Run: https://dchatter.shinyapps.io/movieRec/

Movie Recommender

NETFLIX
House of Cards
2013
1 Season

Sharks gliding ominously beneath the surface of the water? They're a lot less menacing than this Congressman.

Because you watched Orange Is the New Black
HOUSE of CARDS
the L word
IT'S ALWAYS SUNNY IN PHILADELPHIA
New Girl
Because you watched Red Lights
ELECTRIC MIST

Debasis Chatterjee, IST-707

Introduction

In today's world, every customer is faced with multiple choices. For example, If someone is looking for a book to read without any specific idea of what he/she wants, there's a wide range of possibilities how my search might pan out. The Person might waste a lot of time browsing around on the internet and trawling through various sites hoping to strike gold. He might look for recommendations from other people. Also, at some point each one of us must have wondered where all the recommendations that Netflix, Amazon, Google give us, come from. We often rate products on the internet and all the preferences we express and data we share (explicitly or not), are used by recommender systems to generate, in fact, recommendations.

Project Overview

In this project, a collaborative filtering recommender (CFR) system for recommending movies has been developed.

The basic idea of CFR systems is that, if two users share the same interests in the past, e.g. they liked the same book or the same movie, they will also have similar tastes in the future. If, for example, user A and user B have a similar purchase history and user A recently bought a book that user B has not yet seen, the basic idea is to propose this book to user B.

The collaborative filtering approach considers only user preferences and does not take into account the features or contents of the items (books or movies) being recommended. In this project, in order to recommend movies I will use a large set of users preferences towards the movies from a publicly available movie rating dataset.

Objective and Business Questions

In this project, a collaborative filtering recommender (CFR) system for recommending movies with the main objective of recommending relevant movies to the users has been developed. In the era of competitors like Amazon Prime, Netflix, Hulu, companies want maximum users to stick to their product which will result in higher profits. So, this recommender system plans to provide optimized recommendation of movies to increase viewership on our product. This project plans to give vital information to marketing department so that they can market a movie in order to make profits. Also, the PR team and Design team will use the information generated from this project to place movies efficiently on the website and target specific segment of customers.

There are two approaches develop a recommender system:

- Collaborative filtering: Collaborative filtering approaches build a model from user's past behavior (i.e. items purchased or searched by the user) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that user may have an interest in.
- Content-based filtering: Content-based filtering approaches uses a series of discrete characteristics of an item in order to recommend additional items with similar properties. Content-based filtering methods are totally based on a description of the item and a profile of the user's preferences. It recommends items based on user's past preferences.

Used Libraries

The following libraries were used in this project:

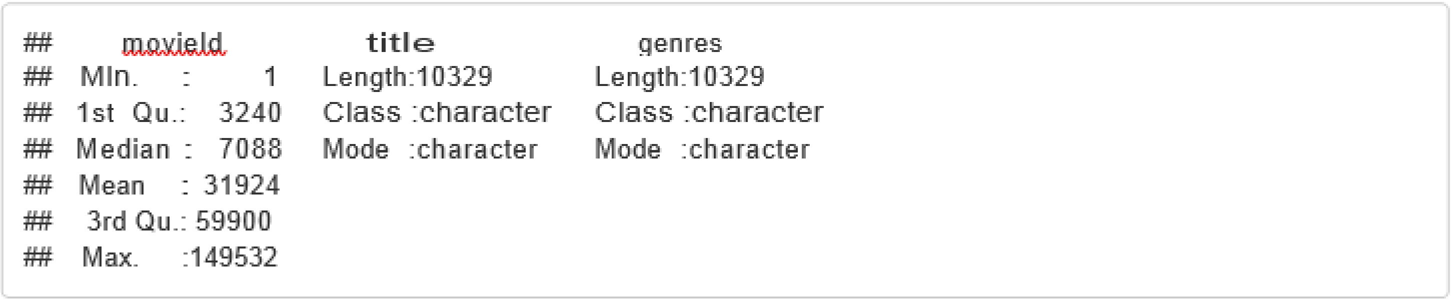
```r
library(recommenderlab)
library(ggplot2)
library(data.table)
library(reshape2)
```

Dataset

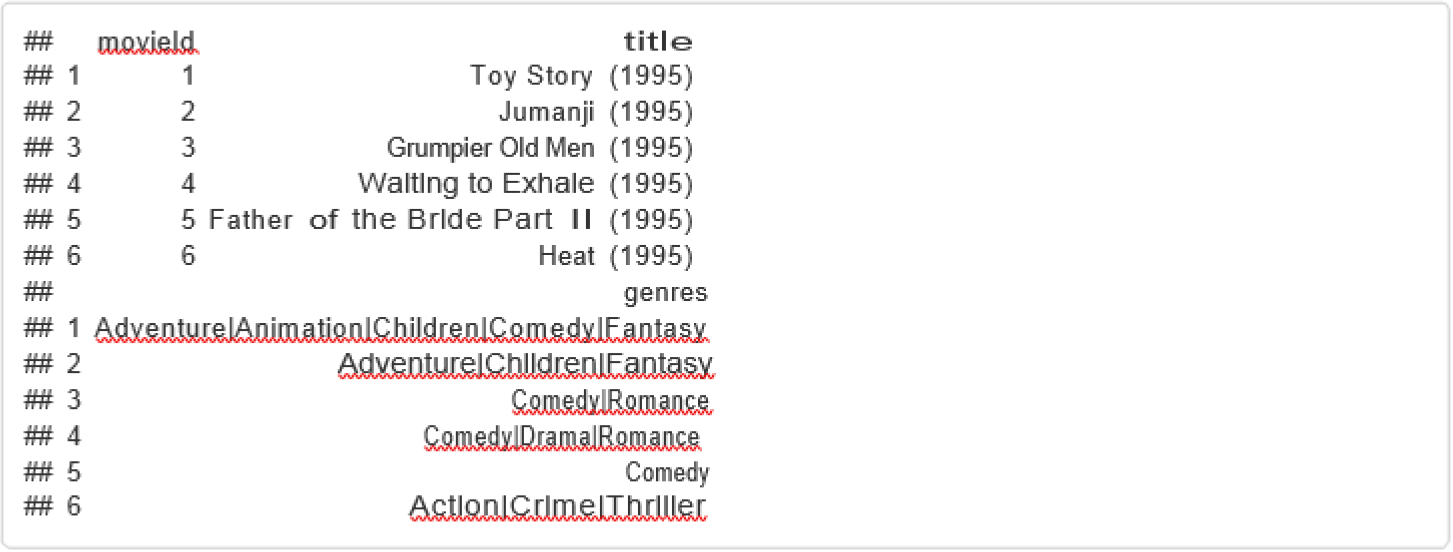
The dataset used was from MovieLens, and is publicly available at http://grouplens.org/datasets/movielens/latest. In order to keep the recommender simple, smallest dataset available (ml-latest-small.zip) is used, which at the time of download contained 105339 ratings and 6138 tag applications across 10329 movies. These data were created by 668 users between April 03, 1996 and January 09, 2016. This dataset was generated on January 11, 2016.

The data are contained in four files: *links.csv*, *movies.csv*, *ratings.csv* and *tags.csv*. The only used files are *movies.csv* and *ratings.csv* to build a recommendation system.

A summary of *movies* is given below, together with several first rows of a dataframe:



```
##      movieId           title              genres
##   Min.   :      1   Length:10329       Length:10329
##   1st Qu.:   3240   Class :character   Class :character
##   Median :   7088   Mode  :character   Mode  :character
##   Mean   :  31924
##   3rd Qu.: 59900
##   Max.   :149532
```



```
##    movieId                             title
## 1        1                  Toy Story (1995)
## 2        2                    Jumanji (1995)
## 3        3           Grumpier Old Men (1995)
## 4        4          Waiting to Exhale (1995)
## 5        5 Father of the Bride Part II (1995)
## 6        6                       Heat (1995)
##                                               genres
## 1 Adventure|Animation|Children|Comedy|Fantasy
## 2                   Adventure|Children|Fantasy
## 3                               Comedy|Romance
## 4                         Comedy|Drama|Romance
## 5                                       Comedy
## 6                        Action|Crime|Thriller
```

And here is a summary and a head of *ratings*:

```
##      userId         movieId           rating        timestamp        
##  Min.   :  1.0   Min.   :     1   Min.   :0.500   Min.   :8.286e+08  
##  1st Qu.:192.0   1st Qu.:  1073   1st Qu.:3.000   1st Qu.:9.711e+08  
##  Median :383.0   Median :  2497   Median :3.500   Median :1.115e+09  
##  Mean   :364.9   Mean   : 13381   Mean   :3.517   Mean   :1.130e+09  
##  3rd Qu.:557.0   3rd Qu.:  5991   3rd Qu.:4.000   3rd Qu.:1.275e+09  
##  Max.   :668.0   Max.   :149532   Max.   :5.000   Max.   :1.452e+09
```

```
##   userId movieId rating  timestamp
## 1      1      16    4.0 1217897793
## 2      1      24    1.5 1217895807
## 3      1      32    4.0 1217896246
## 4      1      47    4.0 1217896556
## 5      1      50    4.0 1217896523
## 6      1     110    4.0 1217896150
```

Both *usersId* and *movieId* are presented as integers and should be changed to factors. Genres of the movies are not easily usable because of their format, I will deal with this in the next step.

Data Pre-processing

Some pre-processing of the data available is required before creating the recommendation system.

First of all, re-organization of the information of movie genres is done in such a way that allows future users to search for the movies they like within specific genres. From the design perspective, this is much easier for the user compared to selecting a movie from a single very long list of all the available movies.

Extract a list of genres

One-hot note is used to encoding to create a matrix of corresponding genres for each movie.

```
##   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
```

Create a matrix to search for a movie by genre

Now, a *search matrix* is created which allows an easy search of a movie by any of its genre.

```
##   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
```

We can see that each movie can correspond to either one or more than one genre.

Converting ratings matrix in a proper format

In order to use the ratings data for building a recommendation engine with *recommenderlab*, I convert rating matrix into a sparse matrix of type *realRatingMatrix*.

```
## 668 x 10325 rating matrix of class 'realRatingMatrix' with 105339 ratings.
```

Exploring Parameters of Recommendation Models

The *recommenderlab* package contains some options for the recommendation algorithm:

```
## [1] "IBCF_realRatingMatrix"          "POPULAR_realRatingMatrix"
## [3] "RANDOM_realRatingMatrix"        "RERECOMMEND_realRatingMatrix"
## [5] "SVD_realRatingMatrix"           "SVDF_realRatingMatrix"
## [7] "UBCF_realRatingMatrix"
```

```
## $IBCF_realRatingMatrix
## [1] "Recommender based on item-based  collaborative  filtering (real  data)."
##
##  $POPULAR_realRatingMatrix
## [1] "Recommender based on item popularity (real data)."
##
##  $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  (real data)."
##
## $SVDF_realRatingMatrix
## [1] "Recommender based on Funk SVD with gradient descend (real data)."
##
## $UBCF_realRatingMatrix
## [1] "Recommender based on user-based collaborative filtering (real data)."
```

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
```

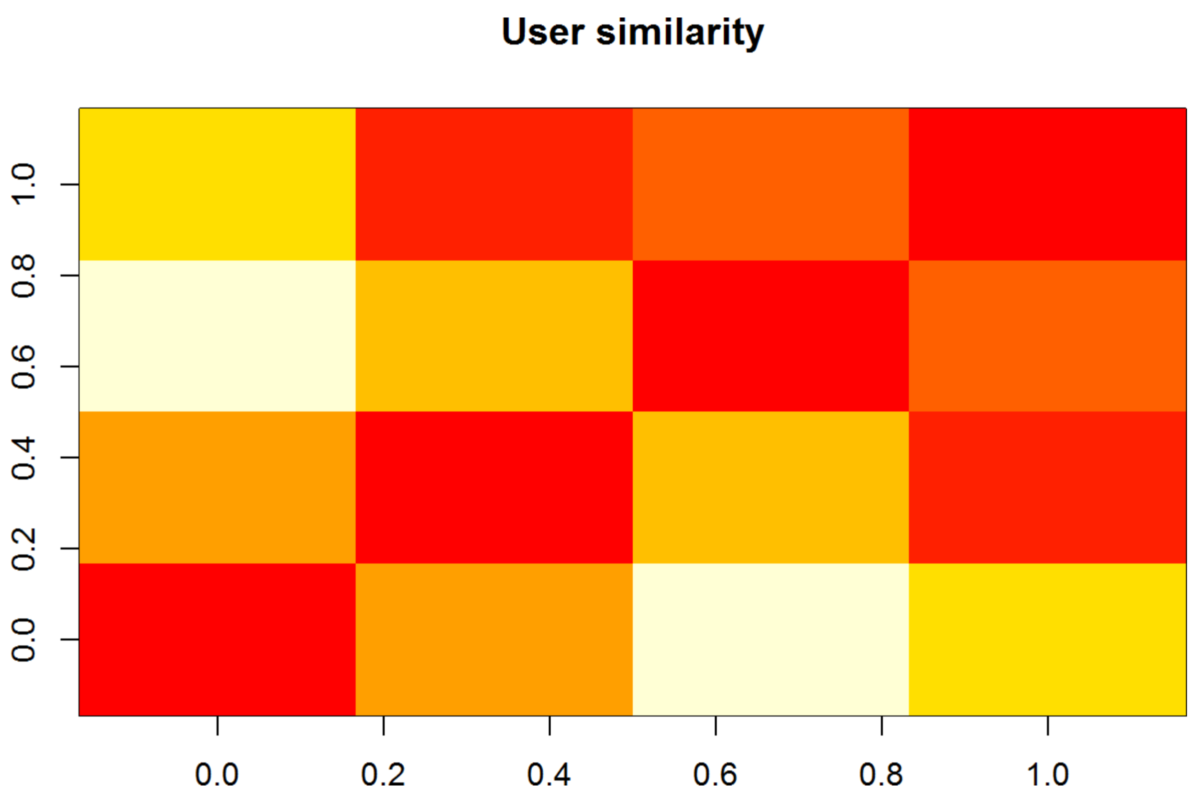
```
## $method
## [1] "cosine"
##
## $nn
## [1] 25
##
## $sample
## [1] FALSE
##
## $normalize
## [1] "center"
```

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, A determination I done on how similar the first four users are with each other by creating and visualizing similarity matrix that uses the cosine distance:

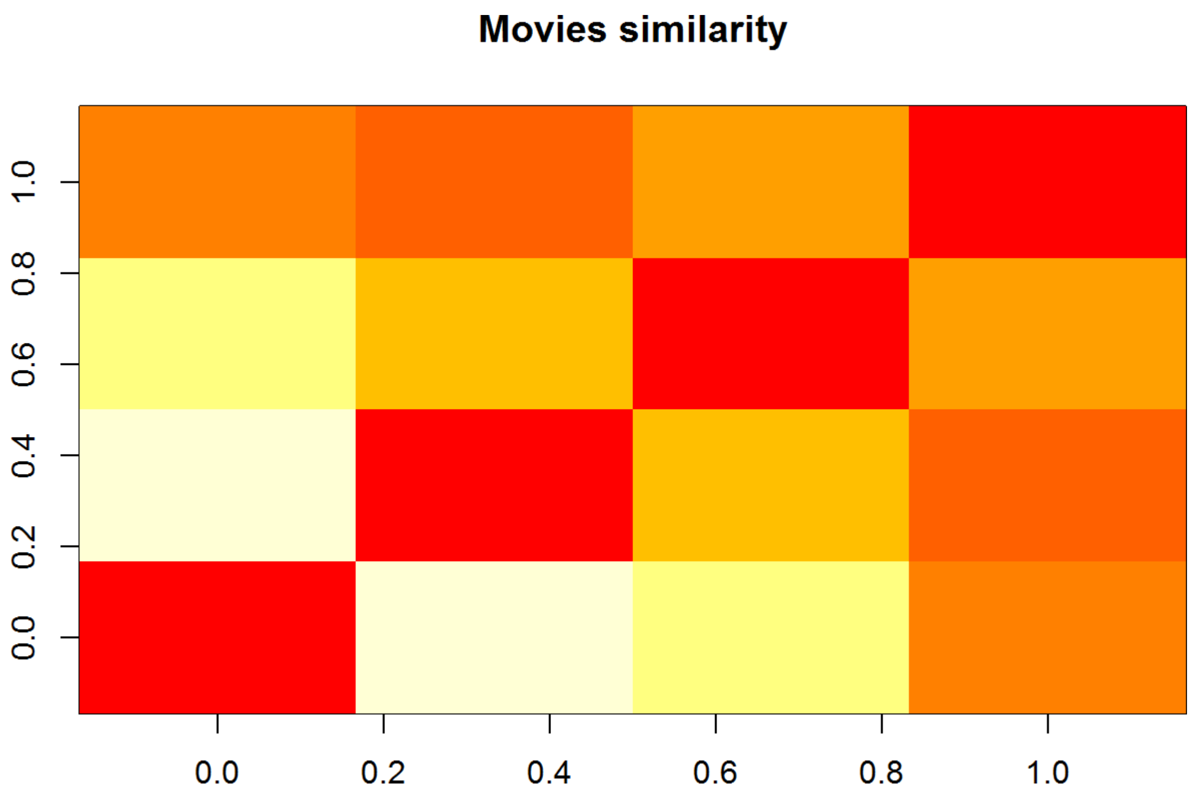
```
##            1         2          3          4
## 1  0.0000000 0.1011133 0.21004361 0.12876575
## 2  0.1011133 0.0000000 0.11555911 0.03461020
## 3  0.2100436 0.1155591 0.00000000 0.05820771
## 4  0.1287658 0.0346102 0.05820771 0.00000000
```



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 0.3830684 0.3374528 0.1347243
## 2 0.3830684 0.0000000 0.1992068 0.1233765
## 3 0.3374528 0.1992068 0.0000000 0.1733663
## 4 0.1347243 0.1233765 0.1733663 0.0000000
```



Further data exploration

Now, explore values of ratings is done.

```
vector_ratings <- as.vector(ratingmat@data)
unique(vector_ratings) # what are unique values of ratings
```

```
## [1] 0.0 5.0 4.0 3.0 4.5 1.5 2.0 3.5 1.0 2.5 0.5
```

