Data 612 Final Project

What2Watch - Recommendation System for Movies

Joseph Simone

May 19, 2020

Contents

Overview

For this project, the goal was to develop and deploy a Collaborative Filtering Recommender System (CFR) for Movie Recommendations.

A Collaborative Filtering approach consists of only the User's Preferences, therefore, does not factors in the values or features of the particular variable being recommended.

In addition, the Movie Lens Dataset was used to gerenate values for this Recommendation System.

After the Trained CFR Model was successfully implemented, this system was deployed in a Shiny R Application.

The Data

The data for this project is the MovieLens Dataset, which can be found here here.

Containing 105339 ratings and 6138 tag applications, across 10329 movies, rated by 668 users.

The zipfile downloaded from the above link contained four files: *links.csv*, *movies.csv*, *ratings.csv* and *tags.csv*.

This system implements the use of the files *movies.csv* and *ratings.csv*.

Decscription of *movies* file:

Summary of of *ratings* file:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   3.500   3.517   4.000   5.000
```

In for this to function properly, both the `usersId` and `movieId` will have to be changed from their data types of integers to factors.

In addition, the genres of the movies will ne to be reformatted.

Data Processing

First, the movie's genres will have to be converted into a one-hot encoding format.

This will serve as the backbone of how users will be able to search for the movies they have watched within specific genres in the long format.

Extract a list of genres

Matrix of Movies and their Genres

The creation of a `search matrix` will act as a database of a movie by their genre(s).

From here we can see the data begin to grow in size and sparcity.

Now each movie will correspond to one or more genres.

realRatingMatrix Creation

This project utilizes the use of the `recommenderlab` R package.

In order to build a recommendation engine within `recommenderlab`, the conversion of the newly created `ratings matrix` into a Sparse Matrix known as a `realRatingMatrix`.

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

Recommendation Models

The *recommenderlab* package contains preconstructed models for the use of recommendation algorithms:

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

```
## $ALS_realRatingMatrix
## [1] "Recommender for explicit ratings based on latent factors, calculated by alternating least squar
##
## $ALS_implicit_realRatingMatrix
## [1] "Recommender for implicit data based on latent factors, calculated by alternating least squares a
##
## $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/
##
## $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
##
## $UBCF_realRatingMatrix
## [1] "Recommender based on user-based collaborative filtering."
```

This project will utilize both the IBCF and UBCF Models for comparison and performance.

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

```
##
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE

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

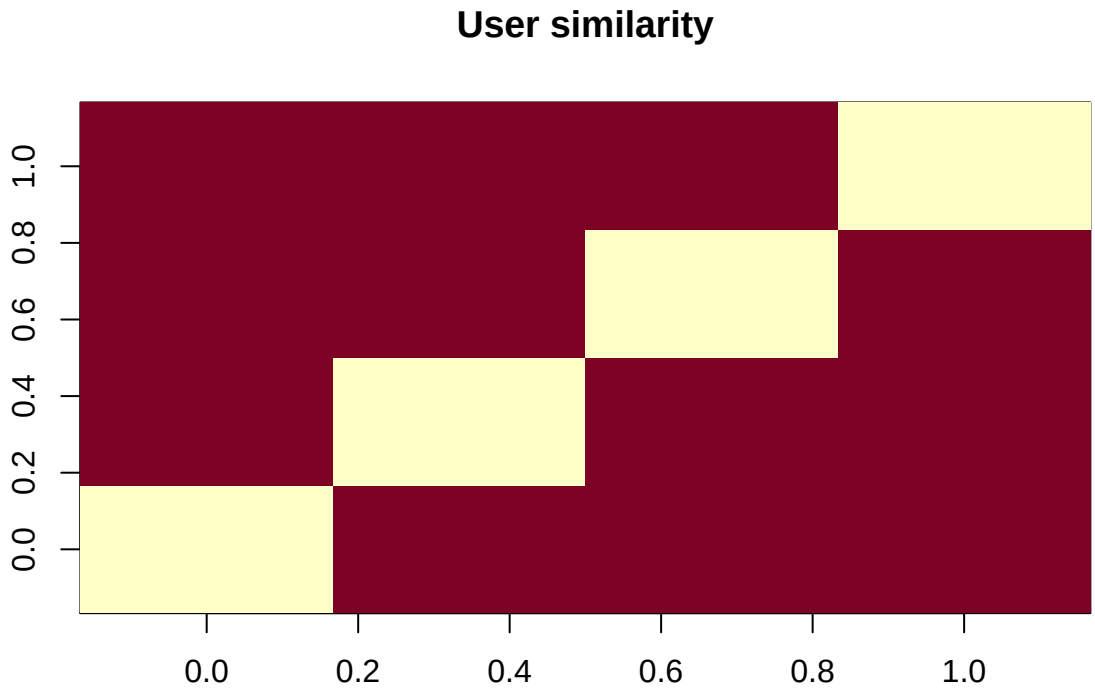
Collaborative Filtering is based on the measuring between the similarity of users or between items.

Within *recommenderlab*, the supported methods to computate similarities are *cosine, pearson*, and *jaccard*.

Exploration of Similarity Data

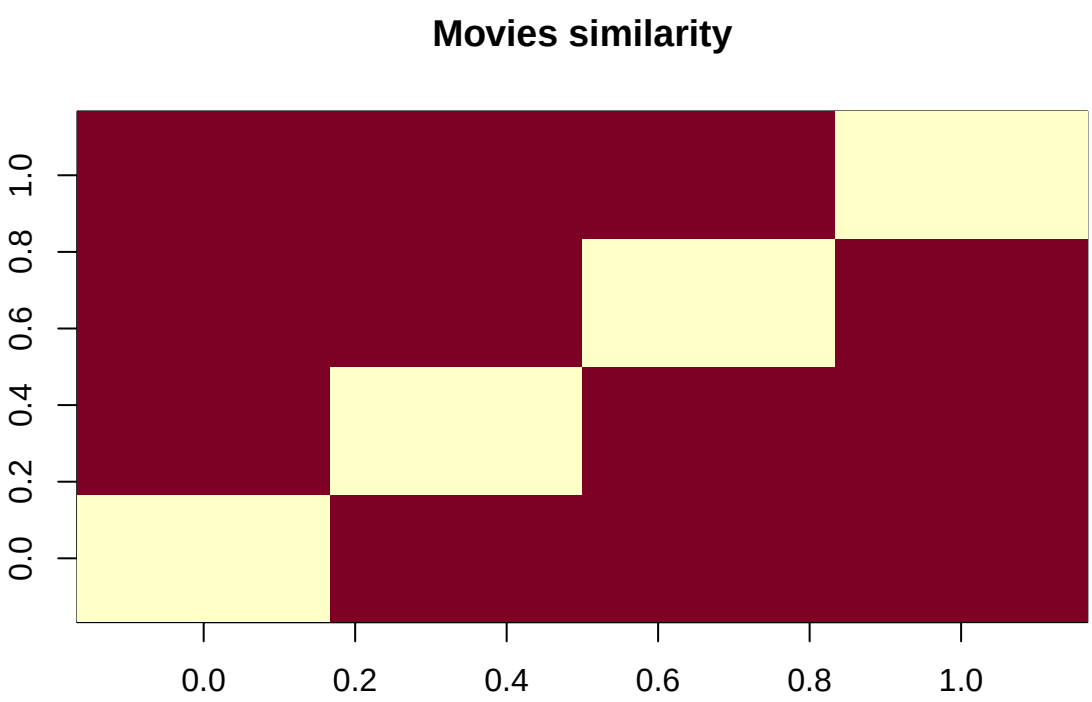
Next, determing how similar the first four users are with each other.

```
##           1         2         3         4
## 1 0.0000000 0.9760860 0.9641723 0.9914398
## 2 0.9760860 0.0000000 0.9925732 0.9374253
## 3 0.9641723 0.9925732 0.0000000 0.9888968
## 4 0.9914398 0.9374253 0.9888968 0.0000000
```



