 0 0  
## 2 1 0 0 0 0 0 0 0 0  
## 3 0 0 0 0 0 1 0 0 0  
## 4 0 0 0 0 0 1 0 0 0  
## 5 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 1 0  
## Western  
## 1 0  
## 2 0  
## 3 0  
## 4 0  
## 5 0  
## 6 0

# define a search matrix

search\_matrix <- cbind(movies[,1:2], genre\_matx\_2)  
head(search\_matrix)

## movieId title Action Adventure Animation  
## 1 1 Toy Story (1995) 0 1 1  
## 2 2 Jumanji (1995) 0 1 0  
## 3 3 Grumpier Old Men (1995) 0 0 0  
## 4 4 Waiting to Exhale (1995) 0 0 0  
## 5 5 Father of the Bride Part II (1995) 0 0 0  
## 6 6 Heat (1995) 1 0 0  
## Children Comedy Crime Documentary Drama Fantasy Film-Noir Horror Musical  
## 1 1 1 0 0 0 1 0 0 0  
## 2 1 0 0 0 0 1 0 0 0  
## 3 0 1 0 0 0 0 0 0 0  
## 4 0 1 0 0 1 0 0 0 0  
## 5 0 1 0 0 0 0 0 0 0  
## 6 0 0 1 0 0 0 0 0 0  
## Mystery Romance Sci-Fi Thriller War Western  
## 1 0 0 0 0 0 0  
## 2 0 0 0 0 0 0  
## 3 0 1 0 0 0 0  
## 4 0 1 0 0 0 0  
## 5 0 0 0 0 0 0  
## 6 0 0 0 1 0 0

# Number of ratings and count of each ratings

vector\_ratings <- as.vector(ratings$rating)  
sort(unique(vector\_ratings))

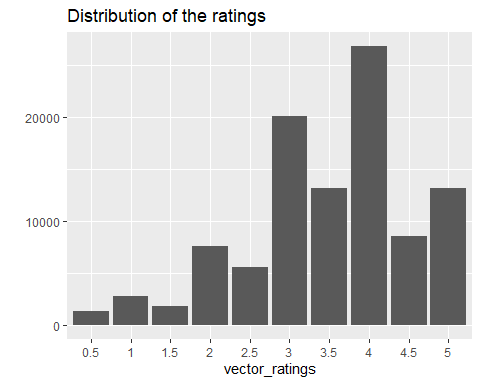
## [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

table\_ratings <- table(vector\_ratings)  
table\_ratings

## vector\_ratings  
## 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5   
## 1370 2811 1791 7551 5550 20047 13136 26818 8551 13211

# Visulization of ratinga count

vector\_ratings <- factor(vector\_ratings)  
qplot(vector\_ratings) +   
 ggtitle("Distribution of the ratings")

 #converting rating matrix into a sparse matrix of class type *realRatingMatrix*

#Create ratings matrix. Rows = userId, Columns = movieId  
ratingmat <- dcast(ratings, userId~movieId, value.var = "rating", na.rm=FALSE)  
ratingmat <- as.matrix(ratingmat[,-1])  
  
#Convert rating matrix into a recommenderlab sparse matrix  
ratingmat <- as(ratingmat, "realRatingMatrix")  
ratingmat

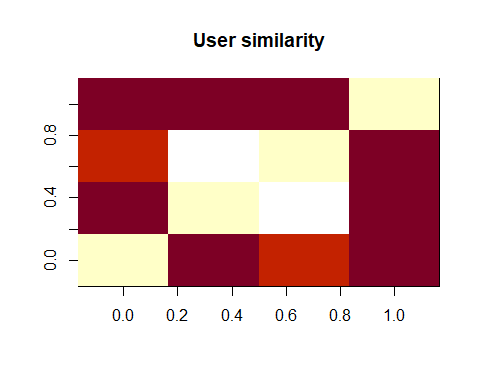
## 610 x 9724 rating matrix of class 'realRatingMatrix' with 100836 ratings.