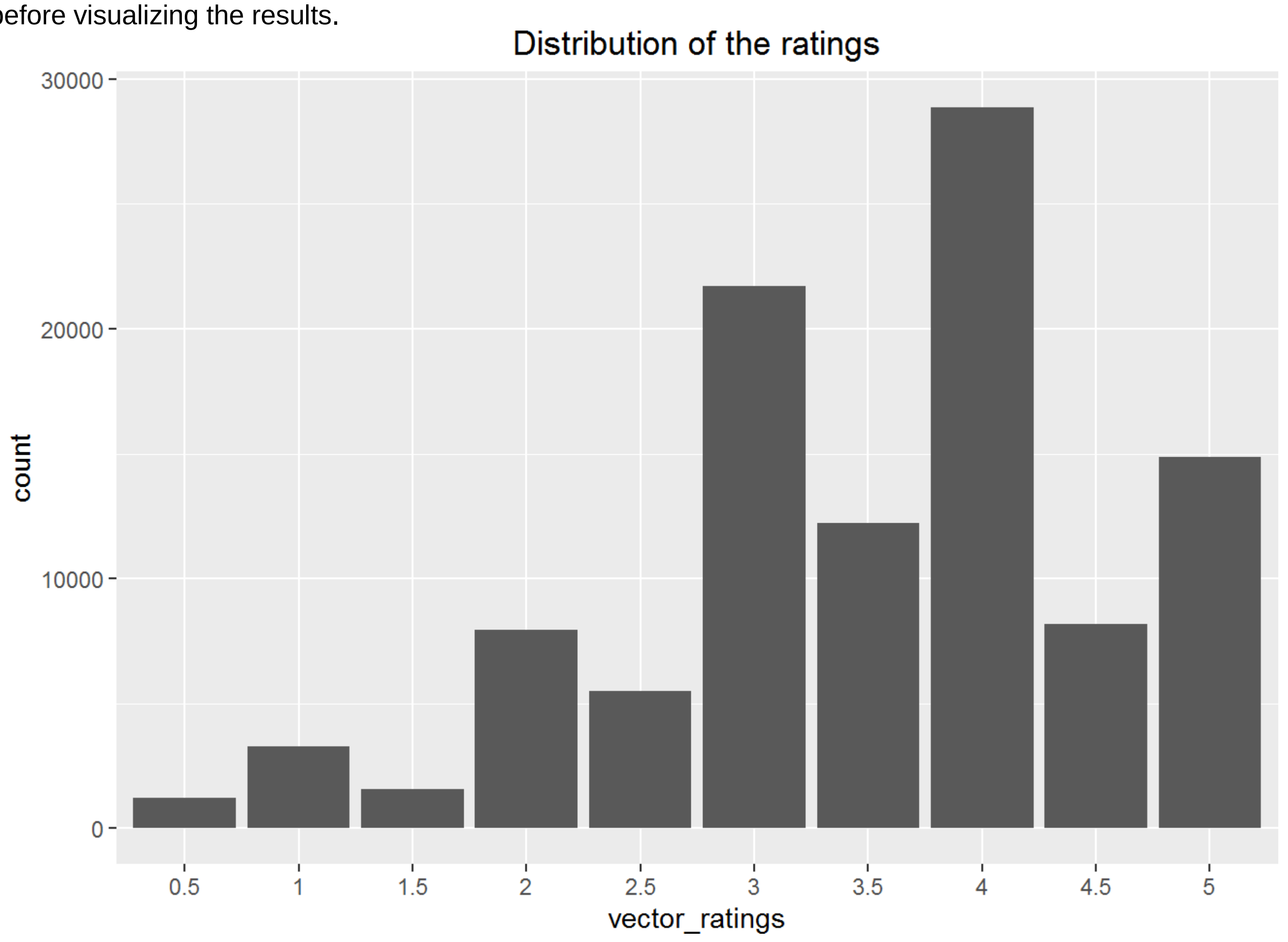
```
table_ratings <- table(vector_ratings) # what is the count of each rating value
table_ratings
```

```
## vector_ratings
##       0     0.5       1     1.5       2     2.5       3     3.5       4
## 6791761    1198    3258    1567    7943    5484   21729   12237   28880
##     4.5       5
##    8187   14856
```

There are 11 unique score values. The lower values mean lower ratings and vice versa.

Distribution of the ratings

According to the documentation, a rating equal to 0 represents a missing value, so I remove them from the dataset before visualizing the results.

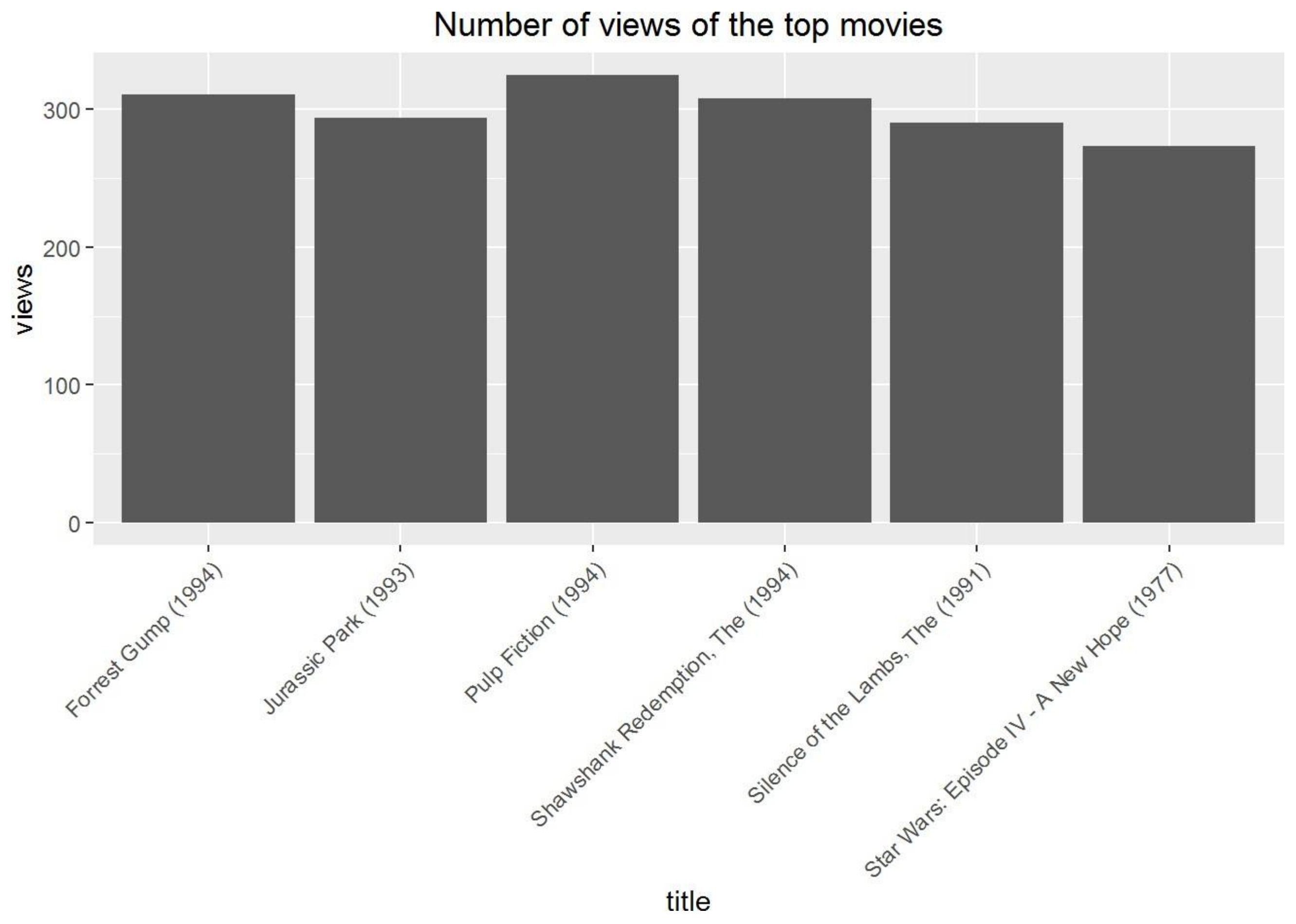


As we see, there are less low (less than 3) rating scores, the majority of movies are rated with a score of 3 or higher. The most common rating is 4.

Number of views of the top movies

Now, let's see what are the most viewed movies.

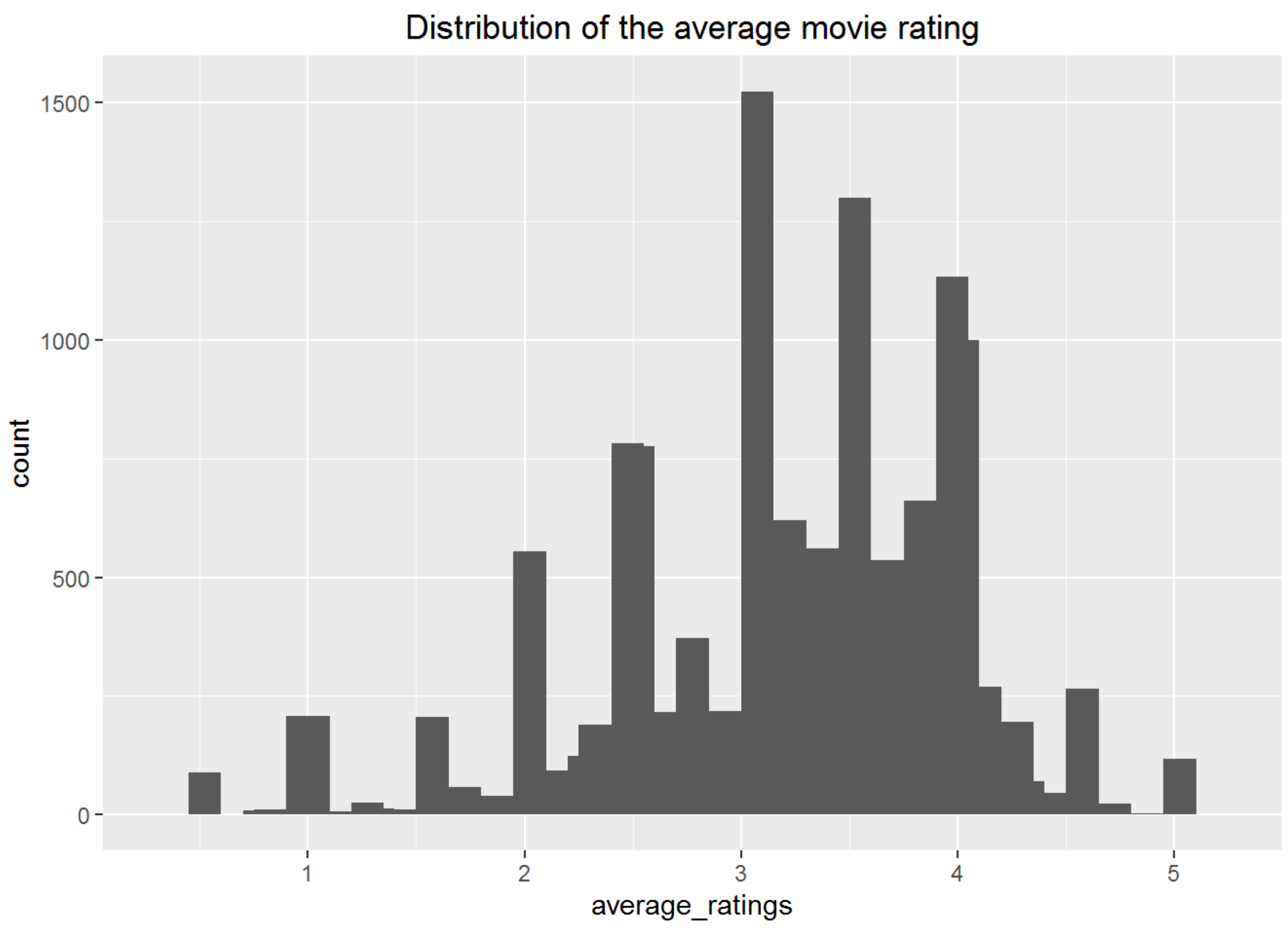
```
##      movie views                                  title
## 296    296   325                    Pulp Fiction (1994)
## 356    356   311                    Forrest Gump (1994)
## 318    318   308       Shawshank Redemption, The (1994)
## 480    480   294                   Jurassic Park (1993)
## 593    593   290       Silence of the Lambs, The (1991)
## 260    260   273 Star Wars: Episode IV - A New Hope (1977)
```

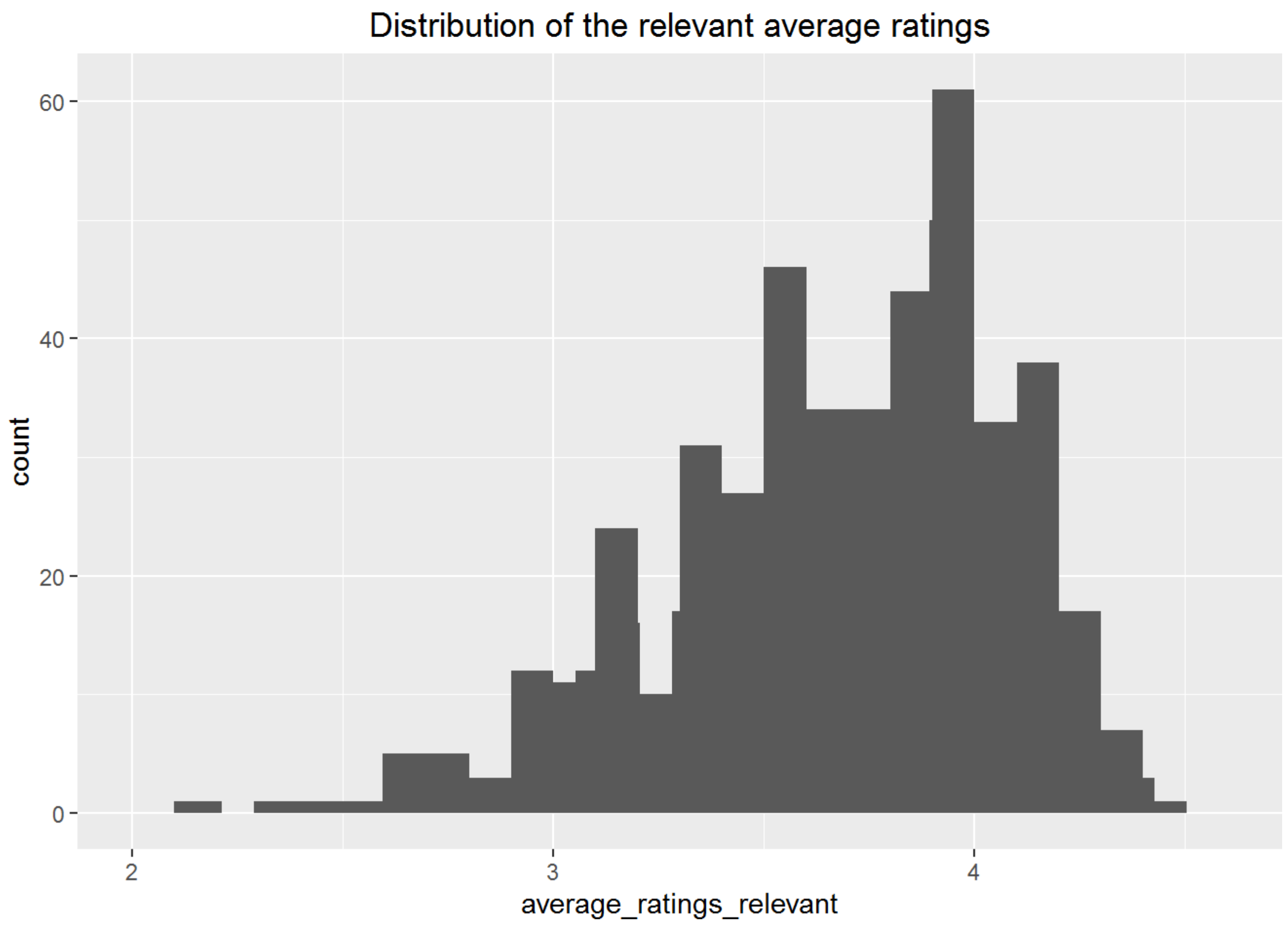


We see that “Pulp Fiction (1994)” is the most viewed movie, exceeding the second-most-viewed “Forrest Gump (1994)” by 14 views.

Distribution of the average movie rating

Now I identify the top-rated movies by computing the average rating of each of them.


Distribution of the average movie rating
count
1500
1000
500
0
1
2
3
4
5
average_ratings

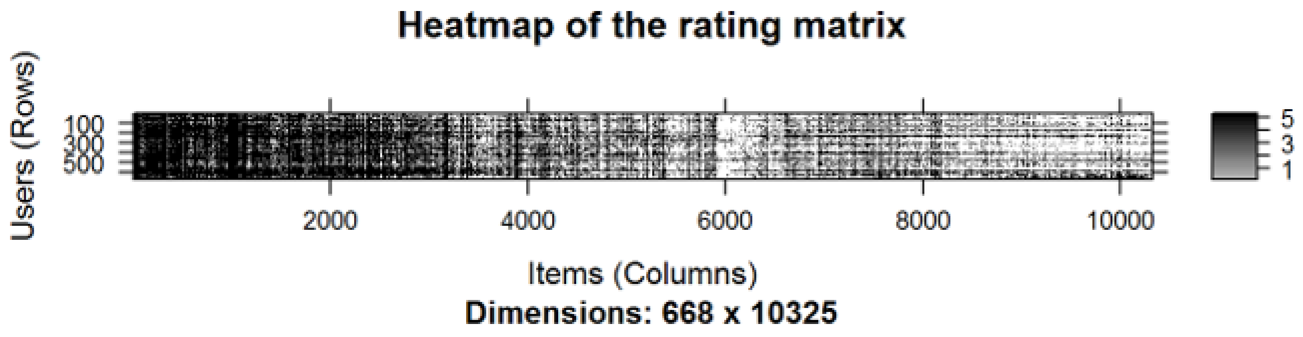

Distribution of the relevant average ratings
count
60
40
20
0
2
3
4
average_ratings_relevant

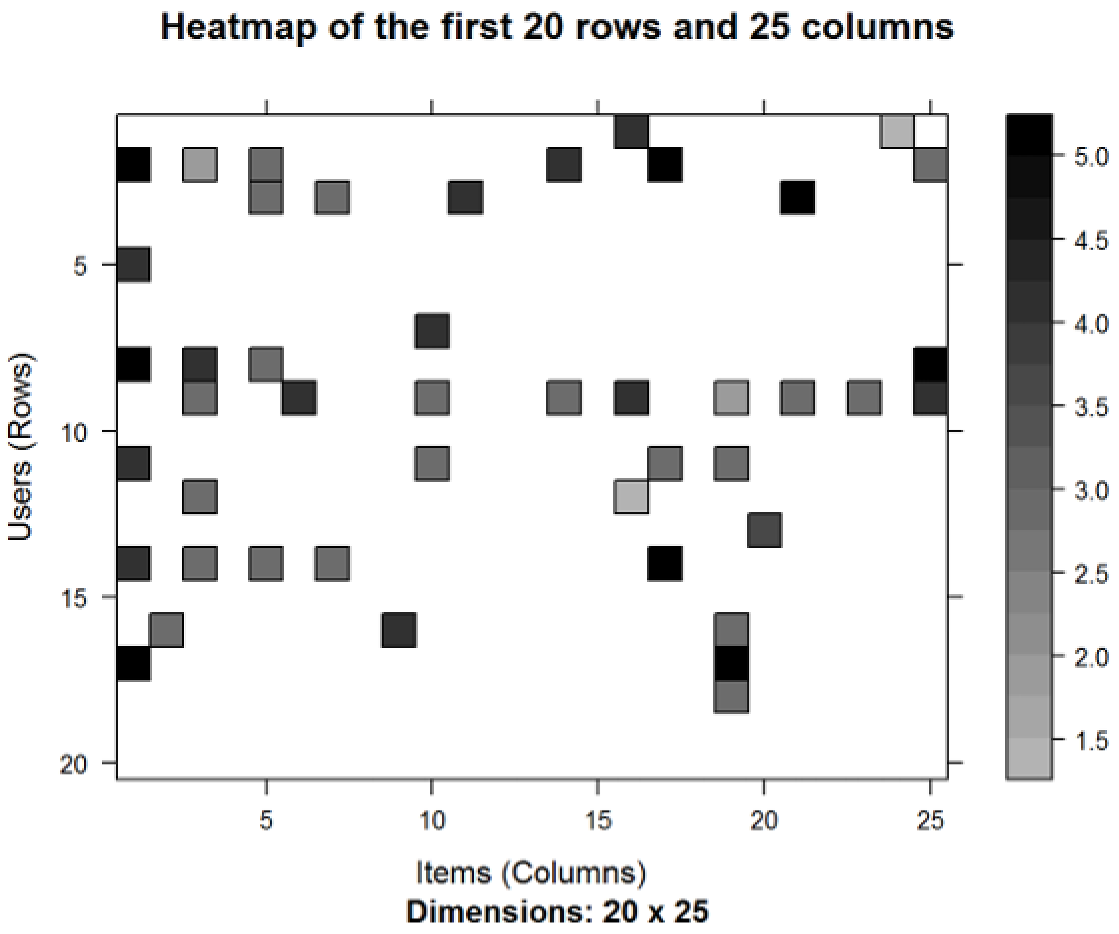
The first image above shows the distribution of the average movie rating. The highest value is around 3, and there are a few movies whose rating is either 1 or 5. Probably, the reason is that these movies received a rating from a few people only, so we shouldn't take them into account.