Using the same approach, for the first four movies.

```
##           1         2         3         4
## 1 0.0000000 0.9669732 0.9559341 0.9101276
## 2 0.9669732 0.0000000 0.9658757 0.9412416
## 3 0.9559341 0.9658757 0.0000000 0.9864877
## 4 0.9101276 0.9412416 0.9864877 0.0000000
```



Data Exploration Continued

Now, exploring the second data file's values of `ratings`.

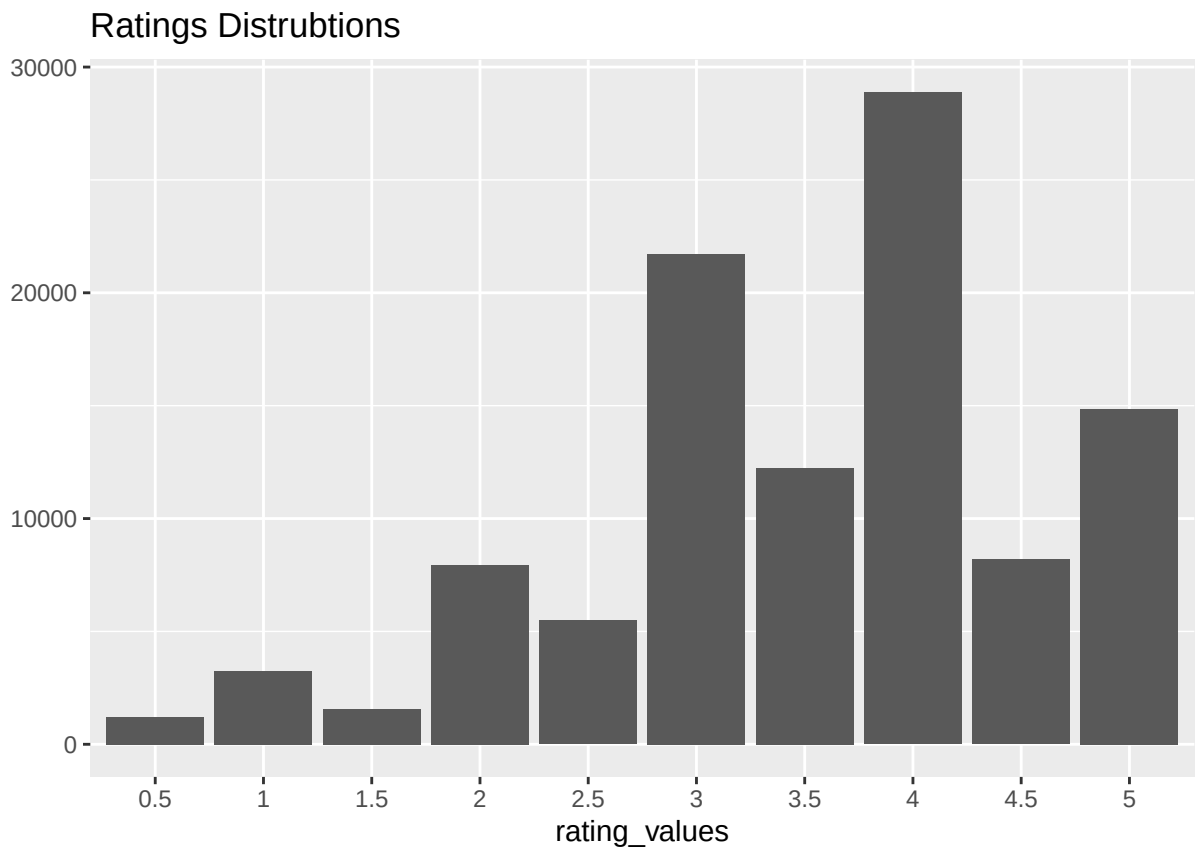
```
##  [1] 0.0 5.0 4.0 3.0 4.5 1.5 2.0 3.5 1.0 2.5 0.5
```

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

There are 11 unique `rating` values.