## Exploring Similarity Data

Collaborative filtering algorithms are based on measuring the similarity between users or between items. For this purpose, *recommenderlab* contains the similarity function. The supported methods to compute similarities are *cosine, pearson*, and *jaccard*.

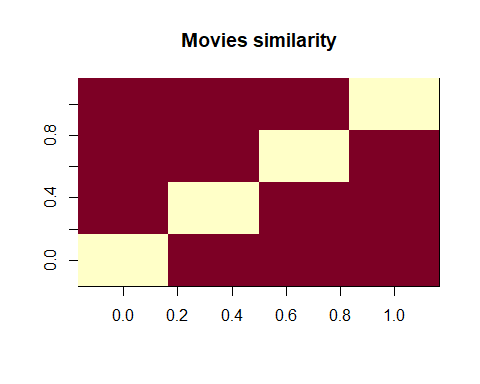
Next, I determine how similar the first four users are with each other by creating and visualizing similarity matrix that uses the cosine distance:

## 1 2 3 4  
## 1 0.0000000 1 0.7919033 0.9328096  
## 2 1.0000000 0 NA 1.0000000  
## 3 0.7919033 NA 0.0000000 1.0000000  
## 4 0.9328096 1 1.0000000 0.0000000

 In the given matrix, each row and each column corresponds to a user, and each cell corresponds to the similarity between two users. The more red the cell is, the more similar two users are. Note that the diagonal is red, since it’s comparing each user with itself.

Using the same approach, I compute similarity between the first four movies.

## 1 2 3 4  
## 1 0.0000000 0.9644641 0.9715415 0.9838699  
## 2 0.9644641 0.0000000 0.9389013 0.9609877  
## 3 0.9715415 0.9389013 0.0000000 1.0000000  
## 4 0.9838699 0.9609877 1.0000000 0.0000000

 ## Data Preparation The data preparation process consists of the following steps:

1. Select the relevant data.
2. Normalize the data.

#In order to predict the most relevant data, rating matrix is defined with the minimum number of users per rated movie as 50 and the minimum views number per movie as 50:

ratings\_movies <- ratingmat[rowCounts(ratingmat) > 50,  
 colCounts(ratingmat) > 50]  
ratings\_movies

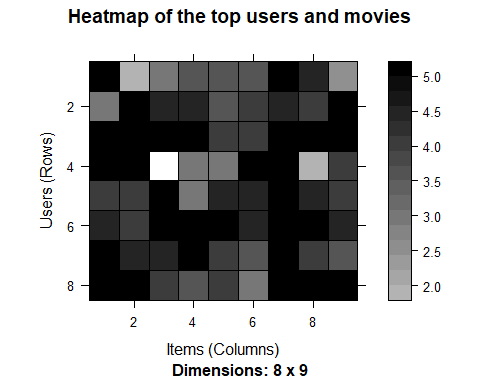
## 378 x 436 rating matrix of class 'realRatingMatrix' with 36214 ratings.

ratingmat

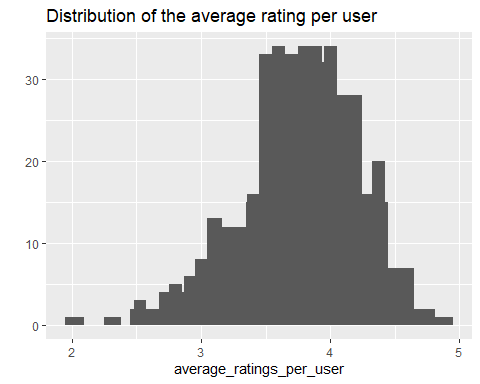
## 610 x 9724 rating matrix of class 'realRatingMatrix' with 100836 ratings.

Such a selection of the most relevant data contains 378 users and 436 movies, compared to previous 610 users and 9742 movies in the total dataset.