I remove the movies whose number of views is below a defined threshold of 50, creating a subset of only relevant movies. The second image above shows the distribution of the relevant average ratings. All the rankings are between 2.16 and 4.45. As expected, the extremes were removed. The highest value changes, and now it is around 4.

Heatmap of the rating matrix

The whole matrix of ratings is visualized by building a heat map whose colors represent the ratings. Each row of the matrix corresponds to a user, each column to a movie, and each cell to its rating.





Since there are too many users and items, the first chart is hard to read. The second chart is built zooming in on the first rows and columns.

Some users saw more movies than the others. So, instead of displaying some random users and items, the most relevant users and items should be selected. Thus visualization is done only the users who have seen many movies and the movies that have been seen by many users. To identify and select the most relevant users and movies, the following these steps are followed:

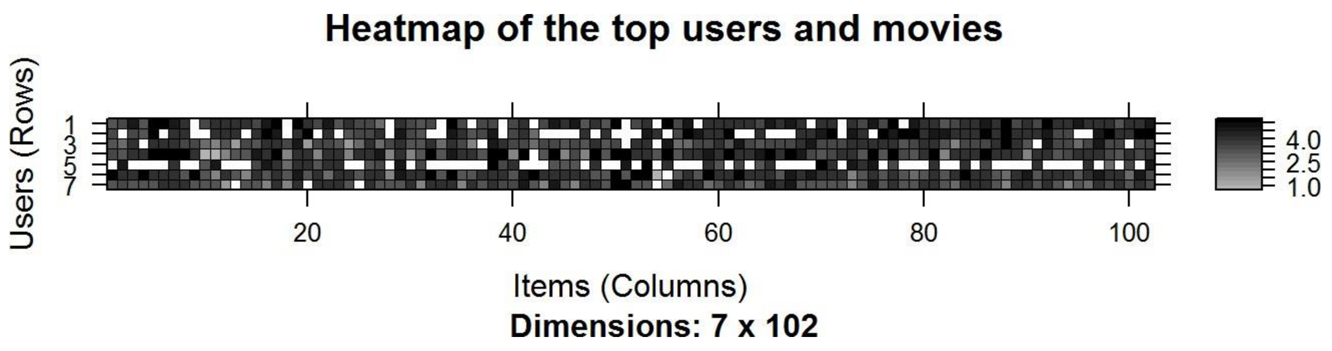
1. Determine the minimum number of movies per user.
2. Determine the minimum number of users per movie.
3. Select the users and movies matching these criteria.

```
## [1] "Minimum number of movies per user:"
```

```
##         99%
## 1198.17
```

```
## [1] "Minimum number of users per movie:"
```

```
## 99%
##  115
```



Let’s take account of the users having watched more movies. Most of them have seen all the top movies, and this is not surprising. Some columns of the heatmap are darker than the others, meaning that these columns represent the highest-rated movies. Conversely, darker rows represent users giving higher ratings. Because of this, it might be useful to normalize the data, which is done in the next step.

Data Preparation

The data preparation process consists of the following steps:

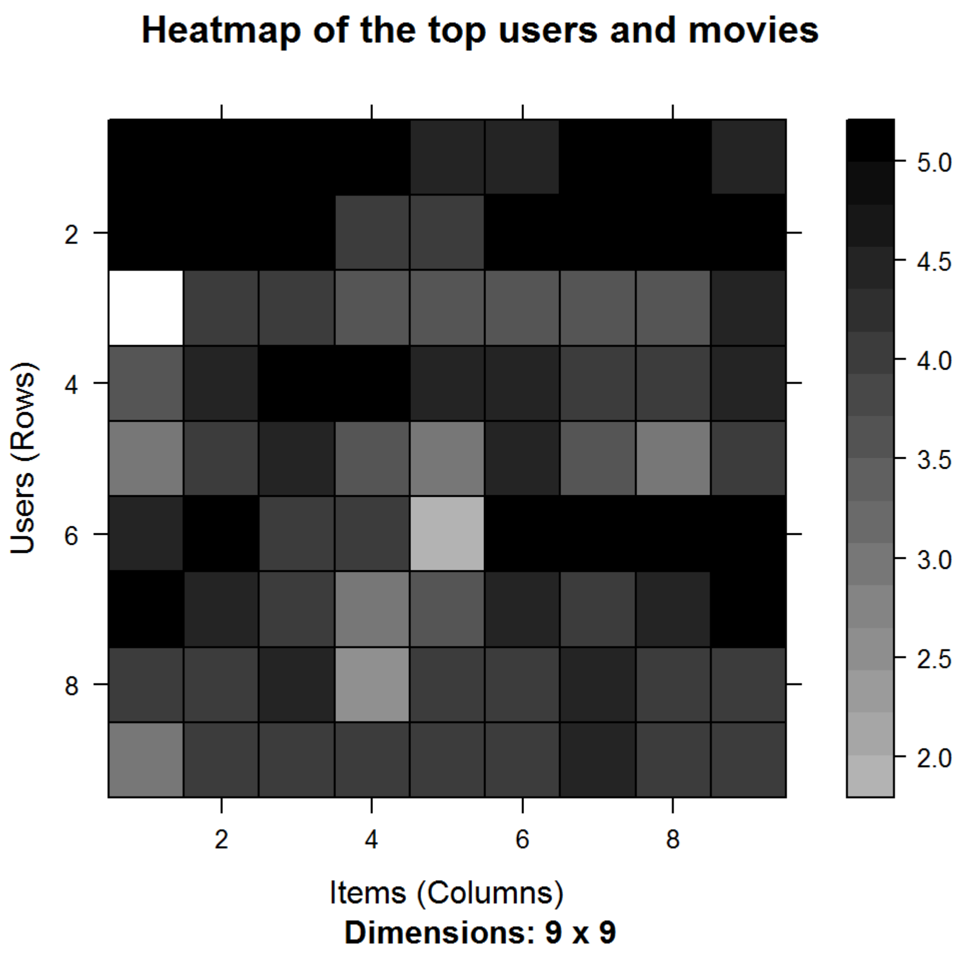
1. Select the relevant data.
2. Normalize the data.
3. Binarize the data.

In order to select the most relevant data, the minimum number of users per rated movie is defined as 50 and the minimum views number per movie as 50:

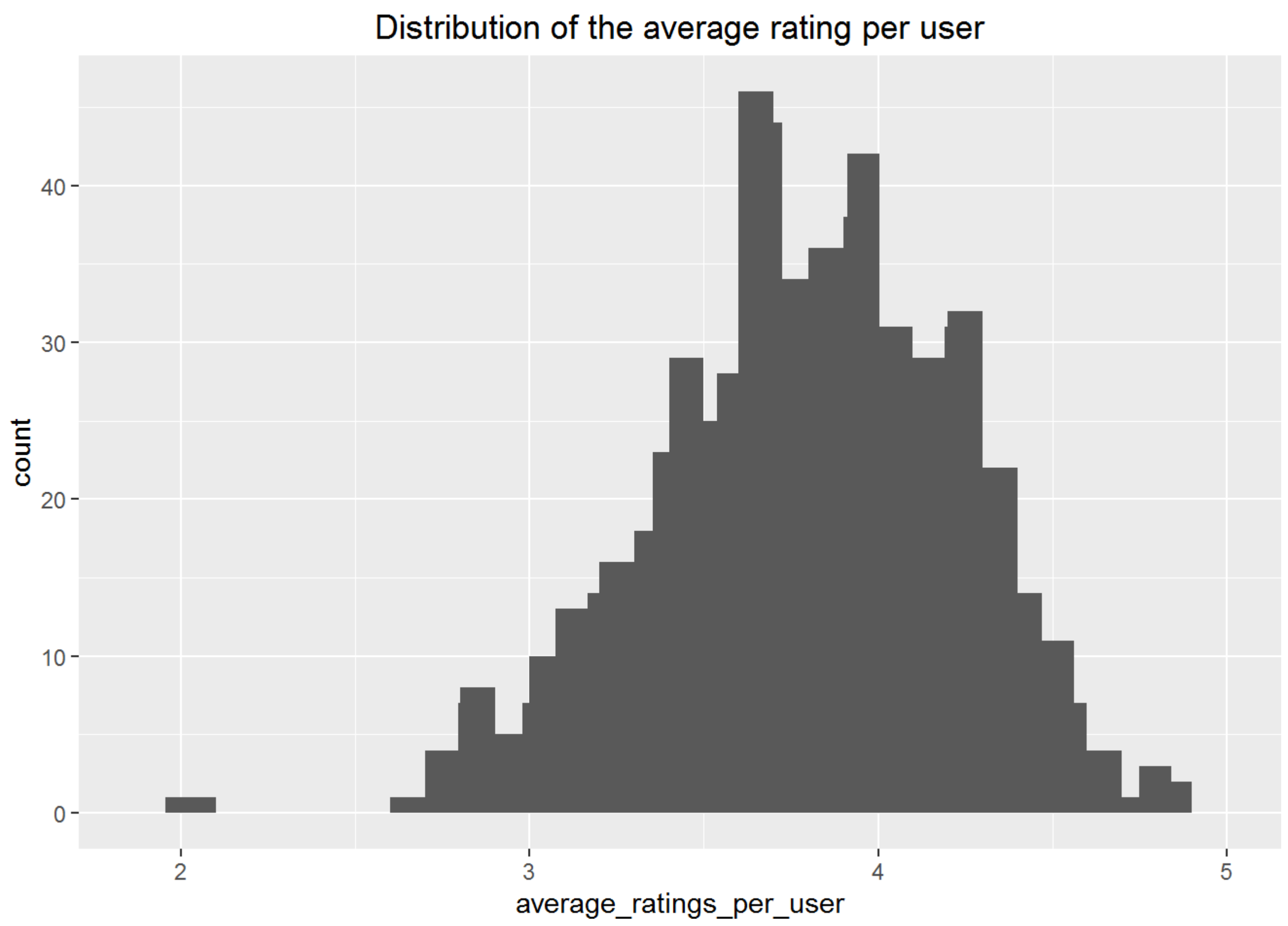
```
## 420 x 447 rating matrix of class 'realRatingMatrix' with 38341 ratings.
```

Such a selection of the most relevant data contains 420 users and 447 movies, compared to previous 668 users and 10325 movies in the total dataset.

Using the same approach as previously, 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`.
```



In the heatmap, some rows are darker than the others. This might mean that some users give higher ratings to all the movies. The distribution of the average rating per user across all the users varies a lot, as the second chart above shows.

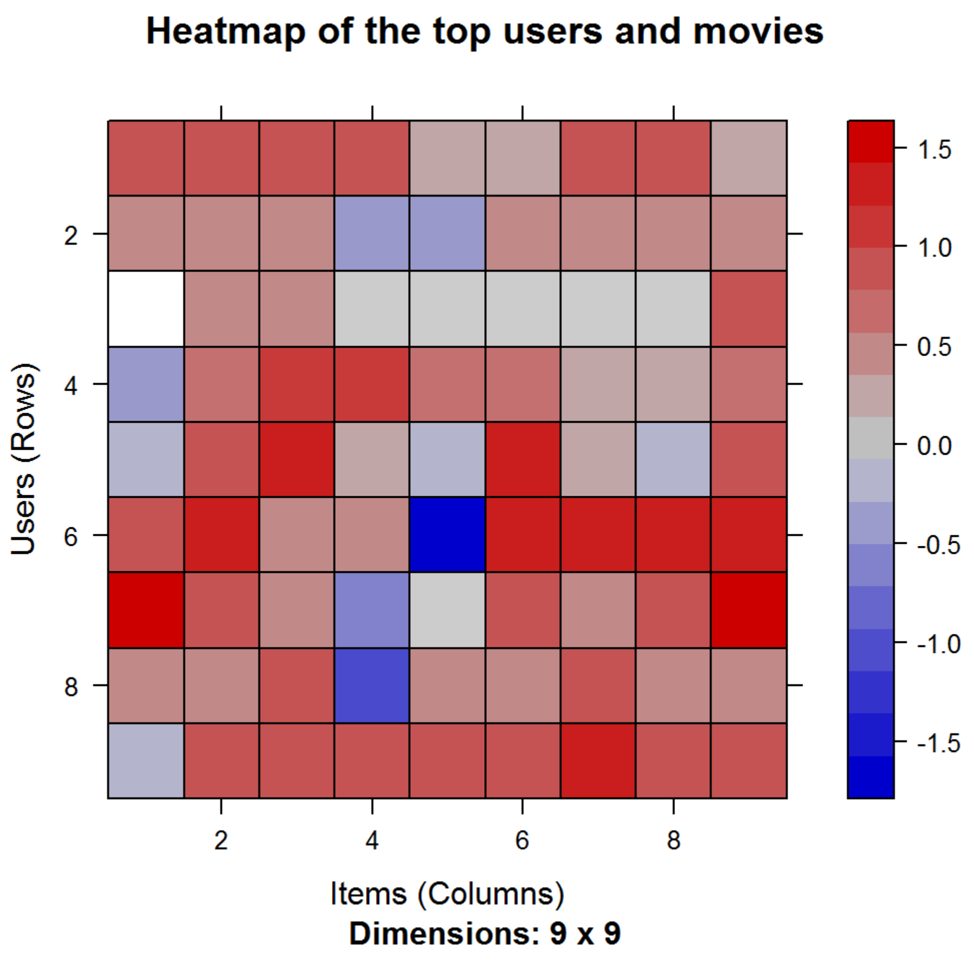
Normalizing data

Having users who give high (or low) ratings to all their movies might bias the results. In order to remove this effect, I normalize the data in such a way that the average rating of each user is 0. As a quick check, calculation of the average rating by users, and it is equal to 0 is done, as expected:

```
ratings_movies_norm <- normalize(ratings_movies)
sum(rowMeans(ratings_movies_norm) > 0.00001)
```

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

Now, the normalized matrix for the top movies is visualized. It is colored now because the data is continuous:



There are still some lines that seem to be more blue or more red. The reason is that only the top movies are visualizing. It is already checked that the average rating is 0 for each user.

Binarizing data

Some recommendation models work on binary data, so it might be useful to binarize the data, that is, define a table containing only 0s and 1s. The 0s will be either treated as missing values or as bad ratings.

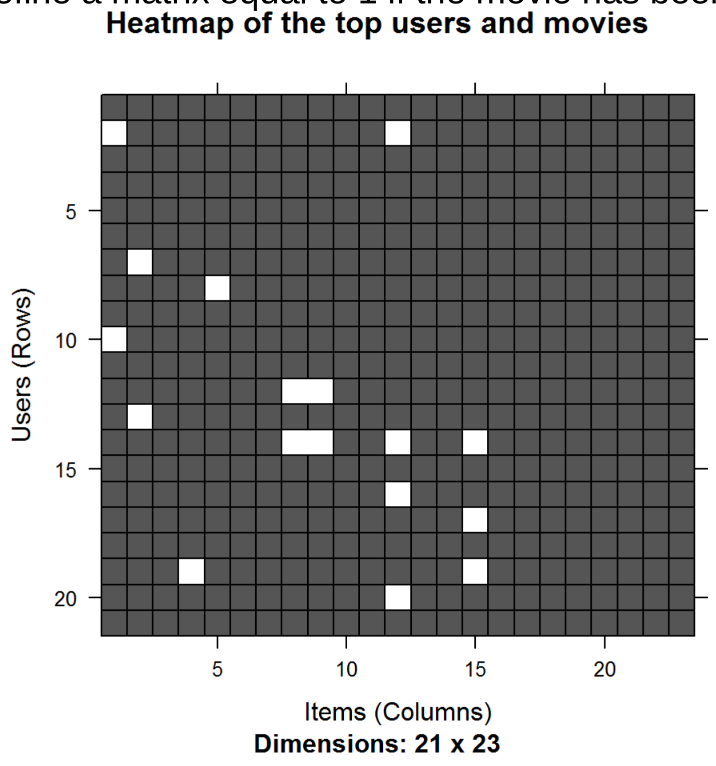
In our case, It can be either:

- Define a matrix having 1 if the user rated the movie, and 0 otherwise. In this case, the information about the rating is lost.
- Define a matrix having 1 if the rating is above or equal to a definite threshold (for example, 3), and 0 otherwise. In this case, giving a bad rating to a movie is equivalent to not having rated it.

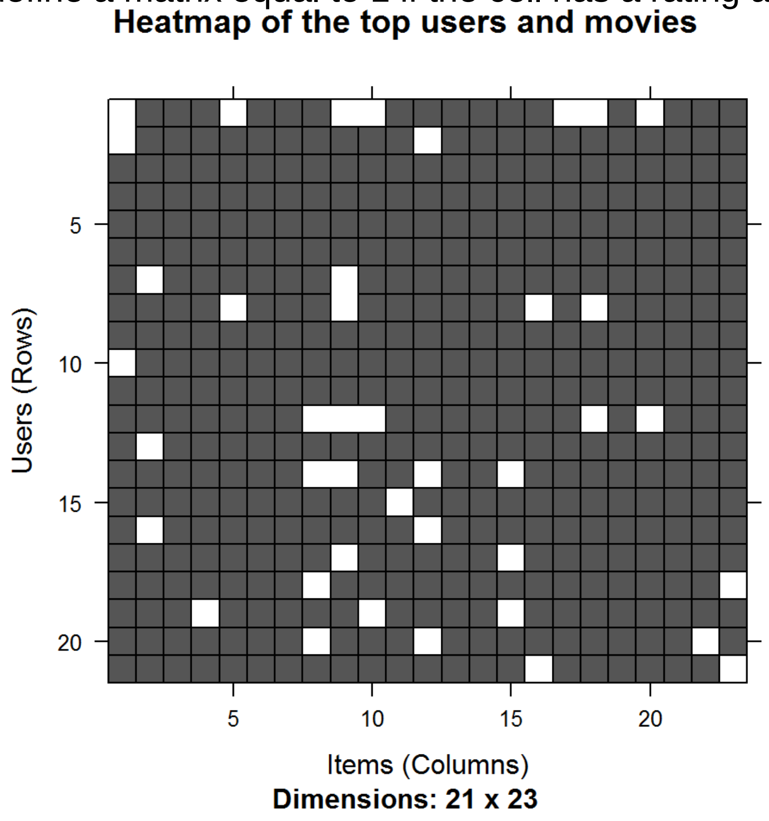
Depending on the context, one choice may be more appropriate than the other.

As a next step, two matrices are defined following the two different approaches and visualize a 5 percent portion of each of binarized matrices.

1st option: define a matrix equal to 1 if the movie has been watched



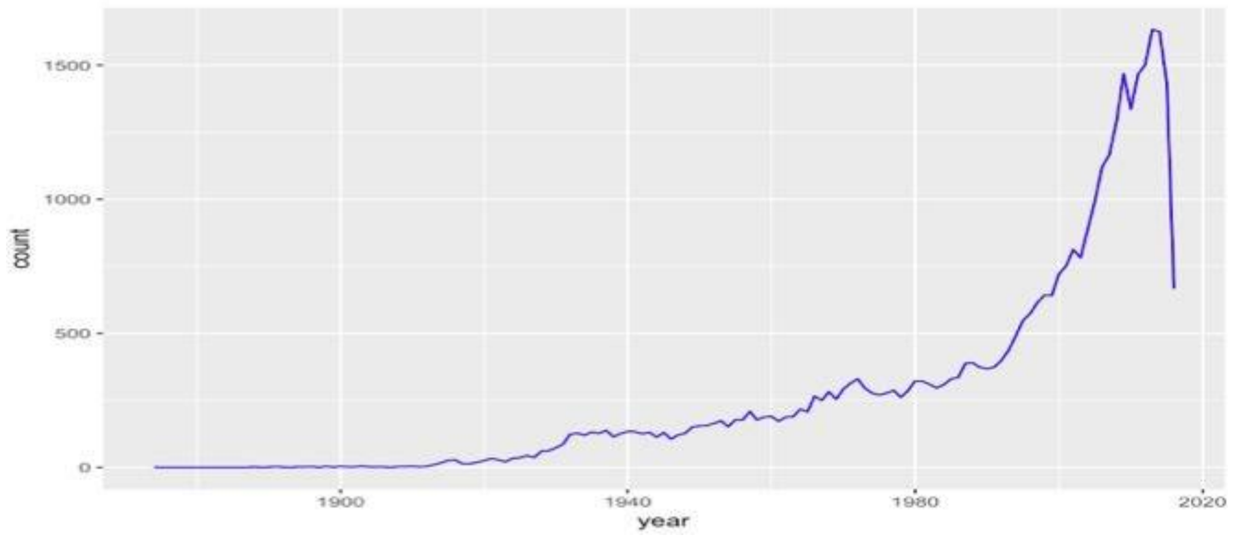
2nd option: define a matrix equal to 1 if the cell has a rating above the threshold



There are more white cells in the second heatmap, which shows that there are more movies with no or bad ratings than those that were not watched by raters.

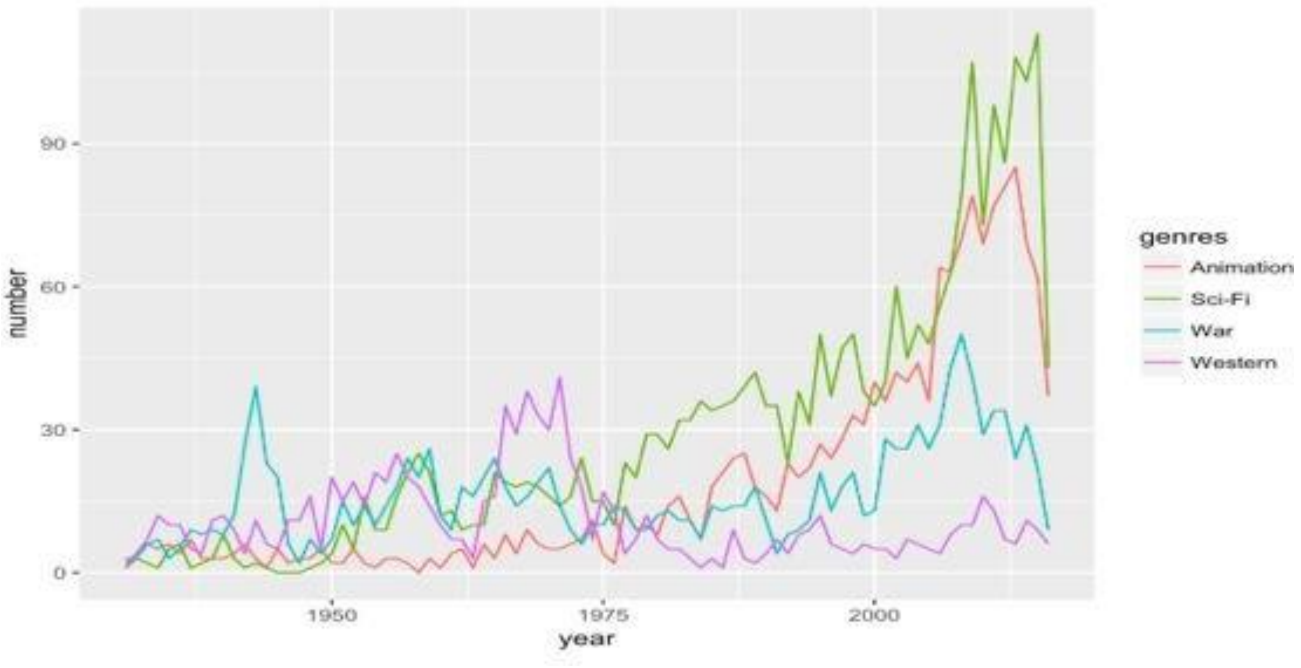
Exploratory Data Analysis:

Descriptive Statistics

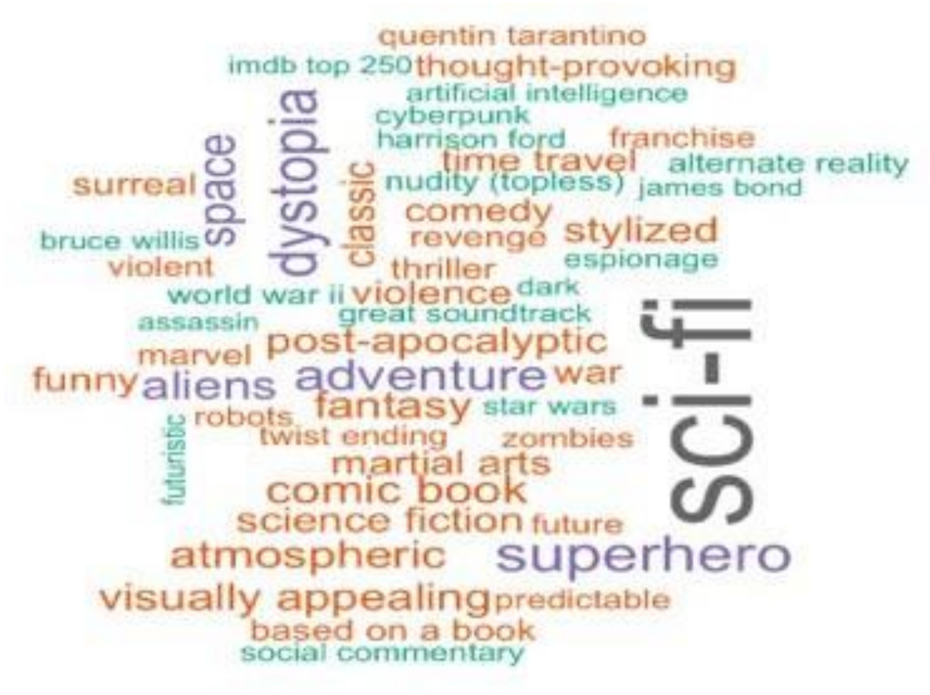


It is seen an exponential growth of the movie business and a sudden drop in 2016 The latter is caused by the fact that the data is collected until October 2016 so we don't have the full data on this year. Growing popularity of the Internet must have had a positive impact on the demand for movies. That is certainly something worthy of further analysis.

These movies are from different genres like Comedy, Romance, Super Hero, Sci-Fi, Animation, War and so on.



We can see that Sci-Fi movies have been increasing exponentially as compared to other genres. Animation movies have also increased in the last 20 years which means that people love enjoying these kind of genres. War movies were popular around the time when big military conflicts occurred - World War II, Vietnam War and most recently War in Afghanistan and Iraq. It's interesting to see how the world of cinematography reflected the state of the real world.



We can clearly see from the word cloud that Sci-fi, superhero, space, dystopia and social commentary are some of the most occurring genres in our dataset.

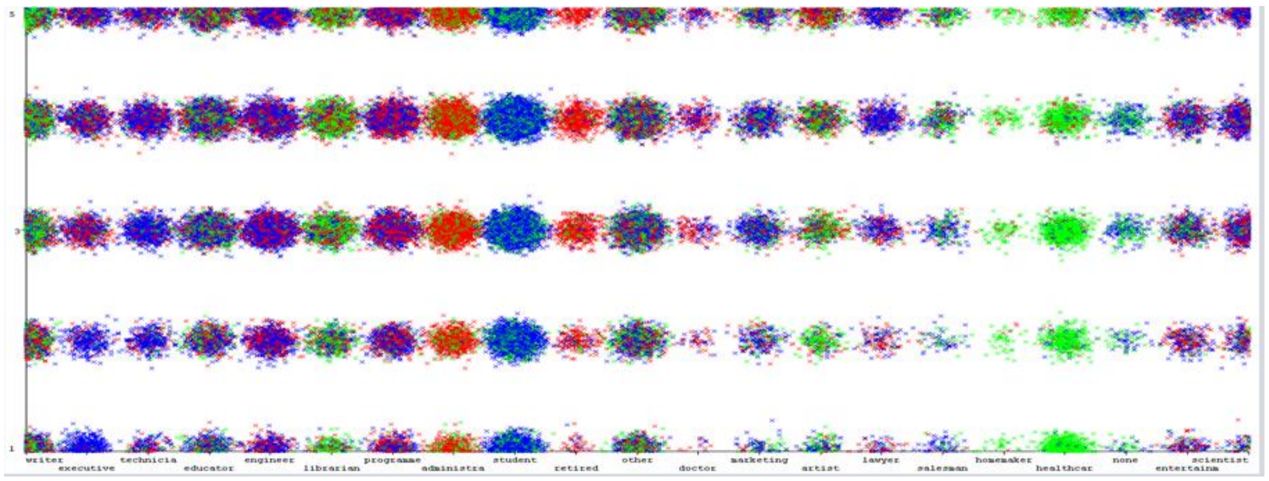
Data Analysis

In this part, we perform K-Means Clustering, Item-Based Collaborative Filtering and User-Based Collaborative Filtering models.

K-Means Clustering: A cluster refers to a collection of data points aggregated together because of certain similarities.

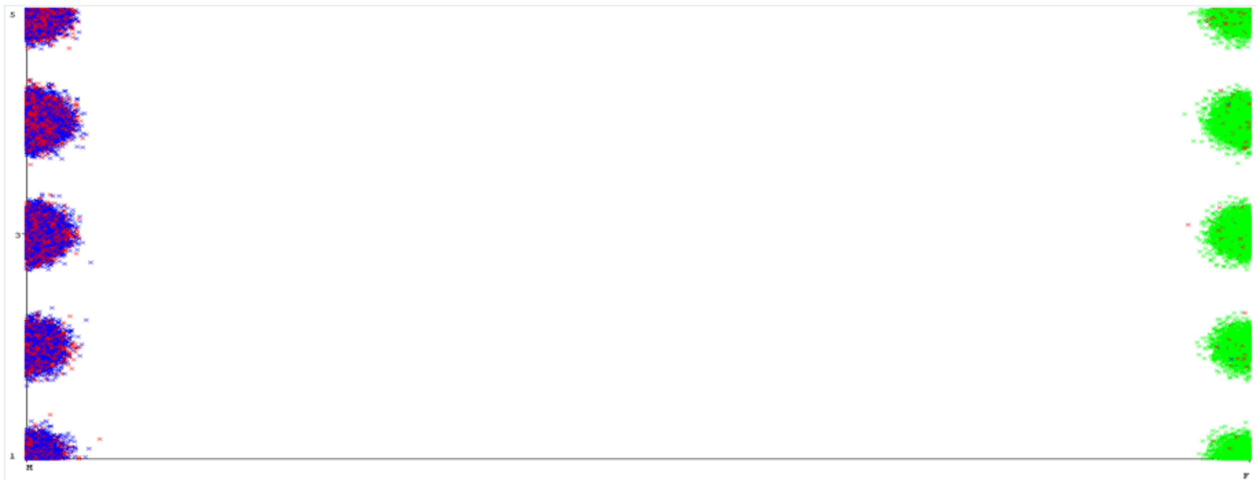
Here we perform k-means clustering to identify if there are any set of users who have similar characteristics in terms of rating a movie or depending on sex. We perform k-means based on occupation to rating ,sex to rating and age to rating.

Occupation to Rating



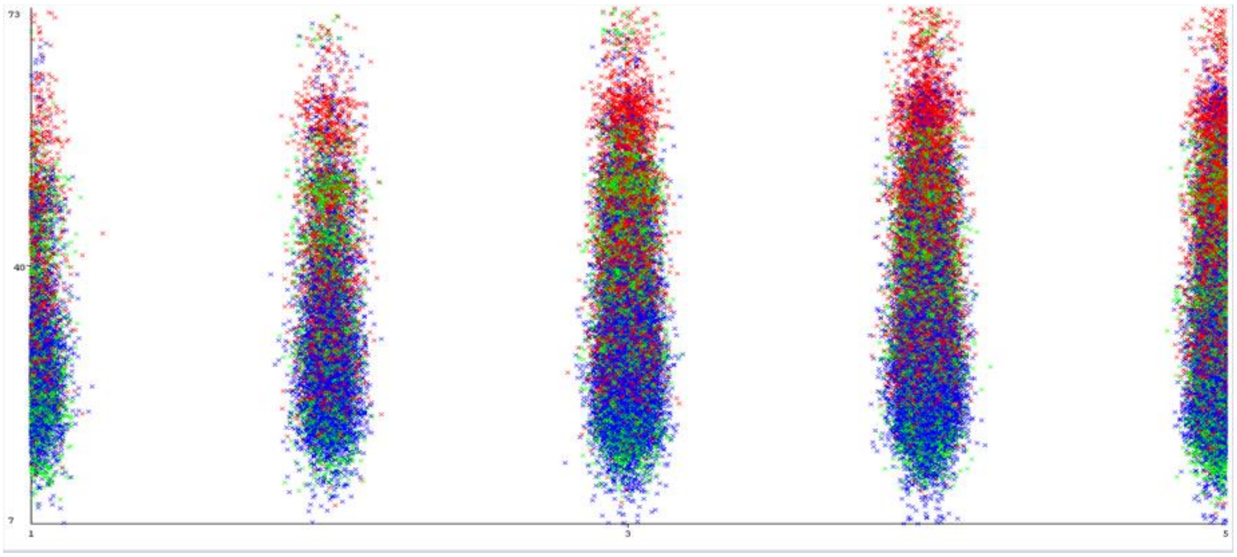
Some occupations such as retired people and doctors are more likely to give high ratings. We can see that the red circle and purple circle increases in intensity as we move up on the y-axis for retired and doctors.

Sex to Rating



Sex difference nearly makes no apparent effect on ratings which means that there are no clusters based on sex.

Age to Rating



Those people(age>40) are inclined to rate higher than those younger people.

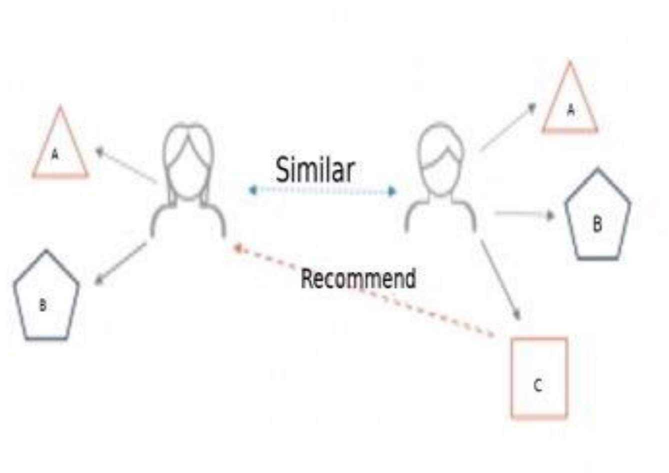
Recommendation System:

There are two kinds of models that we have implemented to build our recommendation system. Item based collaborative filtering and user based collaborative filtering. We have used Pearson and Cosine as the distance function to calculate the similarity between the items.

Pearson Correlation - Measures the association between the users and movies based on different aspects like ratings and genre.

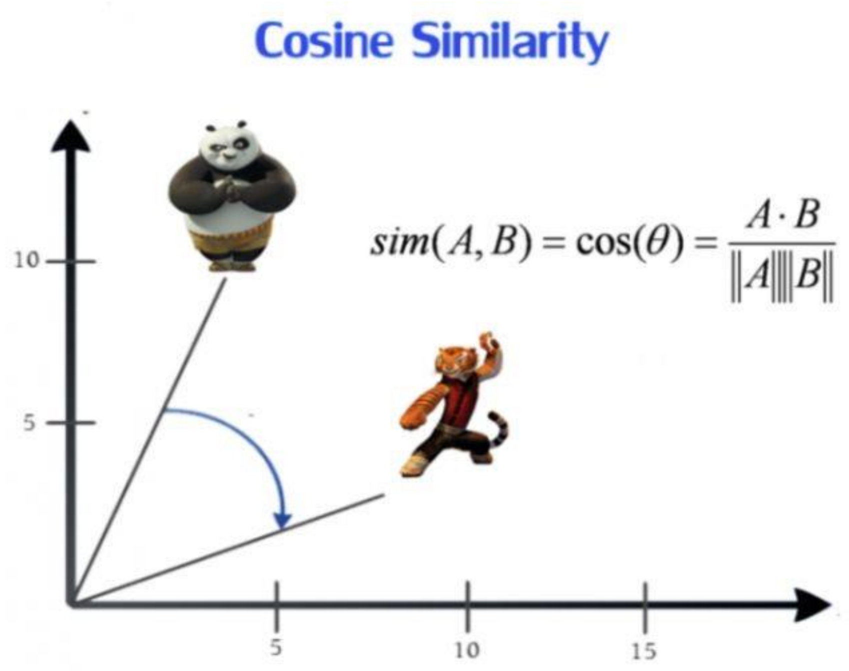
The correlation is a numerical values between -1 and 1 that indicates how much two variables are related to each other. Correlation = 0 means no correlation, while >0 is positive correlation and <0 is negative correlation.

$$\rho_{X,Y} = \mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$



Cosine Similarity - Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them. A simple understanding of this phenomenon:

The cosine similarity is the dot product of two vectors divided by the product of the magnitude of each vector. The reason we divide the dot product by the magnitude is because we are measuring only angle



difference and the dot product is taking the angle difference and magnitude into account. The cosine of a 0 degree angle is 1, therefore the closer to 1 the cosine similarity is the more similar the items are.

ITEM-based Collaborative Filtering Model

Collaborative filtering is a branch of recommendation that takes account of the information about different users. The word “collaborative” refers to the fact that users collaborate with each other to recommend items. In fact, the algorithms take account of user ratings and preferences.

The starting point is a rating matrix in which rows correspond to users and columns correspond to items. The core algorithm is based on these steps:

1. For each two items, measure how similar they are in terms of having received similar ratings by similar users
2. For each item, identify the k most similar items
3. For each user, identify the items that are most similar to the user’s purchases

Defining training/test sets

The model is built using 80% of the whole dataset as a training set, and 20% - as a test set.

Building the recommendation model

Let’s have a look at the default parameters of IBCF model. Here, *k* is the number of items to compute the similarities among them in the first step. After, for each item, the algorithm identifies its *k* most similar items and stores the number. *method* is a similarity function, which is *Cosine* by default, may also be *pearson*. The model is created using the default parameters of method = Cosine and k=30.