Distribution of Ratings

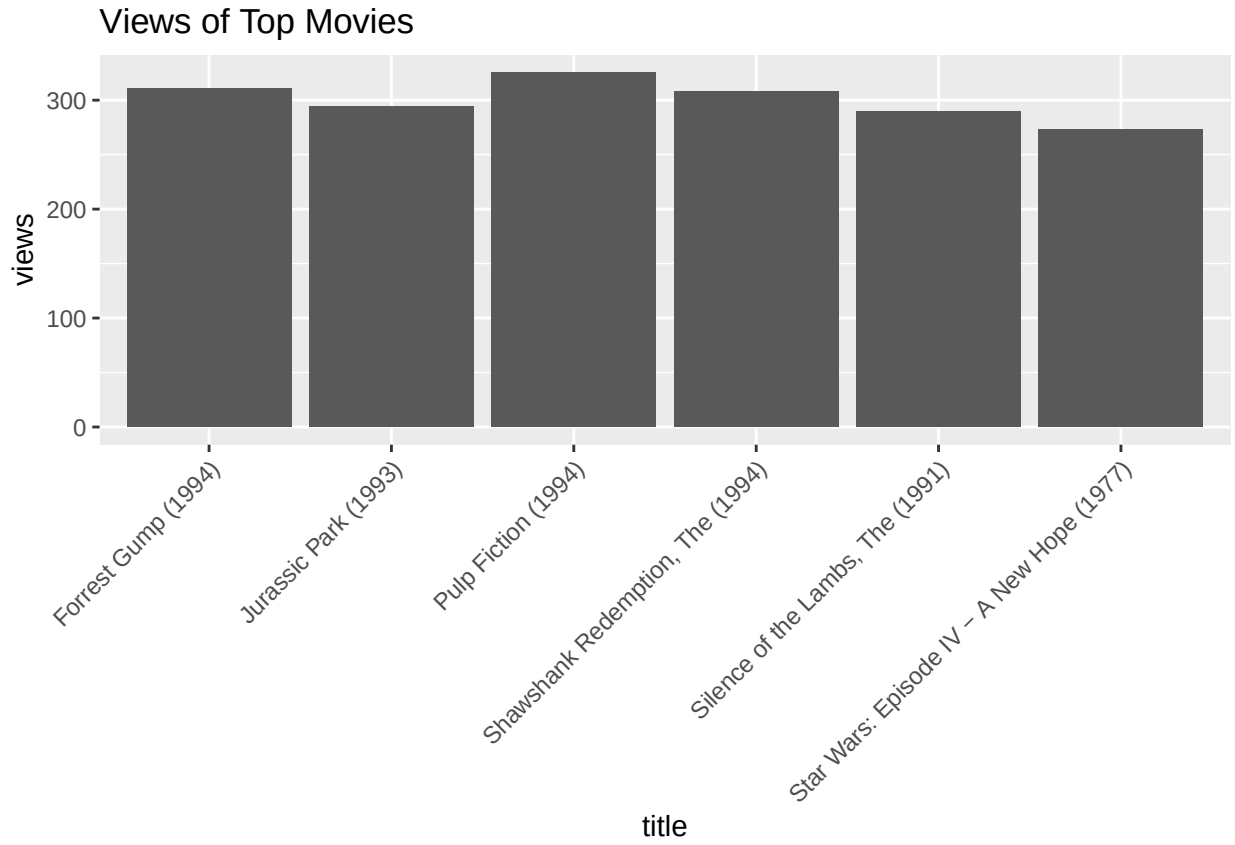
Arating equal to 0 represents a missing value, therefore, them from the dataset before visualizing the results.



The most common rating is 4.

The majority of movies are rated with a score of 3 or higher.

Number of Views ~ Top Movies

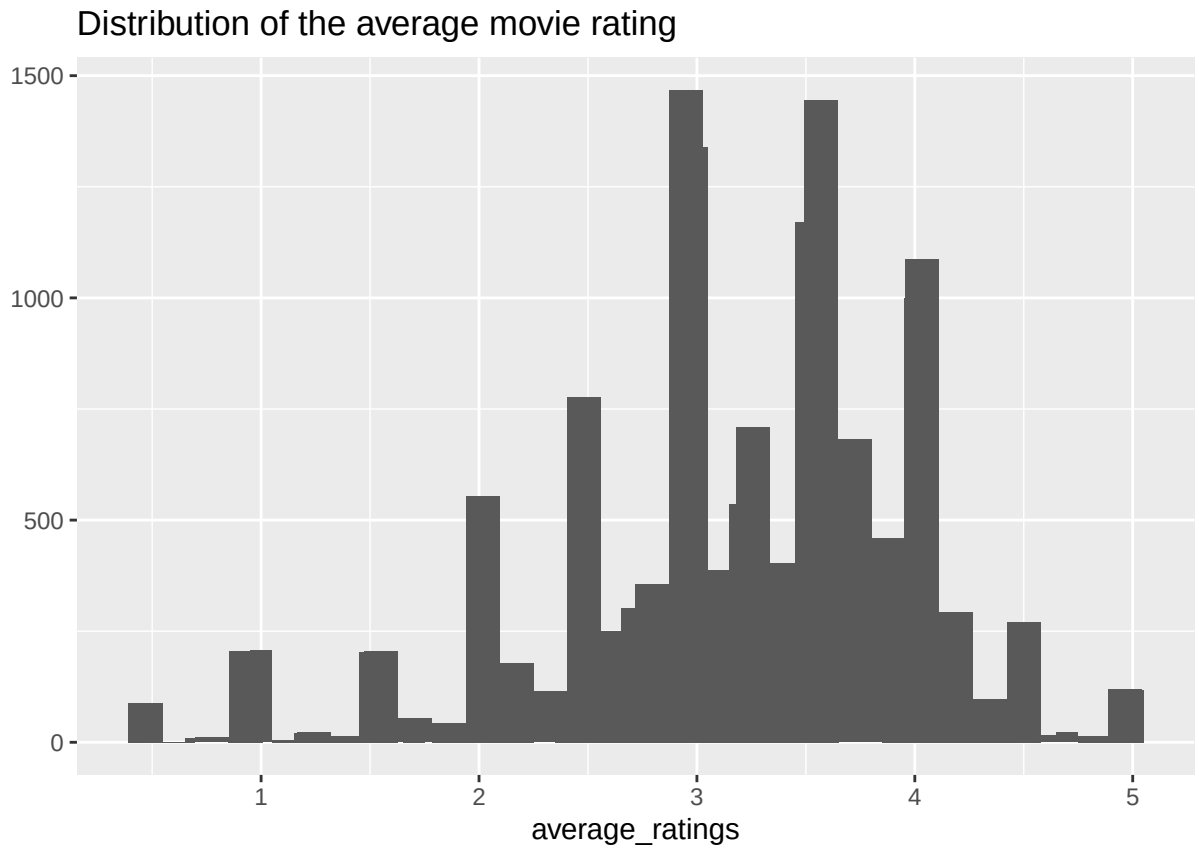


"Pulp Fiction (1994)" is the most watched Movie, with **"Forrest Gump (1994)"** being the second.