Using the same approach as previously, I visualize the top 2 percent of users and movies in the new matrix of the most relevant data:



## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



# Create a Train and Test set

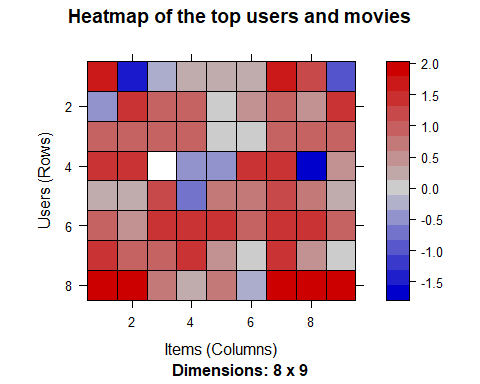
which\_train <- sample(x = c(TRUE, FALSE),   
 size = nrow(ratings\_movies),  
 replace = TRUE,   
 prob = c(0.8, 0.2))  
  
recc\_data\_train <- ratings\_movies[which\_train, ]  
recc\_data\_test <- ratings\_movies[!which\_train, ]

# Normalize the data

ratings\_movies\_norm <- recommenderlab::normalize(ratings\_movies)  
sum(rowMeans(ratings\_movies\_norm) > 0.00001)

## [1] 0

# visualize the normalized matrix for the top movies



recommender\_models <- recommenderRegistry$get\_entries(dataType = "realRatingMatrix")  
names(recommender\_models)

## [1] "HYBRID\_realRatingMatrix" "ALS\_realRatingMatrix"   
## [3] "ALS\_implicit\_realRatingMatrix" "IBCF\_realRatingMatrix"   
## [5] "LIBMF\_realRatingMatrix" "POPULAR\_realRatingMatrix"   
## [7] "RANDOM\_realRatingMatrix" "RERECOMMEND\_realRatingMatrix"   
## [9] "SVD\_realRatingMatrix" "SVDF\_realRatingMatrix"   
## [11] "UBCF\_realRatingMatrix"

lapply(recommender\_models, "[[", "description")

## $HYBRID\_realRatingMatrix  
## [1] "Hybrid recommender that aggegates several recommendation strategies using weighted averages."  
##   
## $ALS\_realRatingMatrix  
## [1] "Recommender for explicit ratings based on latent factors, calculated by alternating least squares algorithm."  
##   
## $ALS\_implicit\_realRatingMatrix  
## [1] "Recommender for implicit data based on latent factors, calculated by alternating least squares algorithm."  
##   
## $IBCF\_realRatingMatrix  
## [1] "Recommender based on item-based collaborative filtering."  
##   
## $LIBMF\_realRatingMatrix  
## [1] "Matrix factorization with LIBMF via package recosystem (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html)."  
##   
## $POPULAR\_realRatingMatrix  
## [1] "Recommender based on item popularity."  
##   
## $RANDOM\_realRatingMatrix  
## [1] "Produce random recommendations (real ratings)."  
##   
## $RERECOMMEND\_realRatingMatrix  
## [1] "Re-recommends highly rated items (real ratings)."  
##   
## $SVD\_realRatingMatrix  
## [1] "Recommender based on SVD approximation with column-mean imputation."  
##   
## $SVDF\_realRatingMatrix  
## [1] "Recommender based on Funk SVD with gradient descend (https://sifter.org/~simon/journal/20061211.html)."  
##   
## $UBCF\_realRatingMatrix  
## [1] "Recommender based on user-based collaborative filtering."

# I will use IBCF and UBCF models. Check the parameters of these two models.

recommender\_models$IBCF\_realRatingMatrix$parameters

## $k  
## [1] 30  
##   
## $method  
## [1] "Cosine"  
##   
## $normalize  
## [1] "center"  
##   
## $normalize\_sim\_matrix  
## [1] FALSE  
##   
## $alpha  
## [1] 0.5  
##   
## $na\_as\_zero  
## [1] FALSE

recommender\_models$UBCF\_realRatingMatrix$parameters

## $method  
## [1] "cosine"  
##   
## $nn  
## [1] 25  
##   
## $sample  
## [1] FALSE  
##   
## $weighted  
## [1] TRUE  
##   
## $normalize  
## [1] "center"  
##   
## $min\_matching\_items  
## [1] 0  
##   
## $min\_predictive\_items  
## [1] 0