```
##  $k
## [1] 30
##
## $method
## [1] "Cosine"
##
## $normalize
## [1] "center"
##
## $normalize_sim_matrix
## [1] FALSE
##
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE
```

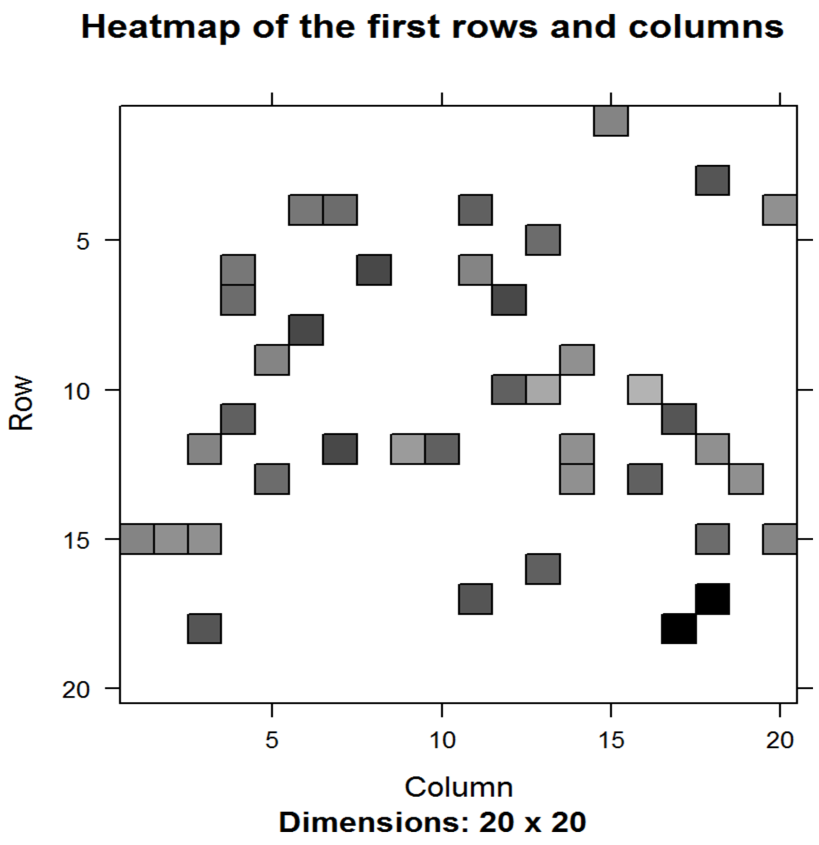
```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 334 users.
```

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

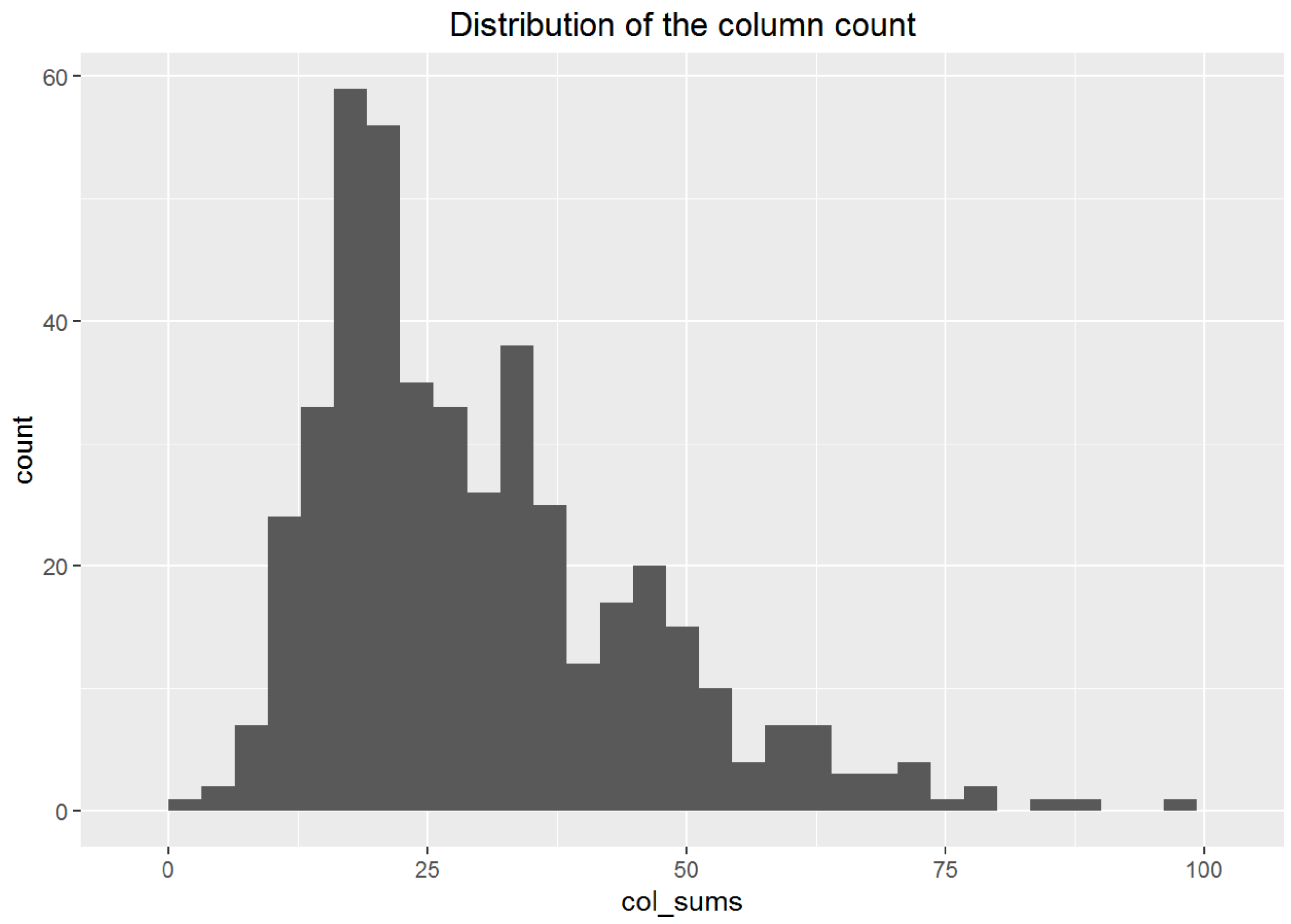
Exploring the recommender model:

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

```
## [1] 447 447
```



```
## row_sums
##    30
##  447
```



dgCMatrix is a similarity matrix created by the model. Its dimensions are 447 x 447, which is equal to the number of items. The heatmap of 20 first items show that many values are equal to 0. The reason is that each row contains only k (30) elements that are greater than 0. The number of non-null elements for each column depends on how many times the corresponding movie was included in the top k of another movie. Thus, the matrix is not necessarily symmetric, which is also the case in our model.

The chart of the distribution of the number of elements by column shows there are a few movies that are similar to many others.

Applying recommender system on the dataset:

Now, it is possible to recommend movies to the users in the test set. *n_recommended* is defind, 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 86 users.
```

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

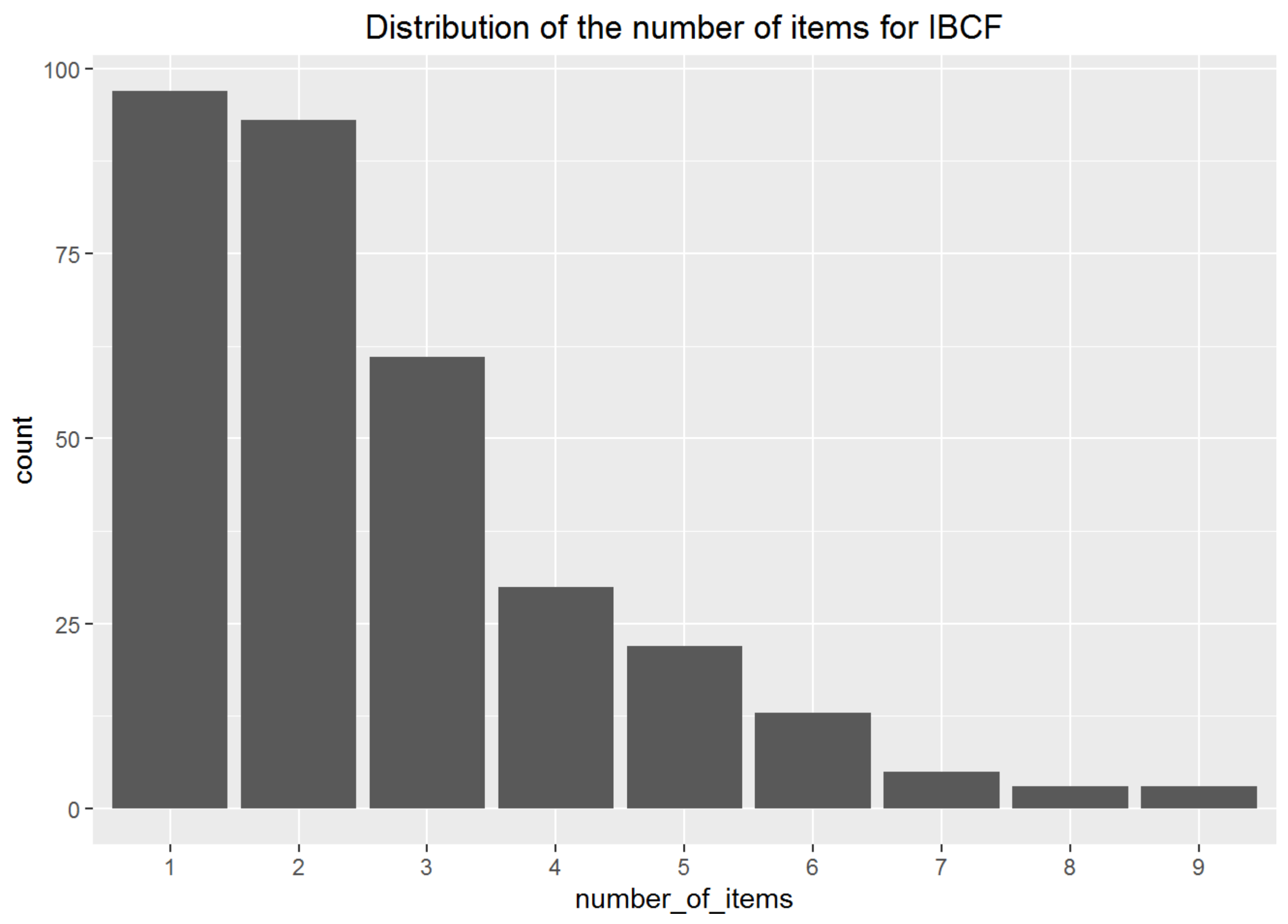
```
##   [1] "Grumpier Old Men (1995)"
##   [2] "Heat (1995)"
##   [3] "Seven (a.k.a. Se7en) (1995)"
##   [4] "Happy Gilmore (1996)"
##   [5] "Rumble in the Bronx (Hont faan kui) (1995)"
##   [6] "Birdcage, The (1996)"
##   [7] "Casper (1995)"
##   [8] "Congo (1995)"
##   [9] "Star Wars: Episode IV - A New Hope (1977)"
##  [10] "Natural Born Killers (1994)"
```

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

```
##       [,1]  [,2] [,3] [,4]
##  [1,]    3 58559  253 5060
##  [2,]    6  6016  318 1090
##  [3,]   47 54286 1729 3052
##  [4,]  104  3147 2329 1183
##  [5,]  112 49272 2571 2542
##  [6,]  141 72998 2761 4011
##  [7,]  158  2542 2959 2019
##  [8,]  160 55820 3081 1408
##  [9,]  260  3578 3147 2947
## [10,]  288  5418 3578 4720
```

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
##  903                        Vertigo  (1958)           9
##  908    North  by  Northwest (1959)           9
## 1203              12 Angry Men (1957)           9
## 36          Dead  Man  Walking  (1995)           8
```

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

IBCF recommends items on the basis of the similarity matrix. It’s an eager-learning model, that is, once it’s built, it doesn’t need to access the initial data. For each item, the model stores the k-most similar, so the amount of information is small once the model is built. This is an advantage in the presence of lots of data.

In addition, this algorithm is efficient and scalable, so it works well with big rating matrices.

USER-based Collaborative Filtering Model

Now, I will use the user-based approach. According to this approach, given a new user, its similar users are first identified. Then, the top-rated items rated by similar users are recommended.

For each new user, these are the steps:

1. Measure how similar each user is to the new one. Like IBCF, popular similarity measures are correlation and cosine.
2. Identify the most similar users. The options are:

- Take account of the top k users (k-nearest_neighbors)
- Take account of the users whose similarity is above a defined threshold

3. Rate the movies rated by the most similar users. The rating is the average rating among similar users and the approaches are:

- Average rating
- Weighted average rating, using the similarities as weights

4. Pick the top-rated movies.

Building the recommendation system:

Again, let’s first check the default parameters of UBCF model. Here, *nn* is a number of similar users, and ‘*method’* is a similarity function, which is *cosine* by default. A recommender model is built leaving the parameters to their defaults and using the training set.

```
## $method
## [1] "cosine"
##
## $nn
## [1] 25
##
## $sample
## [1] FALSE
##
## $normalize
## [1] "center"
```

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 334 users.
```

```
## 334 x 447 rating matrix of class 'realRatingMatrix' with 30135 ratings.
## Normalized using center on rows.
```

Applying the recommender model on the test set

In the same way as the IBCF, It determine the top ten recommendations for each new user in the test set.

```
## Recommendations as 'topNList' with n = 10 for 86 users.
```

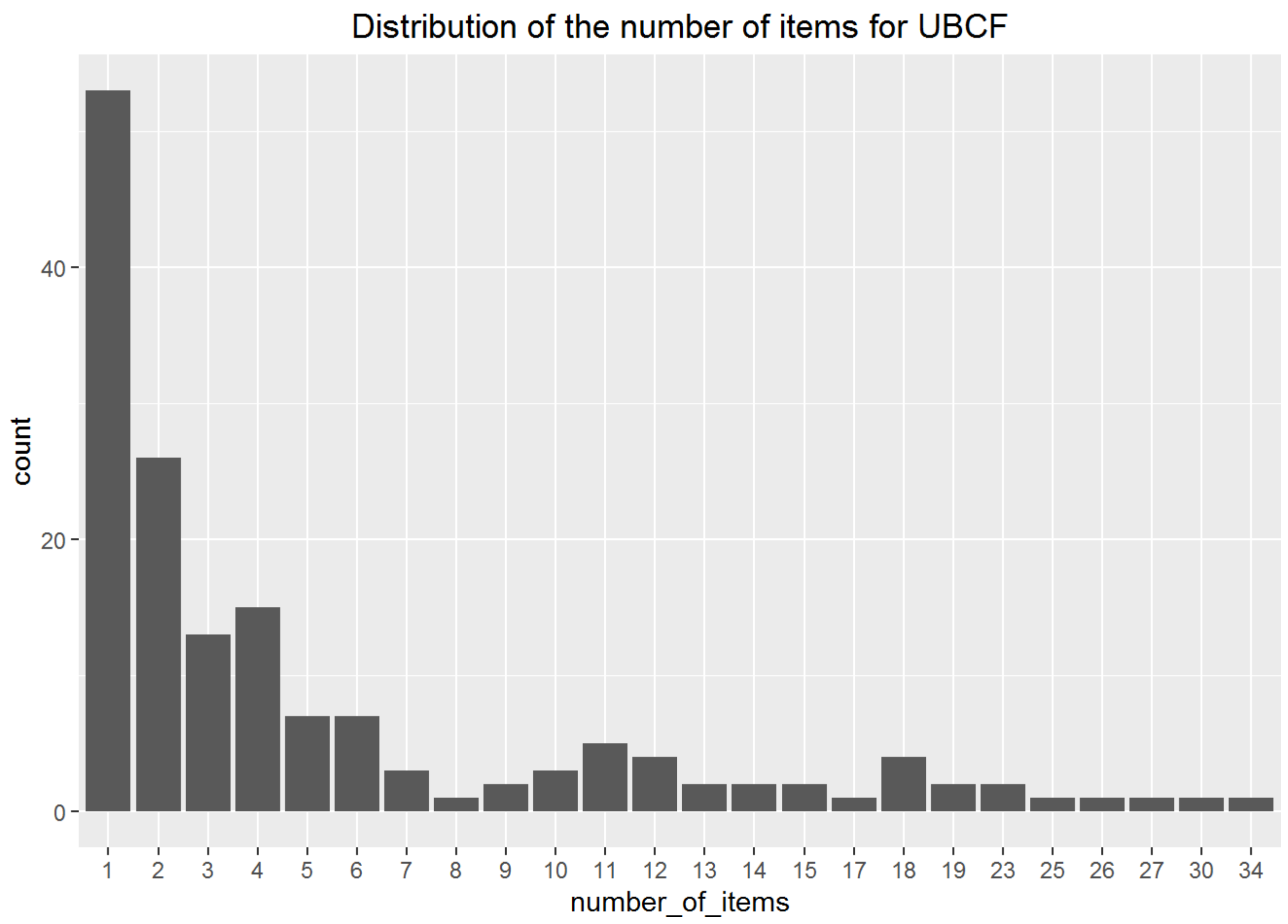
Explore results

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

```
##       [,1]  [,2] [,3]  [,4]
##  [1,]   50   318  318  2329
##  [2,]  296  2959   50  2959
##  [3,]  318   593  593  2858
##  [4,]  356   110  858  5995
##  [5,]  858   551  260  1136
##  [6,]   47  2858  110  4995
##  [7,]  293  2571 1198  4878
##  [8,] 1221  4993 4993  1200
##  [9,] 4993  1258  527  5989
## [10,] 2571 58559 2858 44191
```

The above matrix contain *movieId* of each recommended movie (rows) for the first four users (columns) in the test dataset.