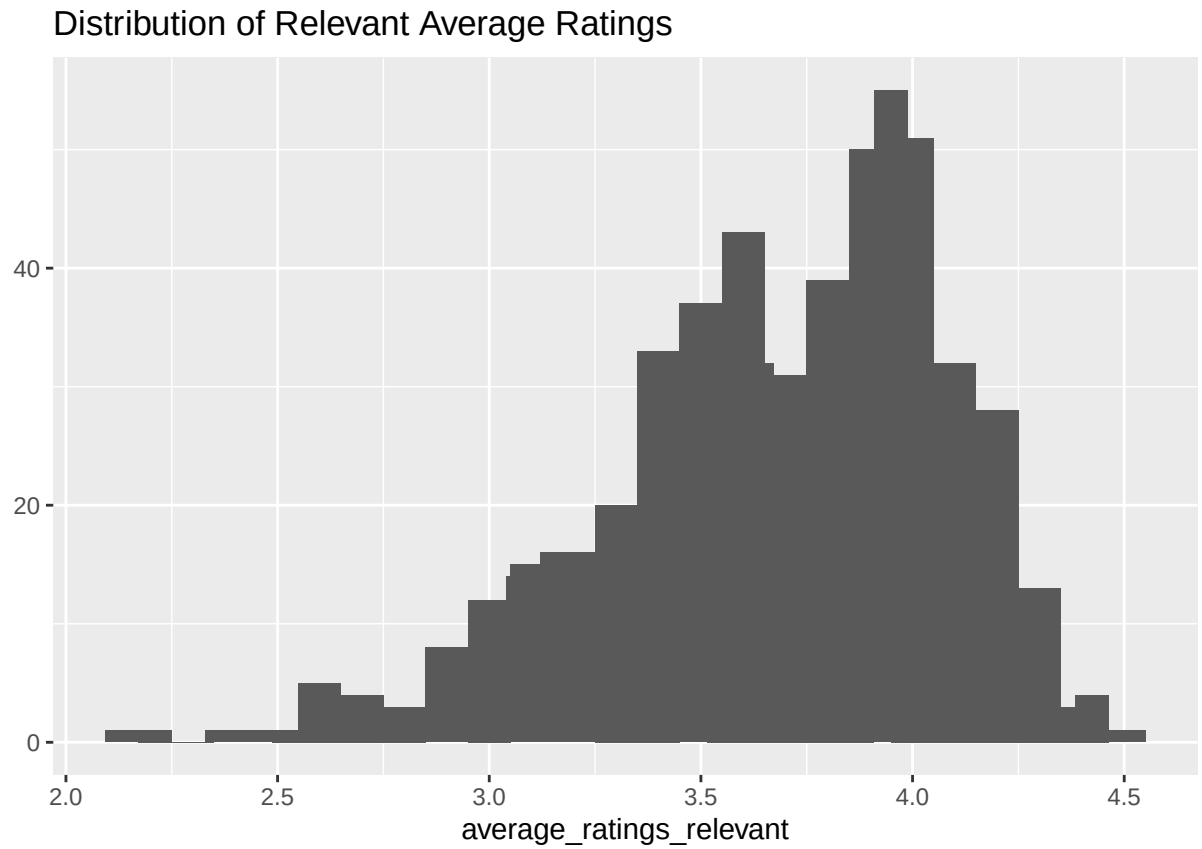
Distribution of the Average Ratings

To find the Top-Rated Movies, the average rating for each was calculated.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The first graph represents the distribution of the average movie rating. The highest value is ~ 3, with a few movies whose rating is either 1 or 5.

This is most likely due to the fact that these movies received a rating from only a few users, thererfore we should exclude these ratings.

The movies where number of views is below the defined threshold of 50 we removed. This creates a more narrow subset of the most relevant movies.

The second graph represents the distribution of the relevant average ratings. The rankings are between 2.16 and 4.45. The highest value changes, and now it is ~ 4.

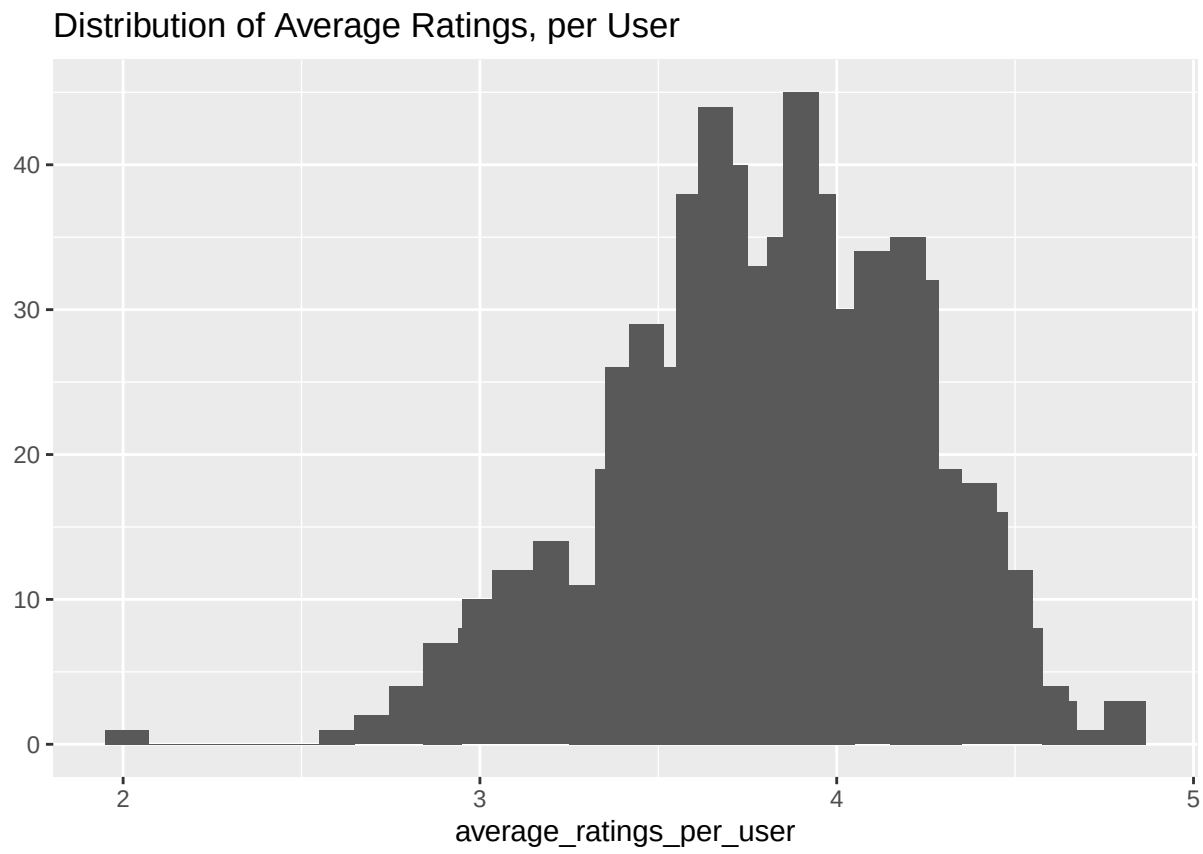
Data Preparation

Part 1 - Revevant Data In order to select relevant data, defining the minimum number of users per rated movie and the minimum views per movie as 50

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

The previous rating-matrix had 668 users and 10325 movies, now the newly create most relevant rating-matrix contains 420 users and 447 movies.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