# Create Recommender Model with “UBCF” on train set

recommender\_model <- Recommender(recc\_data\_train, method = "UBCF", param=list(method="Cosine",nn=30))  
recom <- predict(recommender\_model, recc\_data\_train[1], n=5)  
recom\_list <- as(recom, "list")  
   
recom\_result <- matrix(0,5)  
for (i in c(1:5)){  
 recom\_result[i] <- search\_matrix[as.integer(recom\_list[[1]][i]),2]  
}  
  
recom\_result

## [,1]   
## [1,] "Star Maps (1997)"   
## [2,] "Dr. Dolittle (1998)"   
## [3,] NA   
## [4,] "Ulee's Gold (1997)"   
## [5,] "Negotiator, The (1998)"

evaluation\_scheme <- evaluationScheme(recc\_data\_train, method="cross-validation", k=5, given=3, goodRating=5)  
evaluation\_results <- evaluate(evaluation\_scheme, method="UBCF", n=c(1,3,5,10,15,20))

## UBCF run fold/sample [model time/prediction time]  
## 1 [0.01sec/0.14sec]   
## 2 [0sec/0.11sec]   
## 3 [0sec/0.11sec]   
## 4 [0sec/0.11sec]   
## 5 [0sec/0.1sec]

eval\_results <- getConfusionMatrix(evaluation\_results)[[1]]  
eval\_results

## TP FP FN TN precision recall TPR  
## 1 0.062500 0.875000 15.50000 416.5625 0.06666667 0.004192841 0.004192841  
## 3 0.140625 2.671875 15.42188 414.7656 0.05000000 0.008460026 0.008460026  
## 5 0.203125 4.484375 15.35938 412.9531 0.04333333 0.013462199 0.013462199  
## 10 0.515625 8.859375 15.04688 408.5781 0.05500000 0.027687168 0.027687168  
## 15 0.734375 13.328125 14.82812 404.1094 0.05222222 0.046273618 0.046273618  
## 20 1.000000 17.750000 14.56250 399.6875 0.05333333 0.066089330 0.066089330  
## FPR  
## 1 0.002089703  
## 3 0.006381998  
## 5 0.010713419  
## 10 0.021149425  
## 15 0.031820778  
## 20 0.042377683

## Applying the recommender model on the test set

Determine the top ten recommendations for each new user in the test set.

n\_recommended <- 10  
recc\_predicted <- predict(object = recommender\_model,  
 newdata = recc\_data\_test,   
 n = n\_recommended)   
recc\_predicted

## Recommendations as 'topNList' with n = 10 for 66 users.

## Explore results

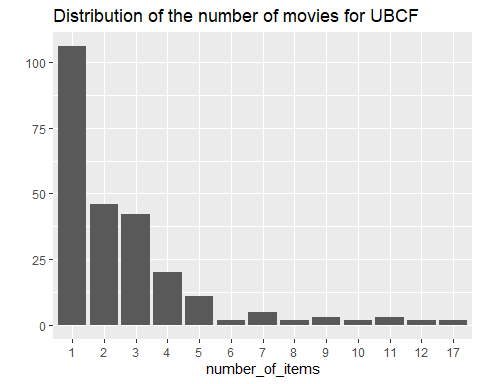
Let’s take a look at the first four users:

recc\_matrix <- sapply(recc\_predicted@items,   
 function(x){ as.integer(colnames(ratings\_movies)[x]) })  
#dim(recc\_matrix)  
recc\_matrix[, 1:4]

## [,1] [,2] [,3] [,4]  
## [1,] 3421 1035 1394 3  
## [2,] 3552 1148 1234 1917  
## [3,] 3471 3489 1220 1207  
## [4,] 33794 3421 1968 596  
## [5,] 4308 8360 253 1203  
## [6,] 3996 7361 3489 1278  
## [7,] 2997 2395 2657 235  
## [8,] 2710 6870 1617 1214  
## [9,] 6874 1234 1203 3471  
## [10,] 1500 1208 750 1079

I also compute how many times each movie got recommended and build the related frequency histogram:

number\_of\_items <- factor(table(recc\_matrix))  
  
chart\_title <- "Distribution of the number of movies for UBCF"  
qplot(number\_of\_items) + ggtitle(chart\_title)

 # The distribution has a longer tail. This means that there are some movies that are recommended much more often than the others. The maximum is more than 30, compared to 10-ish for IBCF.