A computation is done how many times each movie got recommended and build the related frequency histogram:



Compared with the IBCF, 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
## 318  Shawshank  Redemption, The  (1994)              34
## 50          Usual  Suspects,  The  (1995)            30
## 858              Godfather,  The  (1972)             27
## 527           Schindler's  List  (1993)              26
```

Comparing the results of UBCF with IBCF helps find some useful insight on different algorithms. UBCF needs to access the initial data. Since it needs to keep the entire database in memory, it doesn't work well in the presence of a big rating matrix. Also, building the similarity matrix requires a lot of computing power and time.

However, UBCF's accuracy is proven to be slightly more accurate than IBCF (I will also discuss it in the next section), so it's a good option if the dataset is not too big.

Evaluating the Recommender Systems

There are a few options to choose from when deciding to create a recommendation engine. In order to compare their performances and choose the most appropriate model, I follow these steps:

- Prepare the data to evaluate performance
- Evaluate the performance of some models
- Choose the best performing models
- Optimize model parameters

Preparing the data to evaluate models

We need two training and testing data to evaluate the model. There are several methods to create them: 1) splitting the data into training and test sets, 2) bootstrapping, 3) using k-fold.

Splitting the data

Splitting the data into training and test sets is often done using a 80/20 proportion.

For each user in the test set, we need to define how many items to use to generate recommendations. For this, first it is checked that the minimum number of items rated by users to be sure there will be no users with no items to test.

```
min(rowCounts(ratings_movies))
```

```
## [1] 8
```

```
items_to_keep <- 5 #number of items to generate recommendations
rating_threshold <- 3 # threshold with the minimum rating that is considered good
n_eval <- 1 #number of times to run evaluation

eval_sets <- evaluationScheme(data = ratings_movies,
                              method = "split",
                              train = percentage_training,
                              given = items_to_keep,
                              goodRating = rating_threshold,
                              k = n_eval)
eval_sets
```

```
## Evaluation scheme with 5 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.800
## Good ratings: >=3.000000
## Data set: 420 x 447 rating matrix of class 'realRatingMatrix' with 38341 ratings.
```

```
getData(eval_sets, "train") # training set
```

```
## 336 x 447 rating matrix of class 'realRatingMatrix' with 31157 ratings.
```

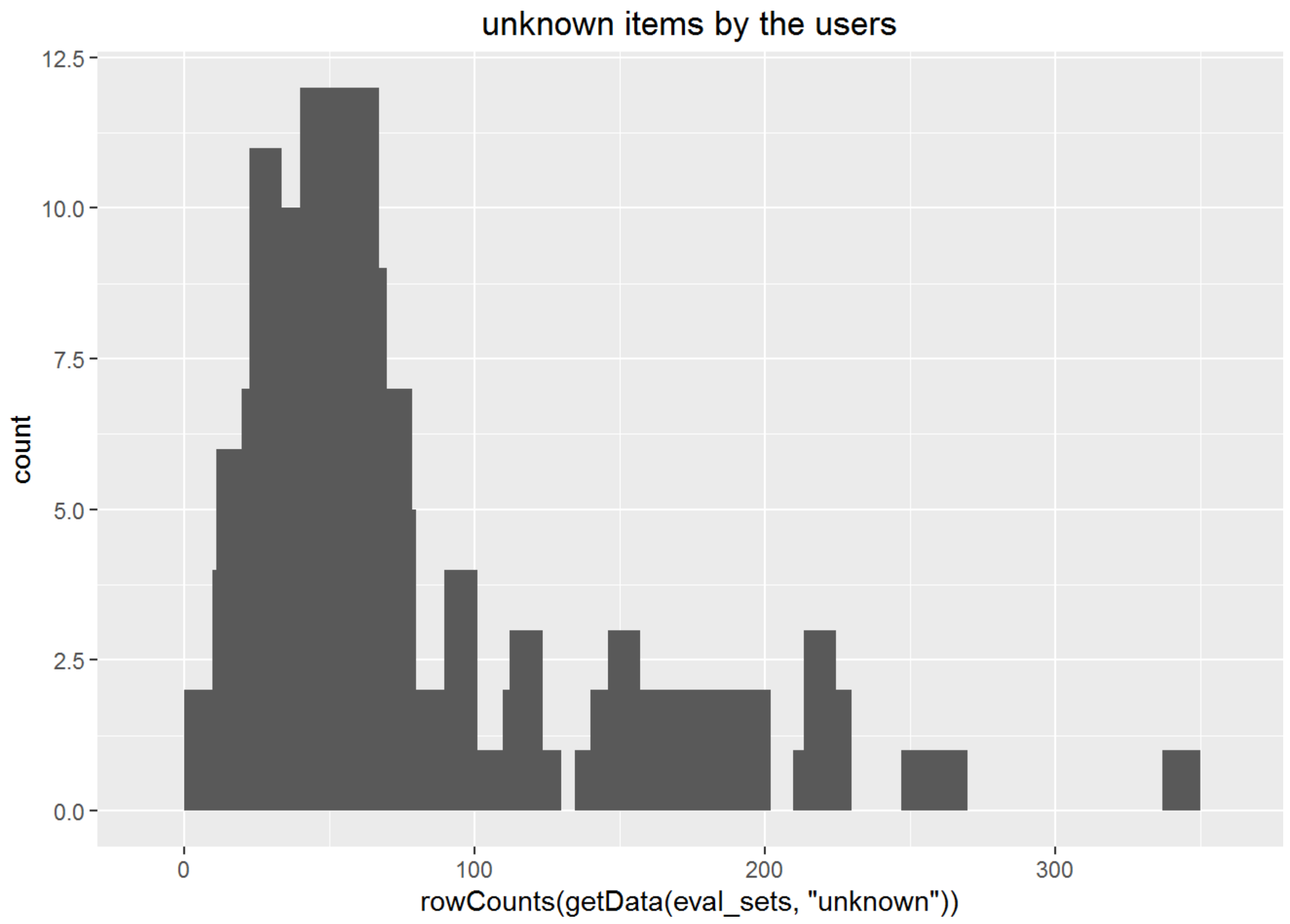
```
getData(eval_sets, "known") # set with the items used to build the recommendations
```

```
## 84 x 447 rating matrix of class 'realRatingMatrix' with 420 ratings.
```

```
getData(eval_sets, "unknown") # set with the items used to test the recommendations
```

```
## 84 x 447 rating matrix of class 'realRatingMatrix' with 6764 ratings.
```

```
qplot(rowCounts(getData(eval_sets, "unknown"))) +
  geom_histogram(binwidth = 10) +
  ggtitle("unknown items by the  users")
```



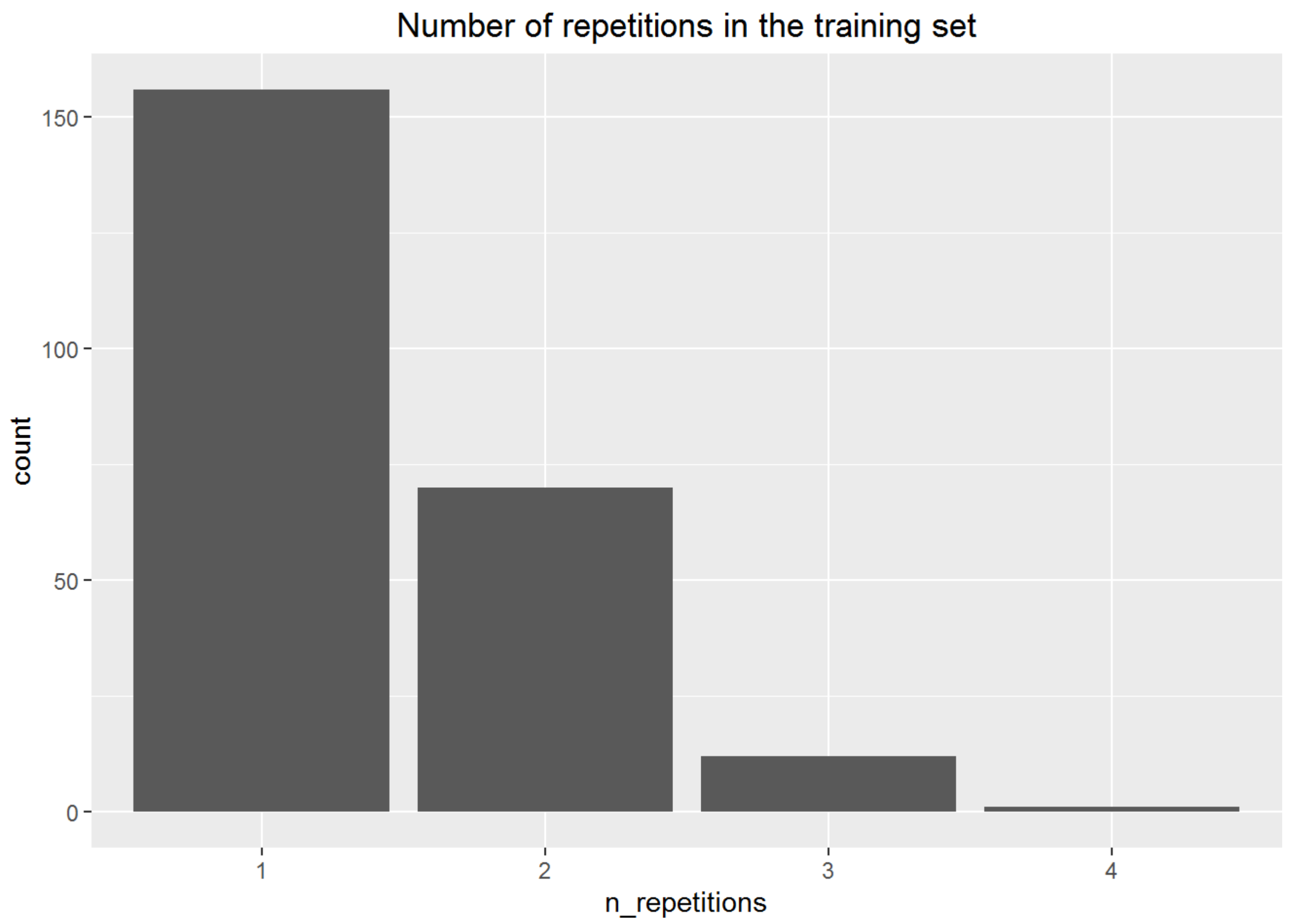
The above image displays the unknown items by the users, which varies a lot.

Bootstrapping the data

Bootstrapping is another approach to split the data. The same user can be sampled more than once and, if the training set has the same size as it did earlier, there will be more users in the test set.

```
eval_sets <- evaluationScheme(data = ratings_movies,
                              method = "bootstrap",
                              train = percentage_training,
                              given = items_to_keep,
                              goodRating =  rating_threshold,
                              k = n_eval)

table_train <- table(eval_sets@runsTrain[[1]])
n_repetitions <- factor(as.vector(table_train))
qplot(n_repetitions) +
  ggtitle("Number of repetitions in the training set")
```



The above chart shows that most of the users have been sampled fewer than four times.

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
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
```

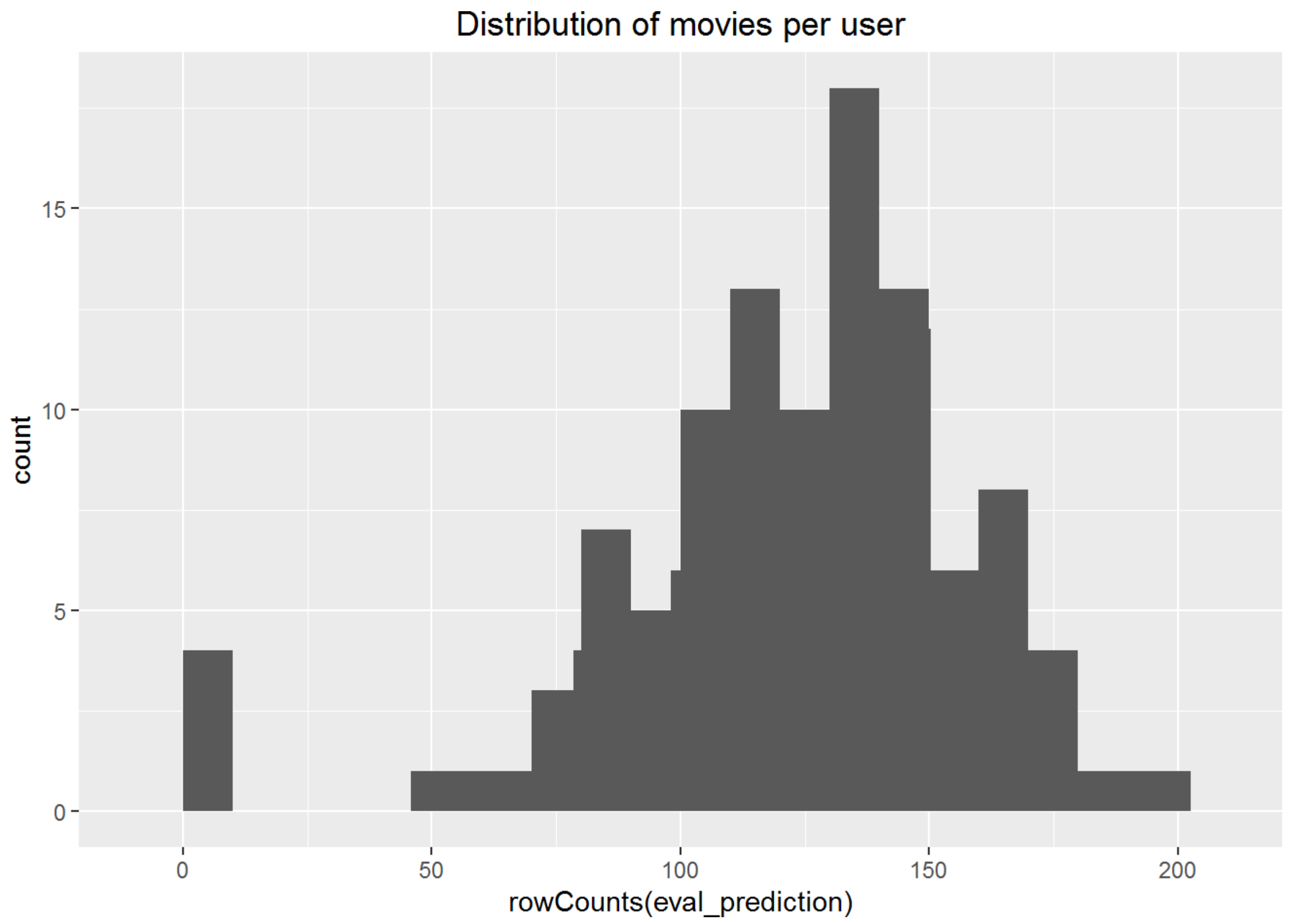
```
## [1] 315 315 315 315
```

Using 4-fold approach, we get four sets of the same size 315.

Evaluating the ratings

k-fold approach is used for evaluation.

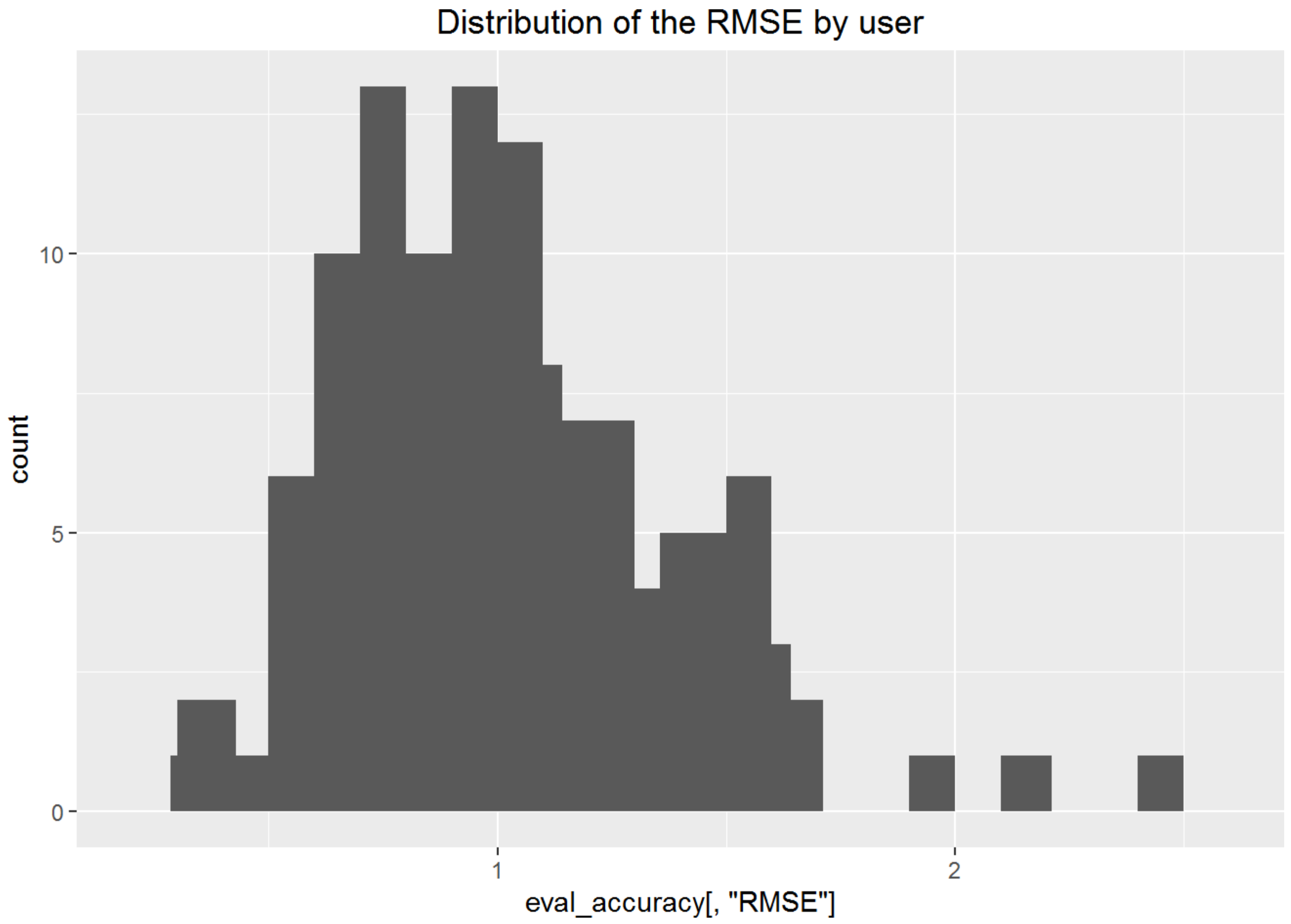
First, re-defined the evaluation sets, build IBCF model and create a matrix with predicted ratings.