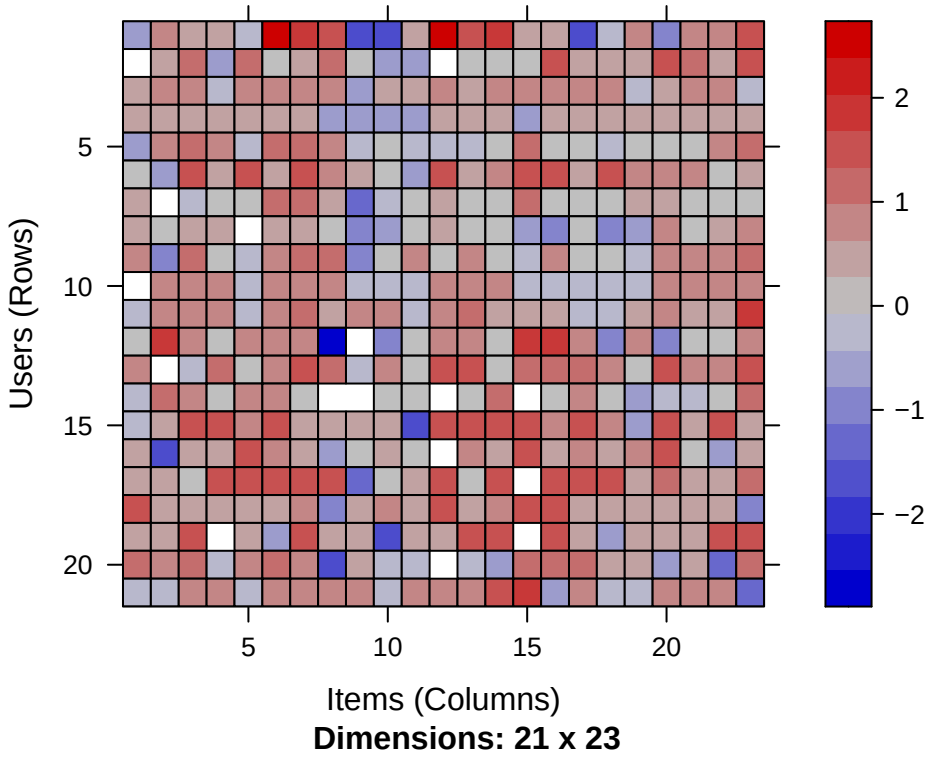


Part 2 - Normalization When dealing with a user pool who rates at a high or low ratings can result in bias.

In an effort to circumvent this problem, the normalization of the data was need

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

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



There are still some lines that seem to be more blue or more red.

This is due to the above chart is visualizing only the top movies.

However, the average rating is still 0 for each user.

Part 3 - Convert Data to Binary In order for the recommendation models to work well with the data we already having, conversion to binary data, will be useful. This is done by defining a matrixta encompassing 0's and 1's.

The 0's will be either treated as missing values or as bad ratings.

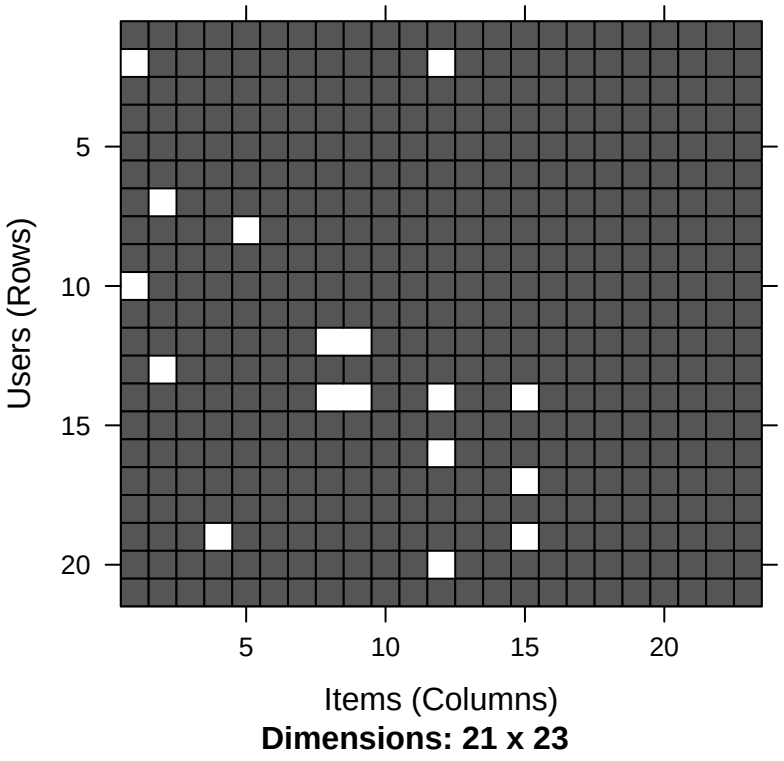
Set-up

- 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.

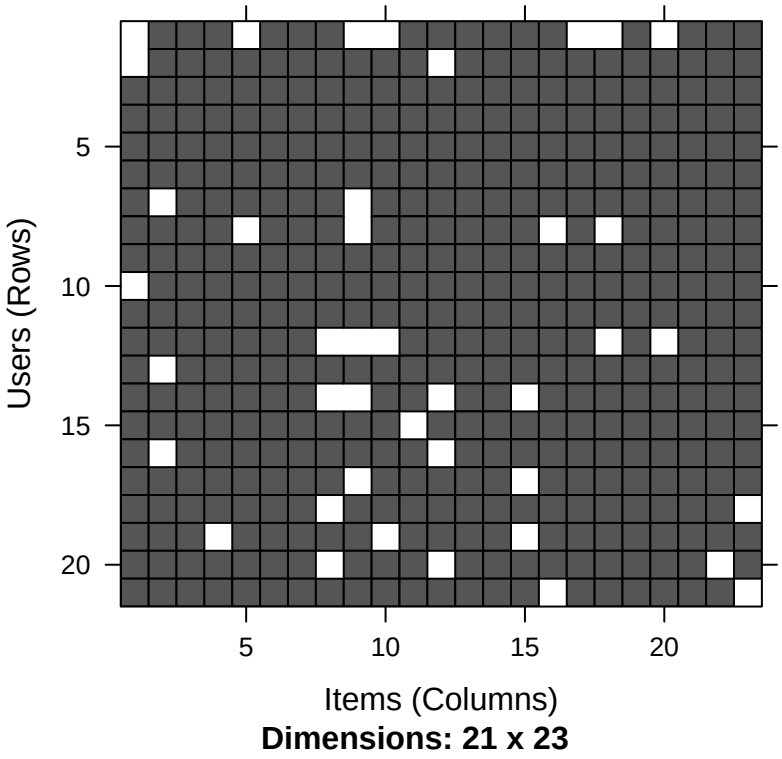
Depending on the context, one option might be more suited to the needs of the models, depending on the context.