Let’s take a look at the top titles:

## Movie title No of items  
## 1035 Sound of Music, The (1965) 17  
## 3489 Hook (1991) 17  
## 1234 Sting, The (1973) 12  
## 1394 Raising Arizona (1987) 12

## ITEM-based Collaborative Filtering Model

## Defining training/test sets

I build the model using 80% of the whole dataset as a training set, and 20% - as a test set.

## Building the recommendation model

## $k  
## [1] 30  
##   
## $method  
## [1] "Cosine"  
##   
## $normalize  
## [1] "center"  
##   
## $normalize\_sim\_matrix  
## [1] FALSE  
##   
## $alpha  
## [1] 0.5  
##   
## $na\_as\_zero  
## [1] FALSE

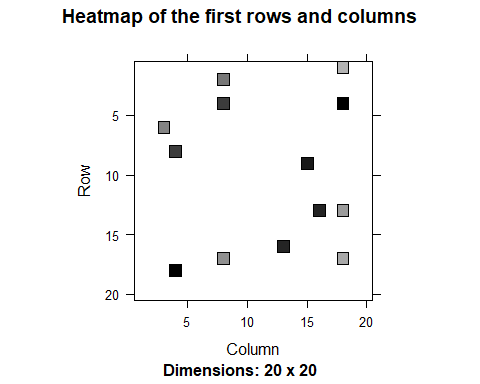
## Recommender of type 'IBCF' for 'realRatingMatrix'   
## learned using 301 users.

## [1] "Recommender"  
## attr(,"package")  
## [1] "recommenderlab"

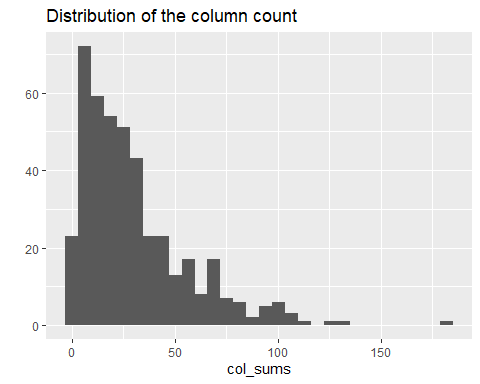
Exploring the recommender model:

## [1] "dgCMatrix"  
## attr(,"package")  
## [1] "Matrix"

## [1] 436 436



## row\_sums  
## 30   
## 436

 ## Applying recommender system on the dataset:

Now, it is possible to recommend movies to the users in the test set. I define *n\_recommended* equal to 10 that specifies the number of movies to recommend to each user.

For each user, the algorithm extracts its rated movies. For each movie, it identifies all its similar items, starting from the similarity matrix. Then, the algorithm ranks each similar item in this way:

* Extract the user rating of each purchase associated with this item. The rating is used as a weight.
* Extract the similarity of the item with each purchase associated with this item.
* Multiply each weight with the related similarity.
* Sum everything up.

Then, the algorithm identifies the top 10 recommendations:

## Recommendations as 'topNList' with n = 10 for 77 users.

Let’s explore the results of the recommendations for the first user:

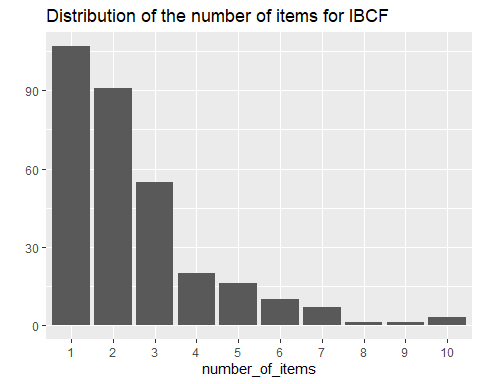
## [1] "Grumpier Old Men (1995)"   
## [2] "Legends of the Fall (1994)"   
## [3] "Outbreak (1995)"   
## [4] "Beverly Hills Cop III (1994)"   
## [5] "Nightmare Before Christmas, The (1993)"  
## [6] "Three Musketeers, The (1993)"   
## [7] "Starship Troopers (1997)"   
## [8] "Splash (1984)"   
## [9] "Big (1988)"   
## [10] "Hot Shots! Part Deux (1993)"

