**Movie Recommendation System Initial Data Results**

**Initial Results**

**Project overview:**

This project is to examine Recommender Systems in regards to User Based Collaborative Filters and Item Based Collaborative Filter

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

**Dataset link:**

[MovieLens Latest Datasets](http://files.grouplens.org/datasets/movielens/ml-latest-small.zip)

**Datasets**

The dataset is GroupLens research lab in the University of Minnesota and available in the MovieLens website

To build a recommendar system I used only **movies.csv** and **ratings.csv** data files.

The structure of movies table is given below:

The total number of rows are 9,742 and 3 variables

movieId: number [1:9742] 1 2 3

title: character [1:9742] "Toy Story (1995)" "Jumanji (1995)"

genres: character [1:9742] "Adventure|Animation|Children|Comedy|Fantasy"

The next table is ratings:

The total number of rows are 100,836 and 3 variables

userId: number [1:100836] 1 1 1

movieId: number [1:100836] 1 3 6 47

ratings: number [1:100836] 4 4 4 5

**Data Exploration and pre-processing**

Most popular movie genres:

Chart, histogram

Description automatically generated

Next I explored ratings table to find out best movie in term of ratings using IMDB weight rating function

The function for WR is:

function(R, v, m, C) {

return (v/(v+m))\*R + (m/(v+m))\*C }

Which gives the following movie list with top 5 rating

Shawshank Redemption, The (1994) 5.0

Pulp Fiction (1994) 5.0

Forrest Gump (1994) 5.0

Matrix, The (1999) 5.0

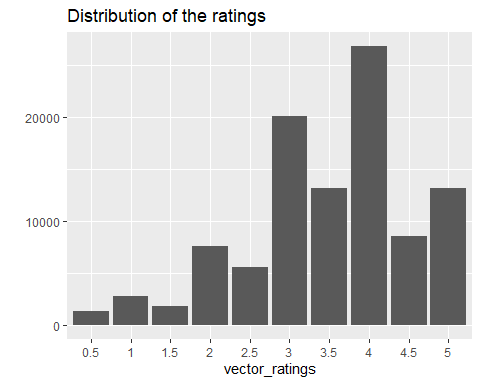
Star Wars: Episode IV - A New Hope (1977) 5.0

To attain the movie features matrix, the pipe-separated genres available in the movies dataset were split. The data.table package has a tstrsplit() function to perform string splits. This is basically movies$genres but each genre is separated into columns.

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

Distribution of the ratings



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

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*.

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

## Exploring Similarity Data

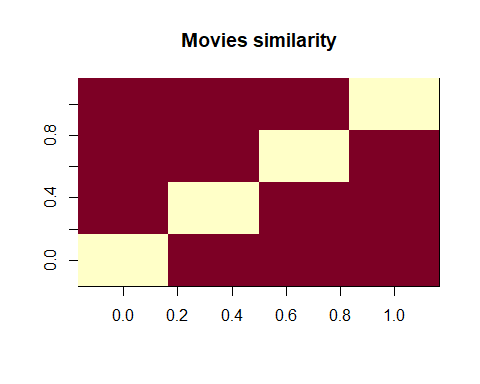
Collaborative filtering algorithms are measuring the similarity between users or between items. For this purpose, recommenderlab contains the similarity function. The supported methods are similarities are cosine, pearson, and jaccard.

Next, to investigate the first four users are with each other by having a similarity matrix

## 

each row and each column corresponds to a user, and each cell corresponds to the similarity between two users.

Using the same approach, We investigate between the first four movies.



**Data Preparation**

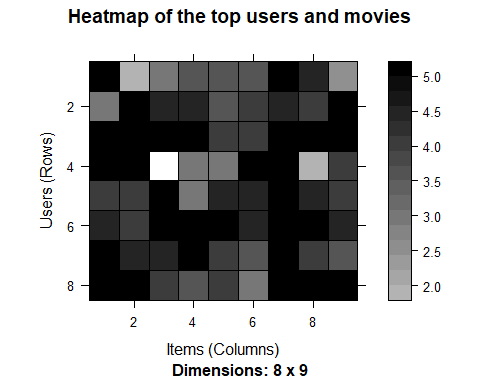
The data preparation process consists of the following steps:

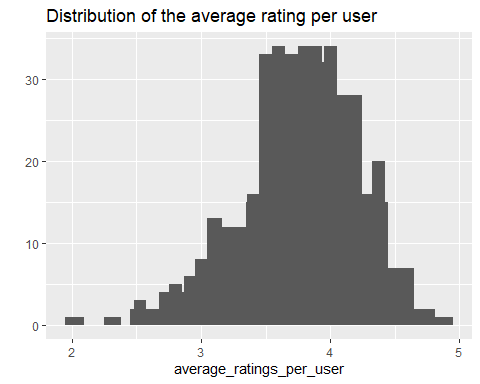
1. Select the relevant data.
2. Normalize

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:

378 x 436 rating matrix of class ‘realRatingMatrix’ with 36214 ratings.