Next, defining of two matrices following the two different "set-up" options which visualize 5% portion of each of newly created binary matrices.

Option 1: Define a matrix equal to 1, if the movie has been watched



Option 2: Define a matrix equal to 1 if the cell has a rating above the threshold



In the second heatmap, there are more white cells which means that there are more movies with no or bad ratings than movies that were not viewed.

ITEM-BASED Collaborative Filtering Model

For this type of model, we will first need to create a `rating-matrix`, which rows corresponds to users and columns corresponds to items.

This approach is based on:

1. For each two items, measure 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 ratings or reviews

Train & Test Sets

Built the model using 80 of the total dataset as a `training set`, and 20 as a `test set`.

Build 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 funtion, which is *Cosine* by default, may also be *pearson*. I create the model 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
```

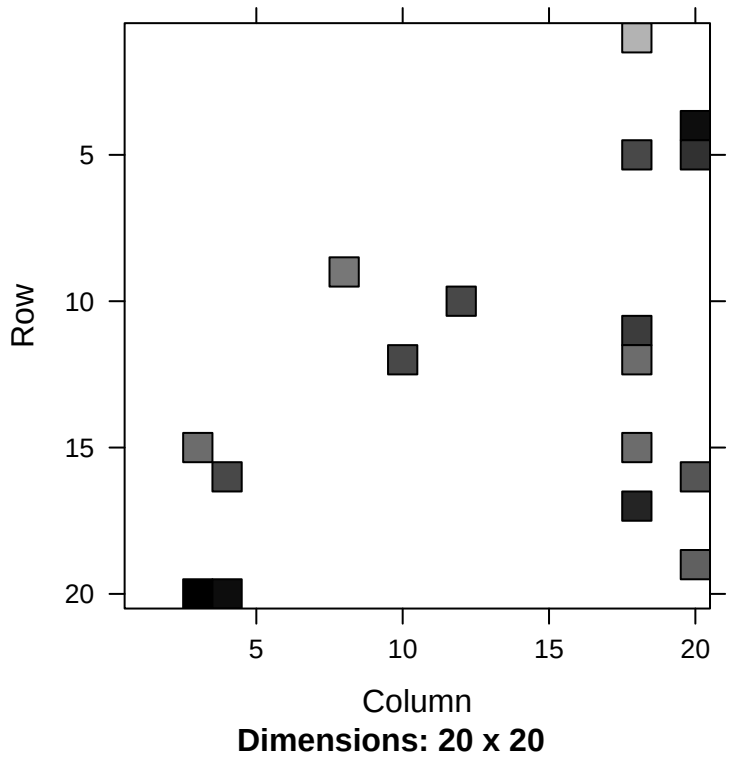
```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 340 users.
```

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

Exploring Model

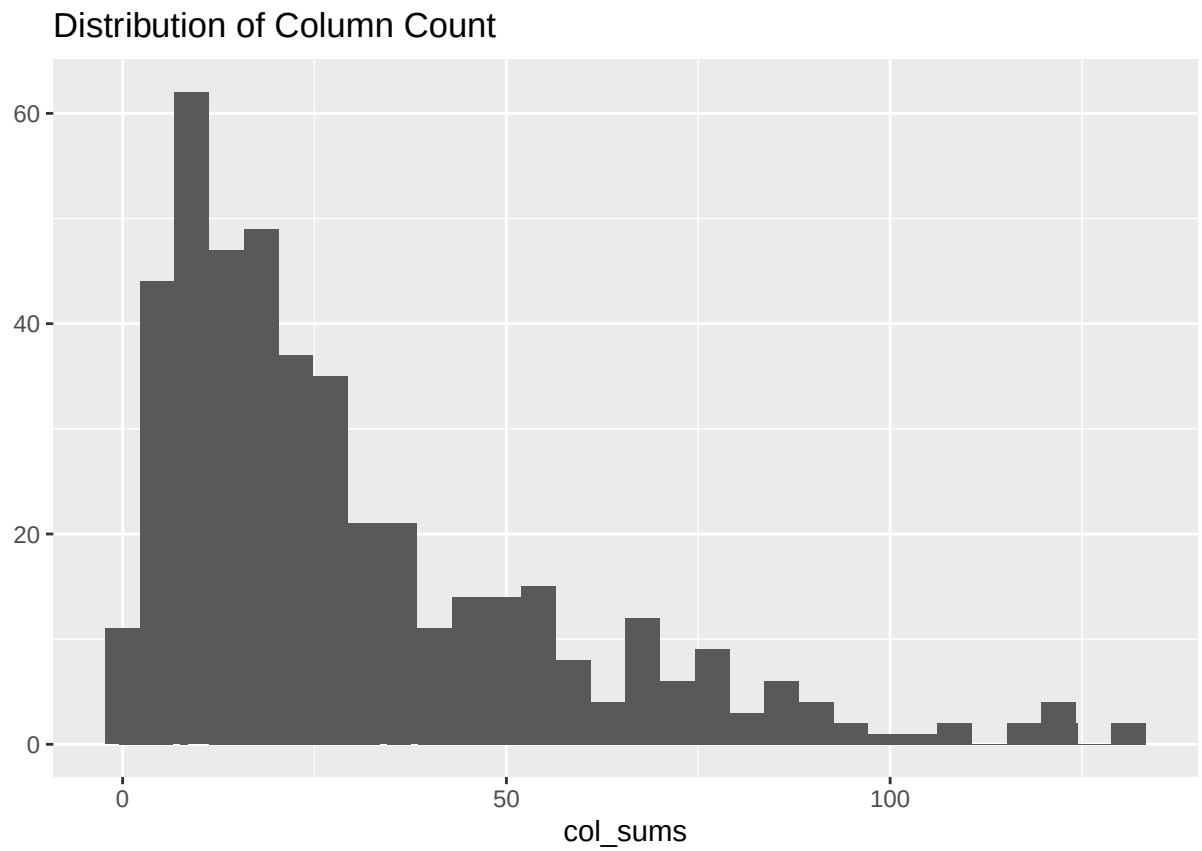
```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

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




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

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



This newly created `dgCMatrix` similarity matrix has dimensions are 447 x 447, which is equal to the number of items.

The Heatmap shows 20 first items show that many values are equal to 0.

This is due to each row contains only $k = 30$ elements that are greater than 0.