It’s also possible to define a matrix with the recommendations for each user. I visualize the recommendations for the first four users:

## [,1] [,2] [,3] [,4]  
## [1,] 3 539 145 16  
## [2,] 266 4025 2710 260  
## [3,] 292 48 6539 527  
## [4,] 420 2353 72998 4022  
## [5,] 551 168 4034 8874  
## [6,] 552 4447 1968 63082  
## [7,] 1676 552 1230 1997  
## [8,] 2100 420 1278 2701  
## [9,] 2797 597 1923 1028  
## [10,] 466 158 6863 1200

Here, the columns represent the first 4 users, and the rows are the *movieId* values of recommended 10 movies.

Now, let’s identify the most recommended movies. The following image shows the distribution of the number of items for IBCF:



## Movie title No of items  
## 3 Grumpier Old Men (1995) 10  
## 11 American President, The (1995) 10  
## 168 First Knight (1995) 10  
## 292 Outbreak (1995) 9

Most of the movies have been recommended only a few times, and a few movies have been recommended more than 5 times.

## Evaluating the Recommender Systems

### Using cross-validation to validate models

The k-fold cross-validation approach is the most accurate one, although it’s computationally heavier.

Using this approach, we split the data into some chunks, take a chunk out as the test set, and evaluate the accuracy. Then, we can do the same with each other chunk and compute the average accuracy.

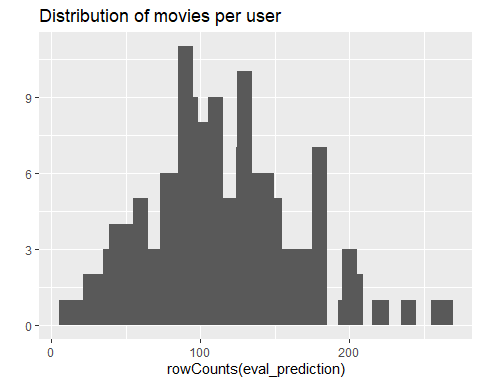
n\_fold <- 4  
rating\_threshold <- 3  
items\_to\_keep <- 5  
eval\_sets <- evaluationScheme(data = ratings\_movies,   
 method = "cross-validation",  
 k = n\_fold,   
 given = items\_to\_keep,   
 goodRating = rating\_threshold)  
size\_sets <- sapply(eval\_sets@runsTrain, length)  
size\_sets

## [1] 282 282 282 282

Using 4-fold approach, we get four sets of the same size 282

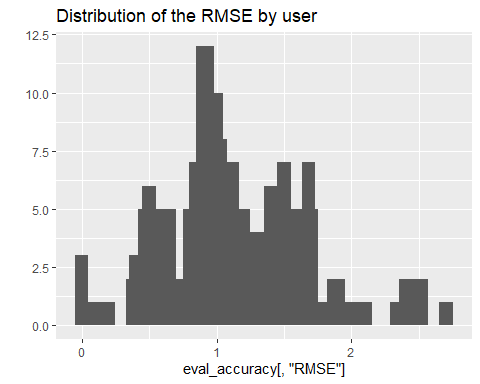
## Evavluating the ratings

First, I re-define the evaluation sets, build IBCF model and create a matrix with predicted ratings.

 The above image displays the distribution of movies per user in the matrix of predicted ratings.