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,] 0.6002487 0.3602984 0.3777928
## [2,] 0.7694427 0.5920421 0.5683099
## [3,] 1.1223170 1.2595954 0.9067505
## [4,] 1.1058170 1.2228312 0.7860035
## [5,] 0.7942651 0.6308570 0.6723061
## [6,] 0.9013878 0.8125000 0.8750000
```



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

```
##      RMSE       MSE       MAE
## 1.1026305 1.2157940 0.7928826
```

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, It made 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. It used *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  [4.49sec/0.06sec]
##    2  [4.47sec/0.06sec]
##    3  [4.53sec/0.06sec]
##    4  [4.46sec/0.07sec]
```

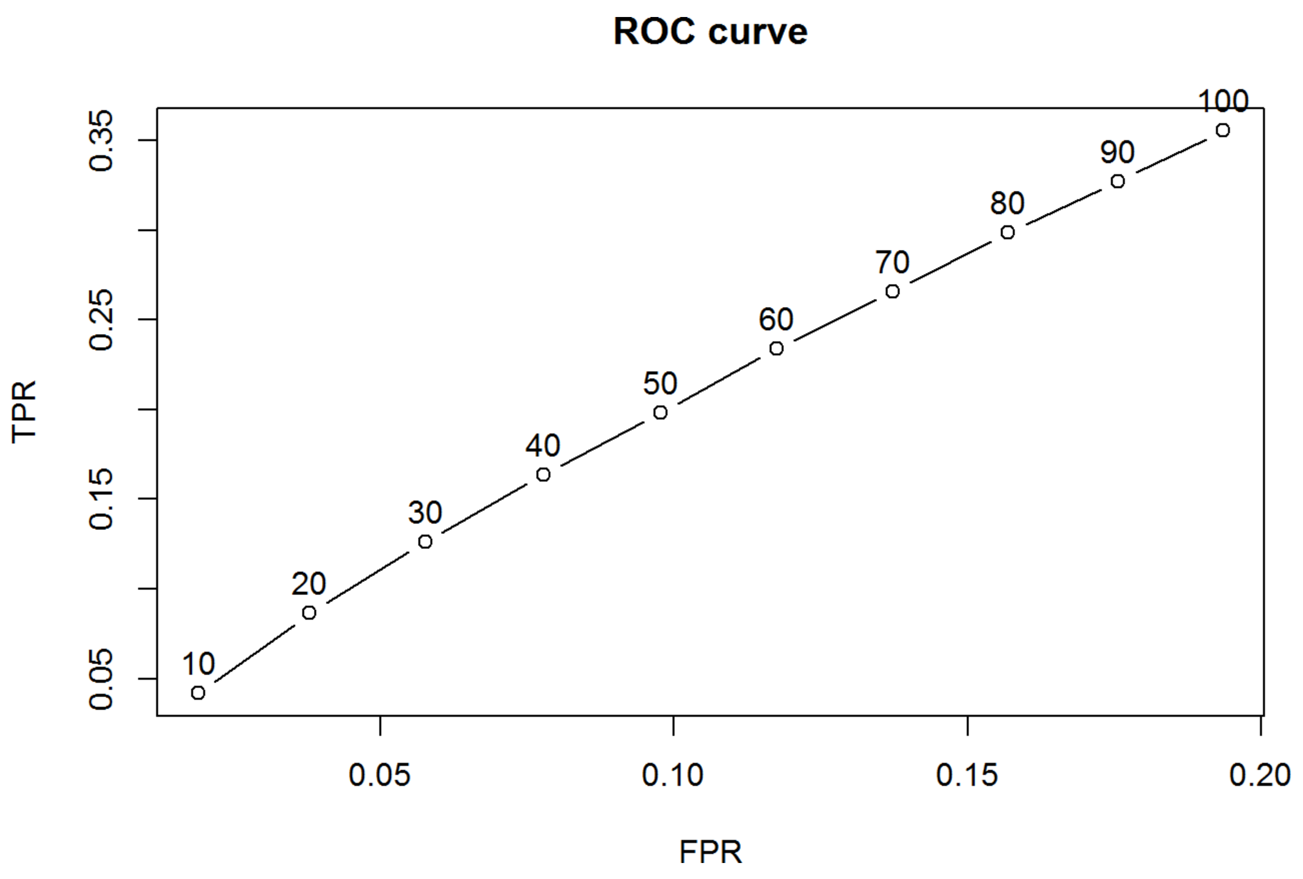
```
##            TP        FP       FN       TN precision     recall        TPR
## 10   2.438095  7.180952 66.63810 365.7429 0.2534653 0.03684398 0.03684398
## 20   4.857143 14.380952 64.21905 358.5429 0.2524752 0.08070848 0.08070848
## 30   6.952381 21.904762 62.12381 351.0190 0.2409241 0.11743856 0.11743856
## 40   9.104762 29.371429 59.97143 343.5524 0.2366337 0.15409146 0.15409146
## 50  11.152381 36.942857 57.92381 335.9810 0.2318812 0.19071582 0.19071582
## 60  13.695238 43.923810 55.38095 329.0000 0.2374257 0.23003707 0.23003707
##            FPR
## 10  0.01909268
## 20  0.03845362
## 30  0.05865530
## 40  0.07869669
## 50  0.09897217
## 60  0.11752674
```

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

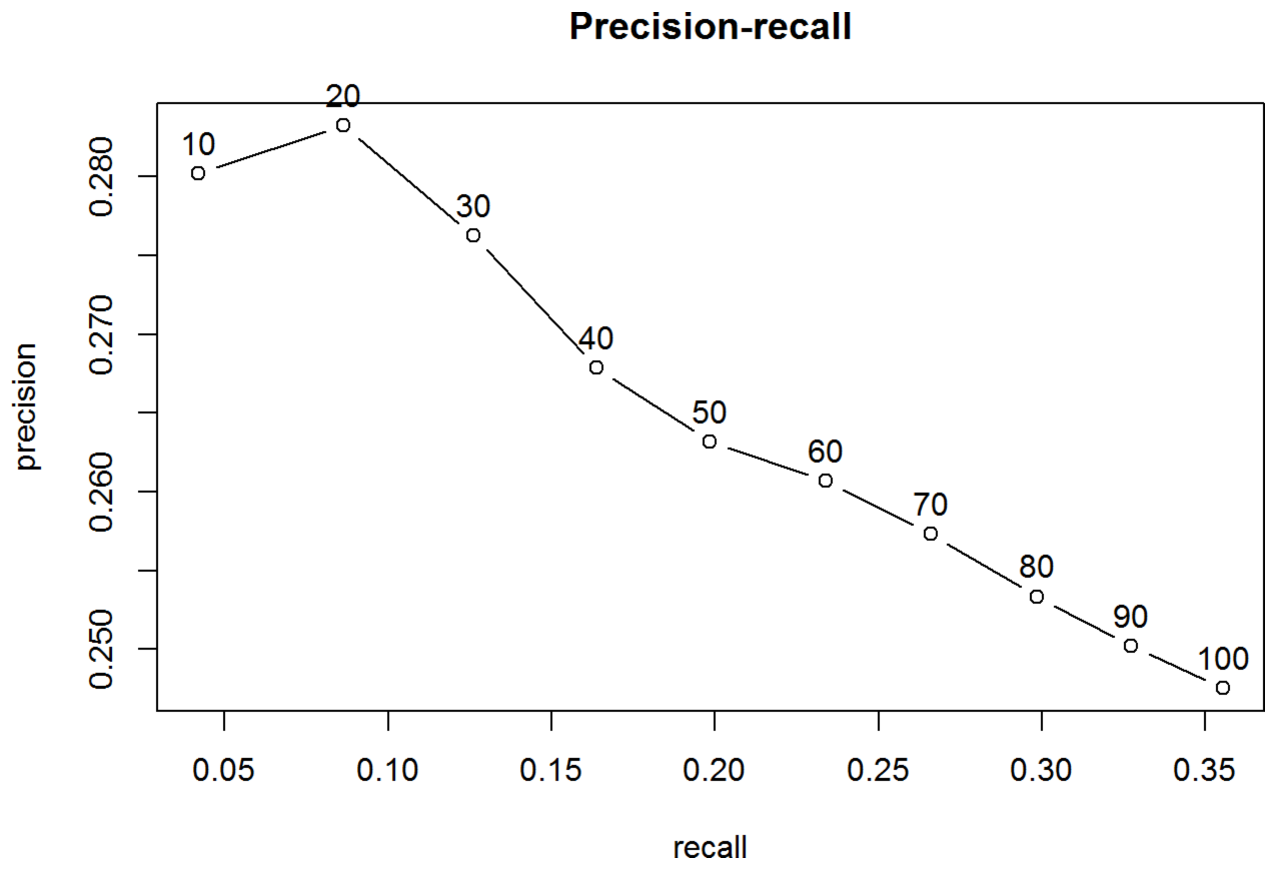
```
##          TP        FP       FN       TN
##  10 10.98095  28.16190 291.2571 1437.600
##  20 22.20000  56.08571 280.0381 1409.676
##  30 32.47619  84.94286 269.7619 1380.819
##  40 41.93333 114.53333 260.3048 1351.229
##  50 51.44762 144.00952 250.7905 1321.752
##  60 61.05714 173.01905 241.1810 1292.743
```

Finally, It plots 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.

Comparing models

In order to compare different models, I define them as a following list:

- Item-based collaborative filtering, using the Cosine as the distance function
- Item-based collaborative filtering, using the Pearson correlation as the distance function
- User-based collaborative filtering, using the Cosine as the distance function
- User-based collaborative filtering, using the Pearson correlation as the distance function
- Random recommendations to have a base line

Then, It defined a different set of numbers for recommended movies (n_recommendations <- c(1, 5, seq(10, 100, 10))), run and evaluate the models:

```
## IBCF run fold/sample  [model  time/prediction  time]
##    1   [4.5sec/0.06sec]
##    2   [4.5sec/0.06sec]
##    3   [4.4sec/0.07sec]
##    4   [4.57sec/0.06sec]
## IBCF run fold/sample  [model  time/prediction  time]
##    1   [8.3sec/0.06sec]
##    2   [8.52sec/0.04sec]
##    3   [8.38sec/0.06sec]
##    4   [8.33sec/0.07sec]
## UBCF run fold/sample [model time/prediction time]
##    1   [0sec/2.25sec]
##    2   [0sec/2.29sec]
##    3   [0sec/2.27sec]
##    4   [0.02sec/2.26sec]
## UBCF run fold/sample [model  time/prediction  time]
##    1   [0.01sec/2.87sec]
##    2   [0sec/2.93sec]
##    3   [0.02sec/2.89sec]
##    4   [0sec/2.84sec]
## RANDOM run fold/sample [model time/prediction time]
##    1   [0sec/0.1sec]
##    2   [0sec/0.09sec]
##    3   [0sec/0.1sec]
##    4   [0sec/0.1sec]
```

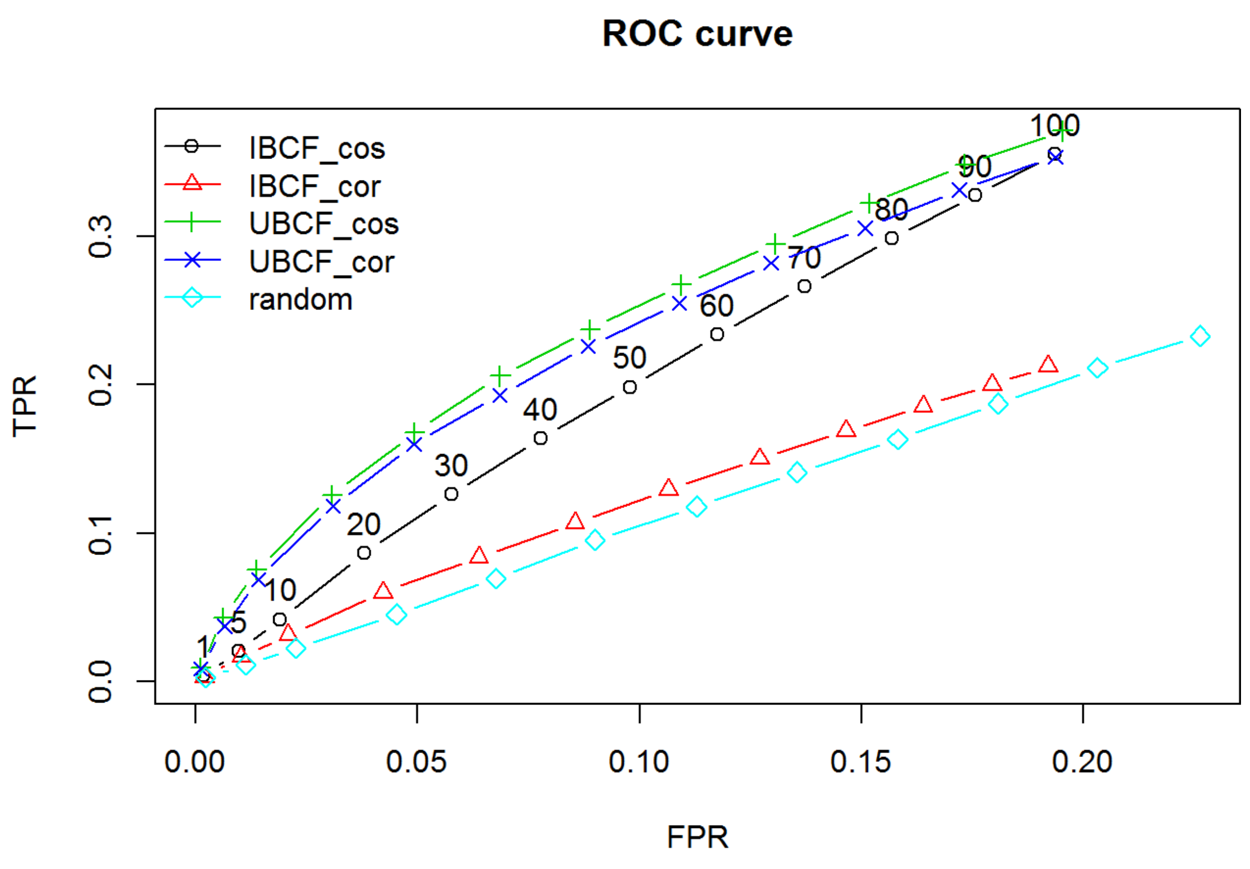
```
## IBCF_cos IBCF_cor UBCF_cos UBCF_cor   random
##     TRUE     TRUE     TRUE     TRUE     TRUE
```

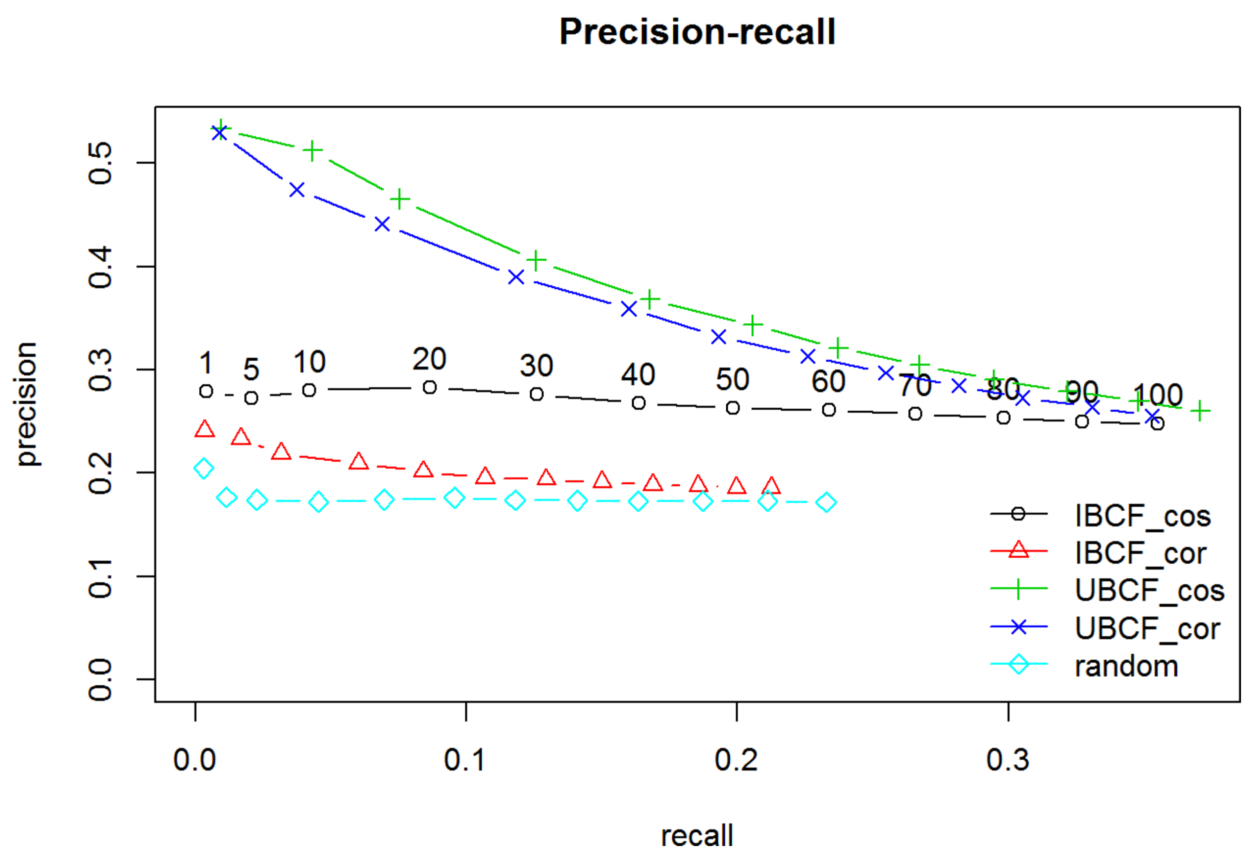
The following table presents as an example the first rows of the performance evaluation matrix for the IBCF with Cosine distance:

```
##    precision      recall         TPR         FPR
## 1  0.2794085 0.004054432 0.004054432 0.001892934
## 5  0.2724670 0.020509942 0.020509942 0.009632347
## 10 0.2802074 0.041911963 0.041911963 0.019024055
## 20 0.2832357 0.086426698 0.086426698 0.038007303
## 30 0.2762620 0.126139933 0.126139933 0.057626777
## 40 0.2678761 0.163710847 0.163710847 0.077791926
```

Identifying the most suitable model

It compares the models by building a chart displaying their ROC curves and Precision/recall curves.