The number of non-null elements for each column depends on the amount the corresponding movie was included in the top k of another movie.

Therfore, this matrix is not simmetric, which is also the same in our model.

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

IBCF Recommendation System Implementation

```
## Recommendations as 'topNList' with n = 10 for 80 users.
```

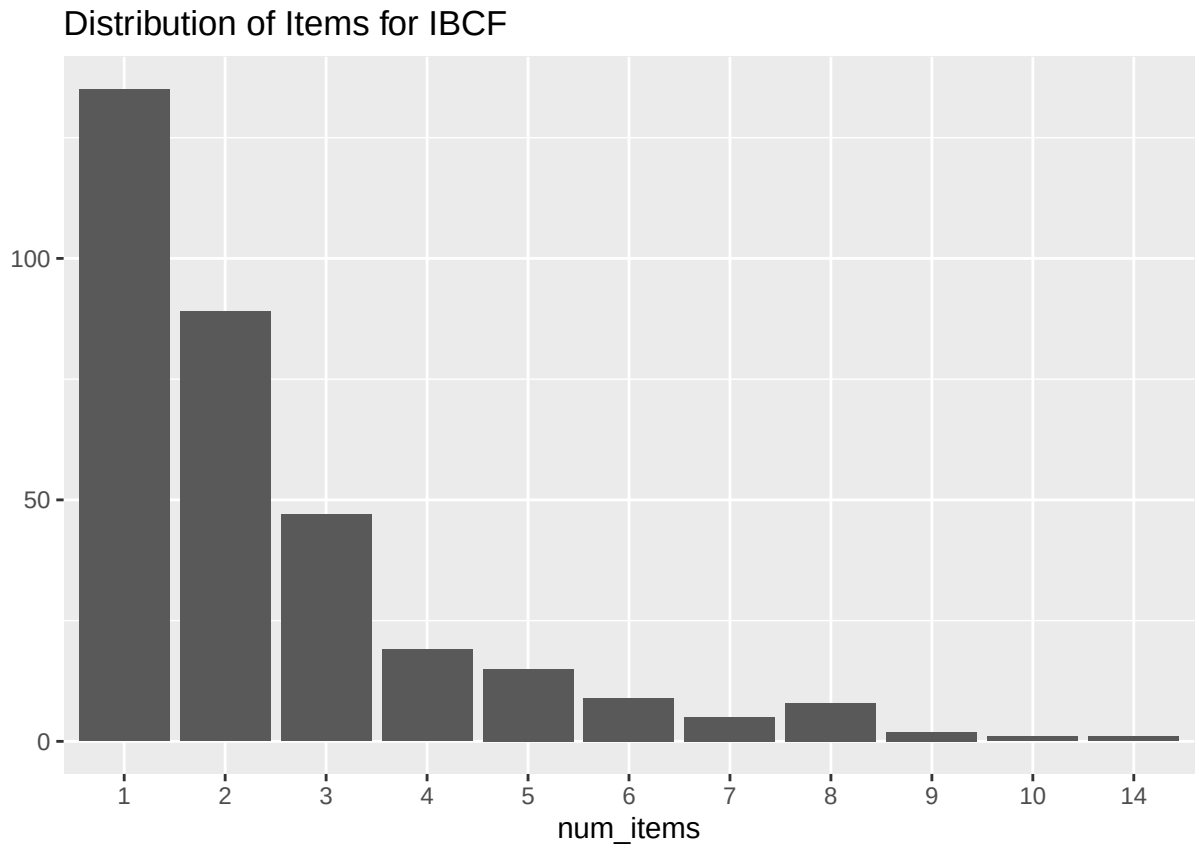
Let's explore the results of the recommendations for the First User

```
##  [1] "Casper (1995)"                      "First Knight (1995)"
##  [3] "Johnny Mnemonic (1995)"             "Judge Dredd (1995)"
##  [5] "Stargate (1994)"                    "Santa Clause, The (1994)"
##  [7] "What's Eating Gilbert Grape (1993)" "Executive Decision (1996)"
##  [9] "Dragonheart (1996)"                 "Fantasia (1940)"
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]  158   50   62  553
##  [2,]  168  110  661 2321
##  [3,]  172  151  785  141
##  [4,]  173  173 1282  435
##  [5,]  316  231 1376  485
##  [6,]  317  236 1673 2291
##  [7,]  337  266 1682 7143
##  [8,]  494  296 1199  104
##  [9,]  653  356 1307  920
## [10,] 1282  410 1339  236
```

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:



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.

Build Model

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. I build a recommender model 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 340 users.
```

```
## 340 x 447 rating matrix of class 'realRatingMatrix' with 31476 ratings.
## Normalized using center on rows.
```

UBCF Recommendation System Implementation

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

```
## Recommendations as 'topNList' with n = 10 for 80 users.
```

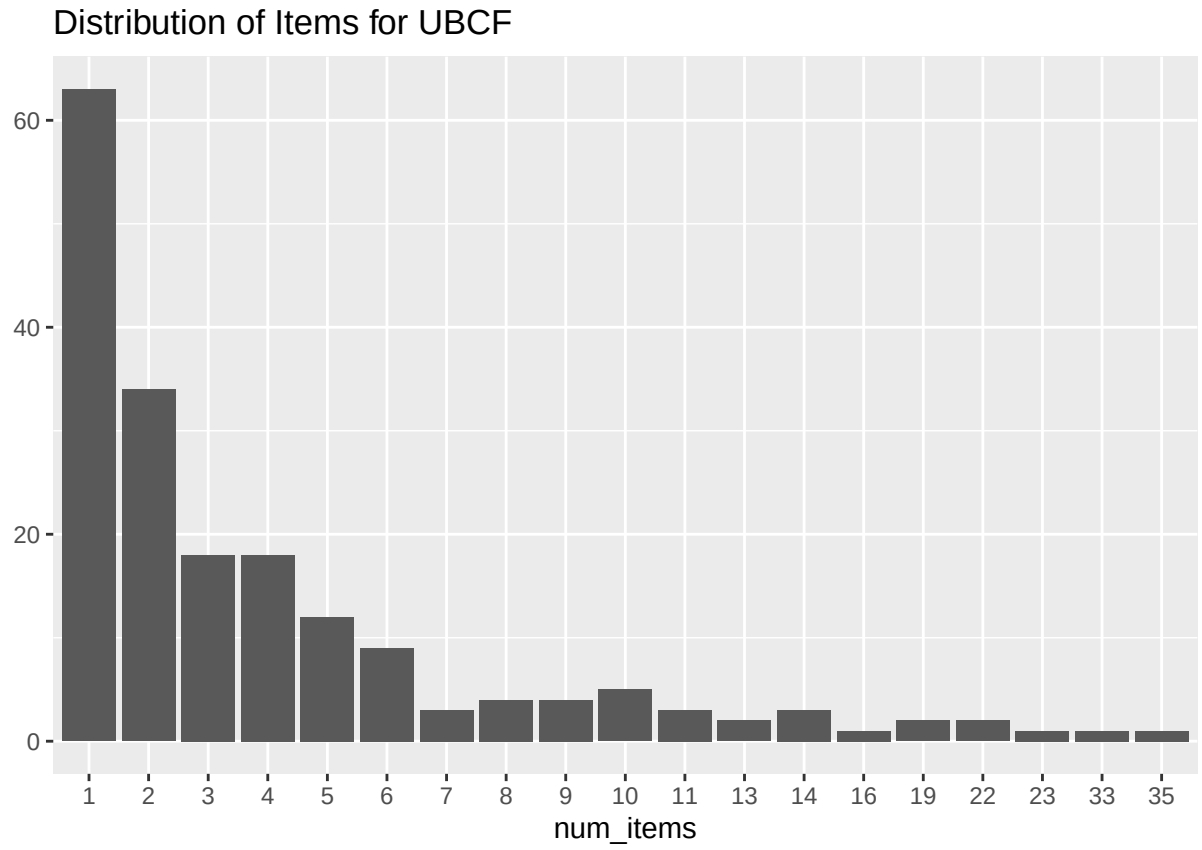
Explore results

First Four Users