Now, I compute the accuracy measures for each user. Most of the RMSEs (Root mean square errors) are in the range of 0.5 to 1.8:

## RMSE MSE MAE  
## [1,] 1.500000 2.250000 1.2500000  
## [2,] 1.705325 2.908135 1.3167317  
## [3,] 1.694832 2.872456 1.4450623  
## [4,] 2.389212 5.708333 1.7500000  
## [5,] 1.349603 1.821429 0.9285714  
## [6,] 1.536026 2.359375 1.2187500

 In order to have a performance index for the whole model, I specify *byUser* as FALSE and compute the average indices:

## RMSE MSE MAE   
## 1.3135835 1.7255017 0.9746154

The measures of accuracy are useful to compare the performance of different models on the same data.

## Evaluating the recommendations

Another way to measure accuracies is by comparing the recommendations with the purchases having a positive rating. For this, I can make use of a prebuilt *evaluate* function in *recommenderlab* library. The function evaluate the recommender performance depending on the number *n* of items to recommend to each user. I use *n* as a sequence n = seq(10, 100, 10). The first rows of the resulting performance matrix is presented below:

## IBCF run fold/sample [model time/prediction time]  
## 1 [0.36sec/0.03sec]   
## 2 [0.35sec/0.01sec]   
## 3 [0.32sec/0.02sec]   
## 4 [0.33sec/0.02sec]

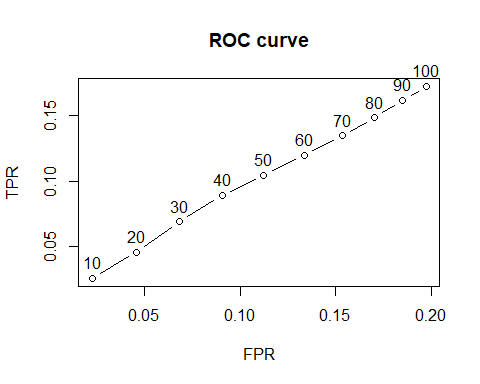
## TP FP FN TN precision recall TPR  
## 10 1.770833 8.229167 69.63542 351.3646 0.1770833 0.02277006 0.02277006  
## 20 3.208333 16.718750 68.19792 342.8750 0.1615385 0.04218315 0.04218315  
## 30 4.927083 24.833333 66.47917 334.7604 0.1662453 0.06920564 0.06920564  
## 40 6.343750 33.083333 65.06250 326.5104 0.1615075 0.08801226 0.08801226  
## 50 7.479167 41.322917 63.92708 318.2708 0.1536698 0.10152940 0.10152940  
## 60 8.604167 49.197917 62.80208 310.3958 0.1487971 0.11338028 0.11338028  
## FPR  
## 10 0.02292504  
## 20 0.04655397  
## 30 0.06910429  
## 40 0.09206880  
## 50 0.11521770  
## 60 0.13728877

In order to have a look at all the splits at the same time, I sum up the indices of columns TP, FP, FN and TN:

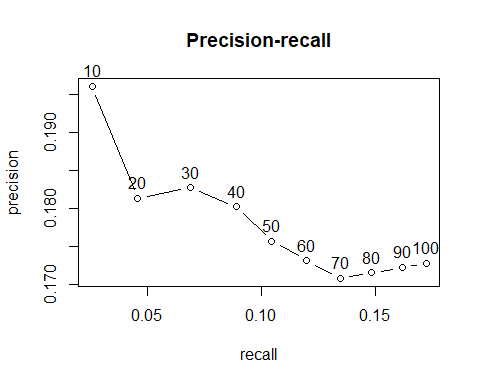
## TP FP FN TN  
## 10 7.666667 31.56250 317.4688 1367.302  
## 20 14.114583 64.12500 311.0208 1334.740  
## 30 21.218750 95.29167 303.9167 1303.573  
## 40 27.697917 126.26042 297.4375 1272.604  
## 50 33.343750 156.73958 291.7917 1242.125  
## 60 38.833333 186.16667 286.3021 1212.698

Finally, I plot the ROC and the precision/recall curves:

plot(results, annotate = TRUE, main = "ROC curve")



plot(results, "prec/rec", annotate = TRUE, main = "Precision-recall")

 If a small percentage of rated movies is recommended, the precision decreases. On the other hand, the higher percentage of rated movies is recommended the higher is the recall.