ROC curve
IBCF_cos
IBCF_cor
UBCF_cos
UBCF_cor
random
TPR
FPR
0.0
0.1
0.2
0.3
0.00
0.05
0.10
0.15
0.20
1
5
10
20
30
40
50
60
70
80
90
100


Precision-recall
IBCF_cos
IBCF_cor
UBCF_cos
UBCF_cor
random
precision
recall
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.1
0.2
0.3
1
5
10
20
30
40
50
60
70
80
90
100

A good performance index is the area under the curve (AUC), that is, the area under the ROC curve. Even without computing it, the chart shows that the highest is UBCF with cosine distance, so it's the best-performing technique.

The UBCF with cosine distance is still the top model. Depending on what is the main purpose of the system, an appropriate number of items to recommend should be defined.

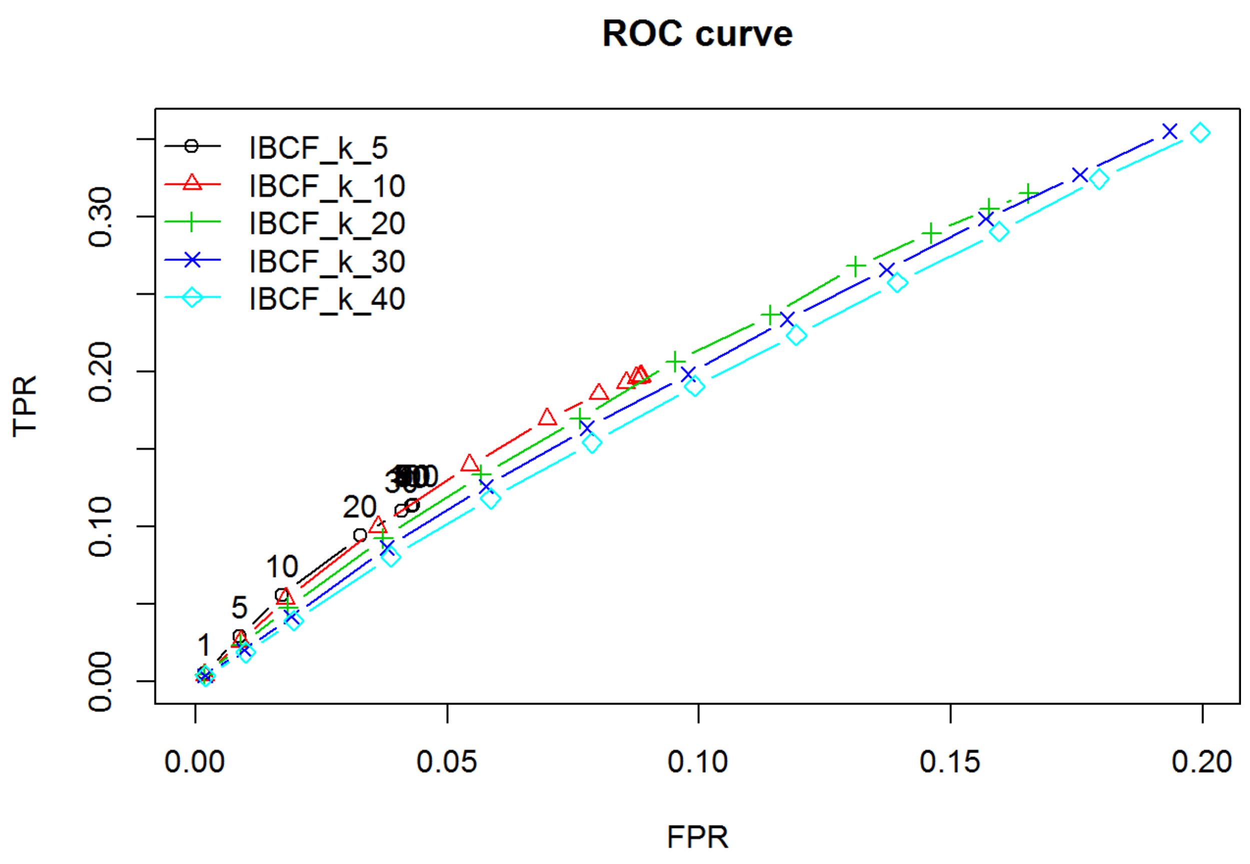
Optimizing a numeric parameter

IBCF takes account of the k-closest items. I will explore more values, ranging between 5 and 40, in order to tune this parameter:

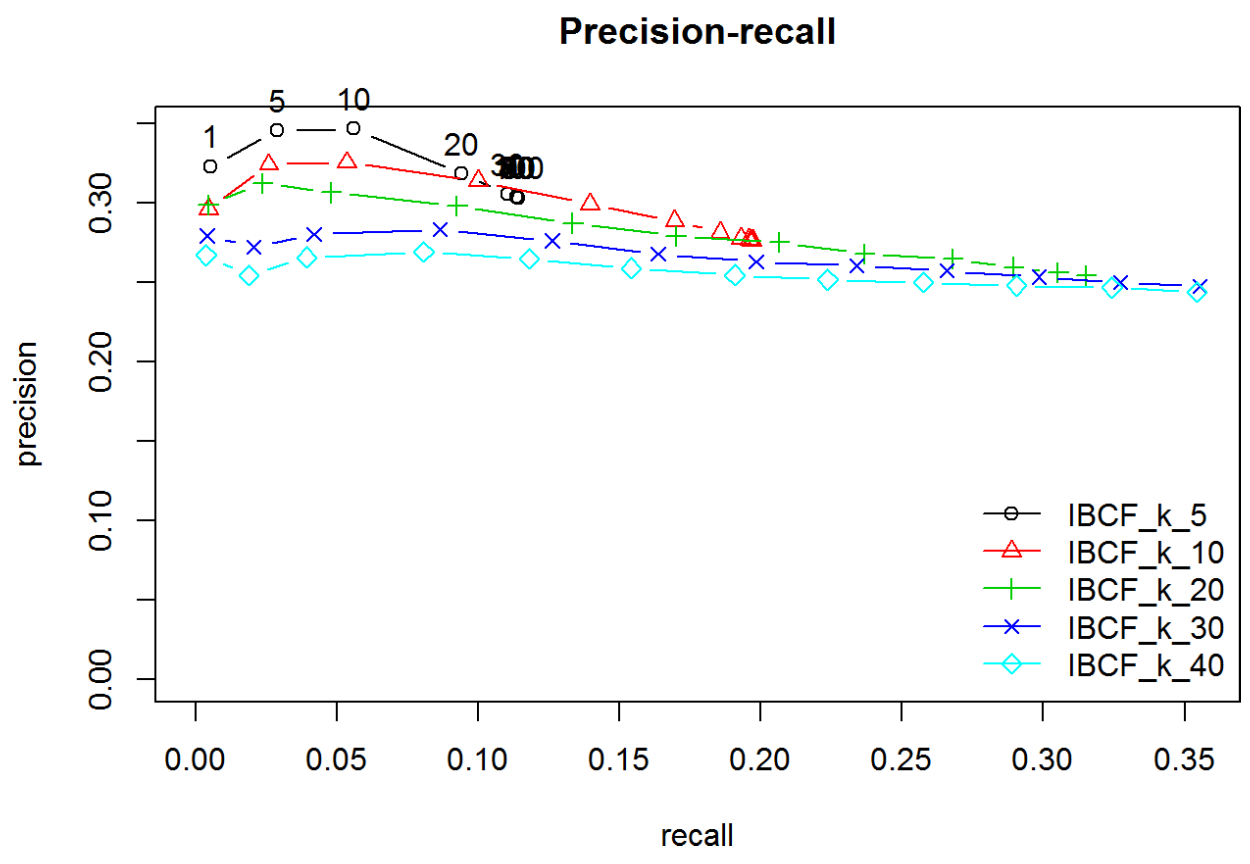
```
vector_k <- c(5, 10, 20, 30, 40)
models_to_evaluate <- lapply(vector_k, function(k){
  list(name = "IBCF",
       param = list(method = "cosine", k = k))
})
names(models_to_evaluate) <- paste0("IBCF_k_", vector_k)
```

Now It built and evaluate the same IBCF/cosine models with different values of the k-closest items:

```
## IBCF run fold/sample [model time/prediction time]
##    1  [4.63sec/0.04sec]
##    2  [4.47sec/0.04sec]
##    3  [4.52sec/0.04sec]
##    4  [4.57sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [4.43sec/0.05sec]
##    2  [4.54sec/0.05sec]
##    3  [4.62sec/0.05sec]
##    4  [4.44sec/0.05sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [4.57sec/0.05sec]
##    2  [4.94sec/0.08sec]
##    3  [4.83sec/0.05sec]
##    4  [4.64sec/0.04sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [4.42sec/0.05sec]
##    2  [4.42sec/0.05sec]
##    3  [4.55sec/0.04sec]
##    4  [4.56sec/0.06sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [4.48sec/0.07sec]
##    2  [4.53sec/0.06sec]
##    3  [4.64sec/0.06sec]
##    4  [4.54sec/0.06sec]
```



ROC curve
IBCF_k_5
IBCF_k_10
IBCF_k_20
IBCF_k_30
IBCF_k_40
TPR
FPR
0.00
0.10
0.20
0.30
0.00
0.05
0.10
0.15
0.20
1
5
10
20
30



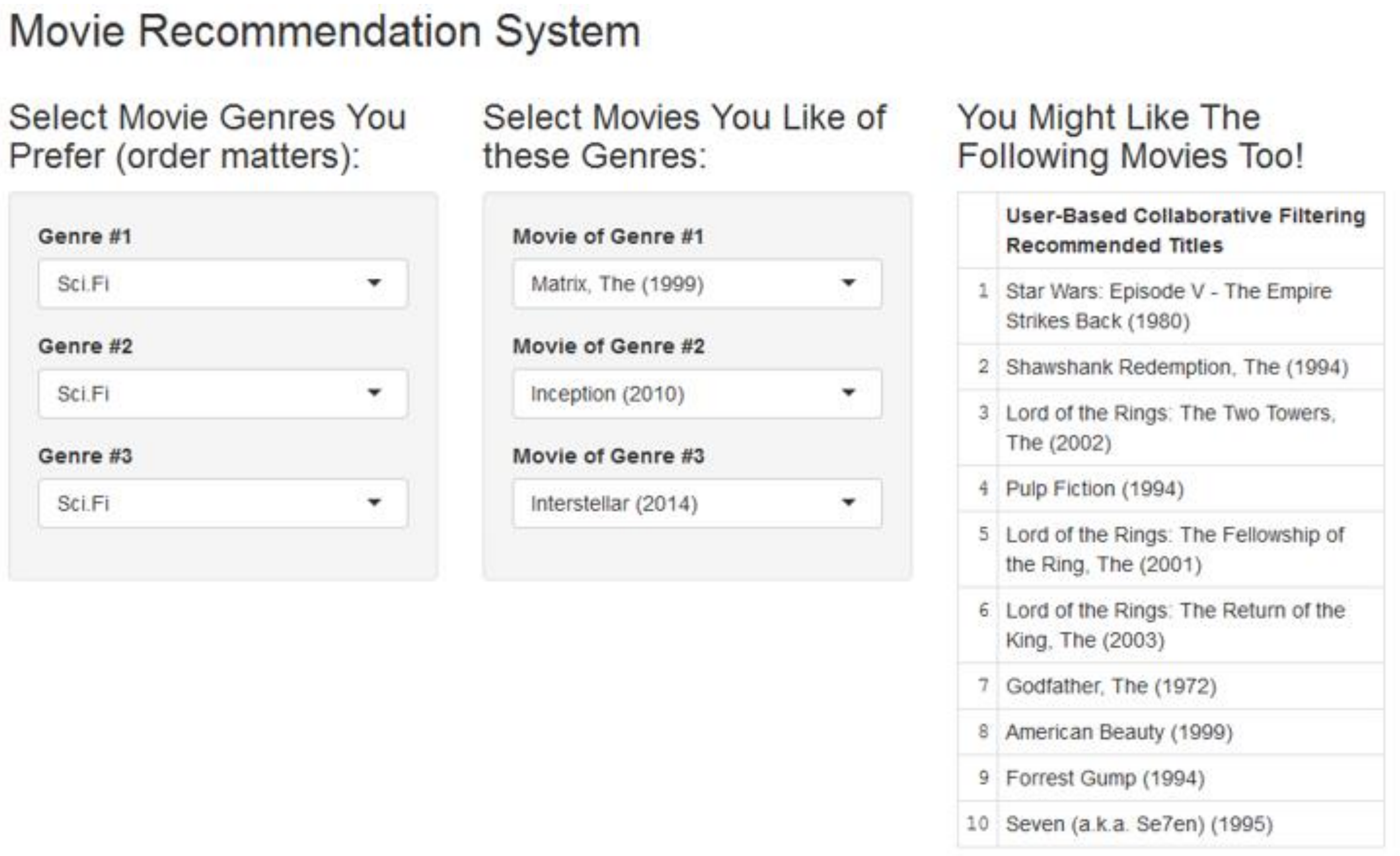
Precision-recall
IBCF_k_5
IBCF_k_10
IBCF_k_20
IBCF_k_30
IBCF_k_40
precision
recall
0.00
0.10
0.20
0.30
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
1
5
10
20
30

Based on the ROC curve's plot, the k having the biggest AUC is 10. Another good candidate is 5, but it can never have a high TPR. This means that, even if we set a very high n value, the algorithm won't be able to recommend a big percentage of items that the user liked. The IBCF with k = 5 recommends only a few items similar to the purchases. Therefore, it can't be used to recommend many items.

Based on the precision/recall plot, k should be set to 10 to achieve the highest recall. If we are more interested in the precision, we set k to 5.

Online App Demonstrating the UBCF Model

I have created a web application for the recommender system using the Shiny package in R, as shown below. The web application is hosted on shinyapps.io.



Movie Recommendation System

Select Movie Genres You Prefer (order matters):

Genre #1: Sci.Fi

Genre #2: Sci.Fi

Genre #3: Sci.Fi

Select Movies You Like of these Genres:

Movie of Genre #1: Matrix, The (1999)

Movie of Genre #2: Inception (2010)

Movie of Genre #3: Interstellar (2014)

You Might Like The Following Movies Too!

	User-Based Collaborative Filtering Recommended Titles
1	Star Wars: Episode V - The Empire Strikes Back (1980)
2	Shawshank Redemption, The (1994)
3	Lord of the Rings: The Two Towers, The (2002)
4	Pulp Fiction (1994)
5	Lord of the Rings: The Fellowship of the Ring, The (2001)
6	Lord of the Rings: The Return of the King, The (2003)
7	Godfather, The (1972)
8	American Beauty (1999)
9	Forrest Gump (1994)
10	Seven (a.k.a. Se7en) (1995)

In this web app, I am presenting a simple recommender system created by the *user-based collaborative approach*. This specific approach was used mainly because it was the best performing method, based on the evaluation performed for this project.

Here the user selected Genre- Action Movie- Hellboy, Genre- Horror Movie- 13 ghosts, Genre- Thriller Movie- 10 to Midnight and our system recommended 10 movies which are the combination of all the 3 genres the user has selected.

Note: the app hosted at Shinyapps.io is running on a free account. That means that it is very restricted in computational resources and may be slow or even not responsible either when a lot of connections are made, or when large files are used.

Challenges

The first challenge we faced in the project is the size of the actual movie lens dataset. The size of the original dataset is 100M which is very difficult to process in R with the kind of operating system we have. Hence, we decided to work on the subset data 100K which is also provided by movie lens website. We perform our analysis and create a recommendation system on this dataset and the same analysis can be applied on the original one of 100M using an advanced computer with high processing power.

The next challenge we faced in the project is the implementation of the R-shiny application to create the user interface for our movie recommendation system. Posting on R-shiny requires a fee to be paid and hence we decided to select the free trial in which we were only allowed to give recommendations based on 1000 movies.

Logically this makes sense because User-based Collaborative Filtering gives recommendations that can be complements to the item the user was interacting with. Whereas, in item based filtering the recommendations we get will likely be direct substitutes, and not complements, of the item the user interacted with. It won’t be effective to have a Item-based recommender if 80% of our movies are of same genre. Ex: if we have movies related to Comedy(same genre) in our dataset then it will recommend only comedy movies.

Conclusions and Discussion

In this project, a collaborative filtering recommender (CFR) system for recommending movies has been developed and evaluated. The online app was created to demonstrate the User-based Collaborative Filtering approach (as it is found to be best model in comparison with others) for recommendation model.

Let’s discuss the **strengths and weaknesses of the User-based Collaborative Filtering** approach in general.

Strengths: User-based Collaborative Filtering gives recommendations that can be complements to the item the user was interacting with. This might be a stronger recommendation than what a item-based recommender can provide as users might not be looking for direct substitutes to a movie they had just viewed or previously watched.

Weaknesses: User-based Collaborative Filtering is a type of Memory-based Collaborative Filtering that uses all user data in the database to create recommendations. Comparing the pairwise correlation of every user in your dataset is not scalable. If there were millions of users, this computation would be very time consuming. Possible ways to get around this would be to implement some form of dimensionality reduction, such as Principal Component Analysis, or to use a model-based algorithm instead. Also, user-based collaborative filtering relies on past user choices to make future recommendations. The implications of this is that it assumes that a user’s taste and preference remains more or less constant over time, which might not be true and makes it difficult to pre-compute user similarities offline.

Also find the **strength and weakness of Item-based Collaborative Filtering**.

Strengths: Content-based recommender systems don’t require a lot of user data. We just need item data and we are able to start giving recommendations to users. Also, our recommendation engine does not depend on lots of user data, so it is possible to give recommendations to even our first customer as long as we have adequate data to build his user profile.

Weakness: Our item data needs to be well distributed. It won’t be effective to have a content-based recommender if 80% of our movies are of same genre. Also, the recommendations we get will likely be direct substitutes, and not complements, of the item the user interacted with. Complements are more likely discovered through collaborative techniques.

Learning

In the era of competitors like Amazon Prime, Netflix, Hulu, companies want maximum users to stick to their product which will result in higher profits. So, this recommender system plans to provide optimized recommendation of movies to increase viewership on our product. This project plans to give vital information to marketing department so that they can market a movie in order to make profits. Also, the PR team and Design team will use the information generated from this project to place movies efficiently on the website and target specific segment of customers.

In this project, we learn how companies like Amazon, Netflix suggests us products based on our purchases and how they target more customers. For ex: If I watched Avengers endgame and Thor Ragnarok on Netflix, the company will be pretty sure that I will be watching some superhero movie so they will recommend me similar type of movies and make money out of me. Understanding how these companies recommend products based on our liking and history was the biggest take away from this project.

Appendix

Code location: https://github.com/dev5680/IST_707_MovieRec

Execute From Published App: https://dchatter.shinyapps.io/movieRec/

Code attached -

movieRec_descr.Rmd   movieRec.R   helpercode.R   server.R   ui.R