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





Normalize the data using z-score.

# the normalized matrix for the top movies

## 

## UBCF Model

recom\_result

## [,1]   
## [1,] "Star Maps (1997)"   
## [2,] "Dances with Wolves (1990)"   
## [3,] "Ulee's Gold (1997)"   
## [4,] "Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)"  
## [5,] "Mary Poppins (1964)"

eval\_results

## TP FP FN TN precision recall  
## 1 0.03225806 0.9032258 15.32258 416.7419 0.03448276 0.002426564  
## 3 0.08064516 2.7258065 15.27419 414.9194 0.02873563 0.011855444  
## 5 0.12903226 4.5483871 15.22581 413.0968 0.02758621 0.015026384  
## 10 0.35483871 9.0000000 15.00000 408.6452 0.03793103 0.024880765  
## 15 0.51612903 13.5161290 14.83871 404.1290 0.03678161 0.048902617  
## 20 0.67741935 18.0322581 14.67742 399.6129 0.03620690 0.056011288  
## TPR FPR  
## 1 0.002426564 0.002165417  
## 3 0.011855444 0.006534609  
## 5 0.015026384 0.010899784  
## 10 0.024880765 0.021537738  
## 15 0.048902617 0.032351336  
## 20 0.056011288 0.043158998

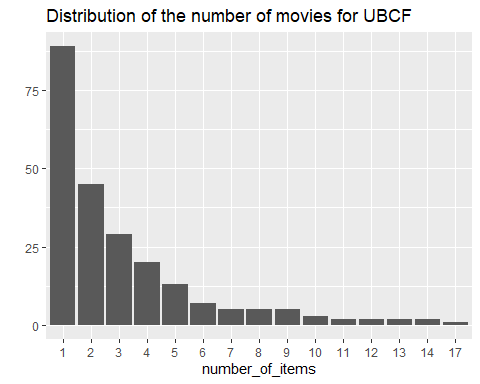
**Applying the recommender model on the test set**

## Recommendations as 'topNList' with n = 10 for 72 users.

## Explore results

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

## [,1] [,2] [,3] [,4]  
## [1,] 253 49272 3489 317  
## [2,] 594 60069 49272 110  
## [3,] 596 1234 3897 527  
## [4,] 2080 1148 4034 1387  
## [5,] 1968 1304 5669 1407  
## [6,] 509 6711 541 1193  
## [7,] 4034 6863 253 1240  
## [8,] 2395 788 3911 1374



UBCF distribution has a longer tail meaning that there are some movies that are recommended much more often than the others.

Let’s take a look at the top titles:

## Movie title No of items  
## 223 Clerks (1994) 17  
## 673 Space Jam (1996) 14  
## 1387 Jaws (1975) 14  
## 1234 Sting, The (1973) 13

## ITEM-based Collaborative Filtering Model

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:

1. Extract the user rating of each purchase associated with this item.
2. Extract the similarity of the item with each purchase associated with this item.
3. Multiply each weight with the related similarity.
4. Sum everything up.

Then, the algorithm identifies the top 10 recommendations:

## Recommendations as 'topNList' with n = 10 for 61 users.

Results of the recommendations for the first user:

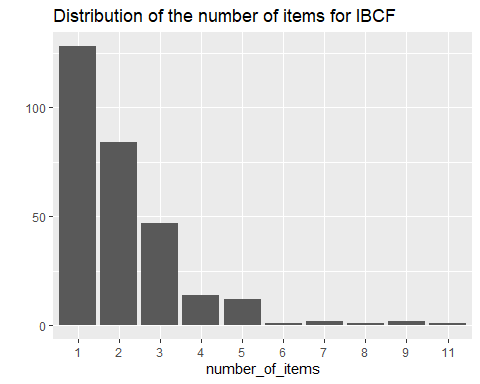
## [1] "There's Something About Mary (1998)"  
## [2] "Blazing Saddles (1974)"   
## [3] "Spider-Man (2002)"   
## [4] "Bourne Identity, The (2002)"   
## [5] "Lady and the Tramp (1955)"   
## [6] "Groundhog Day (1993)"   
## [7] "Blood Diamond (2006)"   
## [8] "Natural Born Killers (1994)"   
## [9] "Quiz Show (1994)"   
## [10] "Batman (1989)"

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,] 1923 364 555 56367  
## [2,] 3671 1220 8644 1917  
## [3,] 5349 1917 908 46578  
## [4,] 5418 1923 35836 3033  
## [5,] 2080 2683 6863 68358  
## [6,] 1265 4027 41566 5299  
## [7,] 49530 4979 2268 2329  
## [8,] 288 6942 6870 6016  
## [9,] 300 595 33493 353  
## [10,] 592 1199 48385 3671

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  
## 7 Sabrina (1995) 11  
## 3 Grumpier Old Men (1995) 9  
## 6 Heat (1995) 9  
## 21 Get Shorty (1995) 8

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

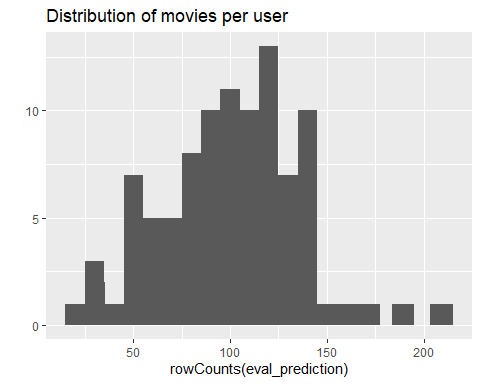
The k-fold cross-validation approach is the most accurate one, although it’s computationally heavier.

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.

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

## Evavluating the ratings

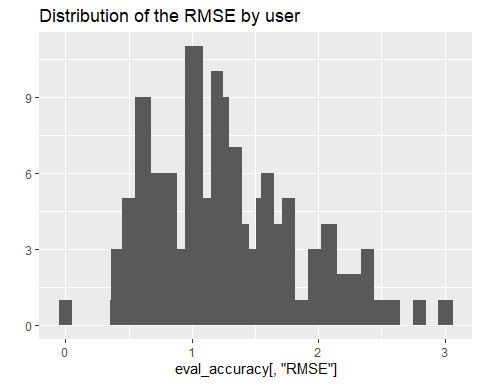
First, I re-define the evaluation sets, build IBCF model and create a matrix with predicted ratings.



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,] 1.8540496 3.4375000 1.3750000  
## [2,] 1.0992422 1.2083333 0.9166667  
## [3,] 0.5983598 0.3580344 0.4035566  
## [4,] 0.4630599 0.2144245 0.3256274  
## [5,] 2.0722253 4.2941176 1.8235294  
## [6,] 2.8480012 8.1111111 2.6666667



performance index for the whole model, I specify *byUser* as FALSE and compute the average indices:

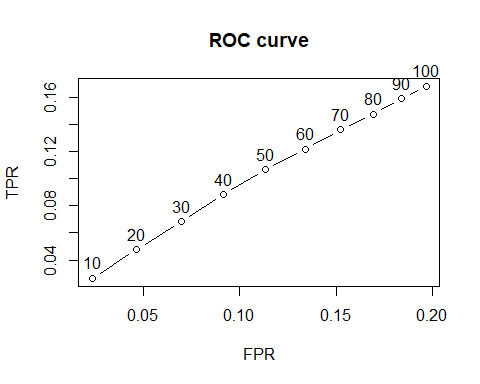
## RMSE MSE MAE   
## 1.3471892 1.8149189 0.9867995

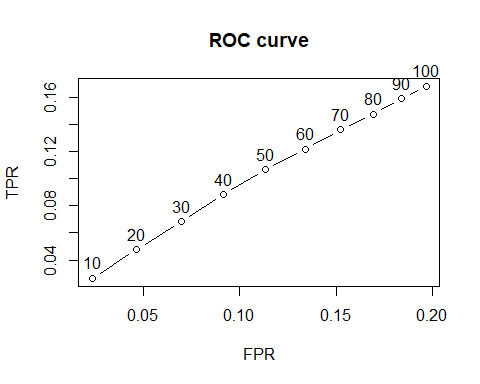
## Evaluating the recommendations

Another way to measure accuracies is by comparing the recommendations with the purchases that have a positive rating. The 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:

## TP FP FN TN  
## 10 7.604167 32.08333 317.3542 1366.958  
## 20 14.552083 64.62500 310.4062 1334.417  
## 30 21.114583 96.82292 303.8438 1302.219  
## 40 27.739583 127.85417 297.2188 1271.188  
## 50 33.895833 158.23958 291.0625 1240.802  
## 60 39.291667 187.14583 285.6667 1211.896

ROC and the precision/recall curves:





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.

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 (IBCF) can provide as users might not be looking for direct substitutes to a movie they had just viewed or previously watched. This is a initial conclusion from data