```
##        [,1] [,2] [,3] [,4]
##  [1,]   318  356 2571  318
##  [2,]  4226  318  541 2959
##  [3,]  4973  296 5995 2571
##  [4,]  2916  110 7153   50
##  [5,]   908  527  858 6016
##  [6,]  2571  260  924  527
##  [7,]  1200  377 3578 2329
##  [8,]  5418   50 4993 4995
##  [9,] 79132   34 2716 7153
## [10,]  7153  457 1225 4886
```

The matrix above contains contain the `movieId` of each recommended movie, rows, for the first four users, cloumns, in the test dataset.

Frequency Histogram



Compared with the `IBCF`, the distribution of some movies that are recommended much more often than the others.

The maximum is more than 30, compared to 10 for `IBCF`.

Top Movie Titles

Comparison of the outcomes of both the `UBCF` with the `IBCF`, aids in finding useful insight on different methods.

The 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.

In addition, 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.

Evaluatiion of the Recommender Systems

In order to compare the models' performances and choose the best suited model:

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

Data Preparation for the data to Evaluate Models

- Splitting the data into `Training` and `Test` Sets
- `Bootstrapping` Data
- `K-Fold` Approach

Splitting Data

Splitting the data into Training and Test Sets at a 80/20 proportion.

First, for each user in the test set, we need to define how the number of moviess to use, in order to generate recommendations.

To achieve this we need to check the minimum number of movies rated by users to be sure there will be no users with no movie to test.

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

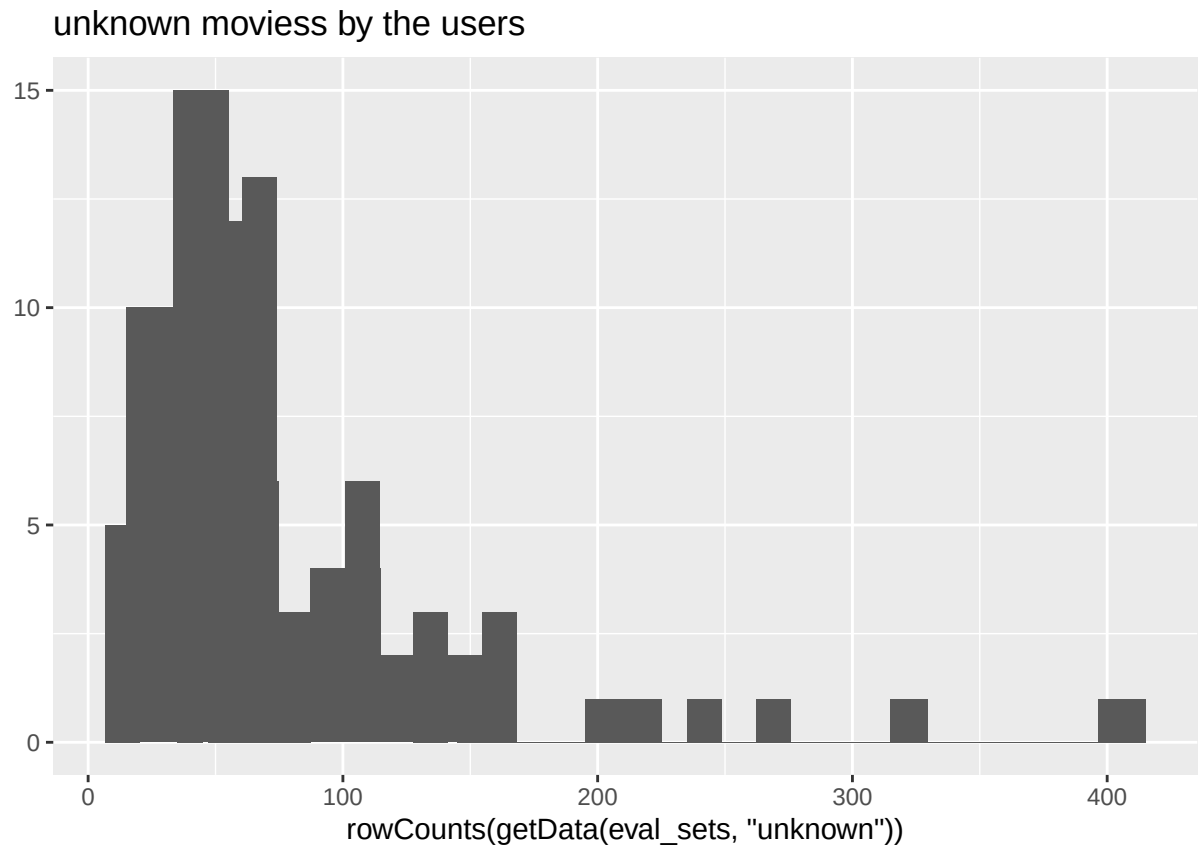
```
## Evaluation scheme with 5 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.800
## Good ratings: >=3.000000
## Data set: 420 x 447 rating matrix of class 'realRatingMatrix' with 38341 ratings.
```

```
## 336 x 447 rating matrix of class 'realRatingMatrix' with 31111 ratings.
```

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

```
## 84 x 447 rating matrix of class 'realRatingMatrix' with 6810 ratings.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

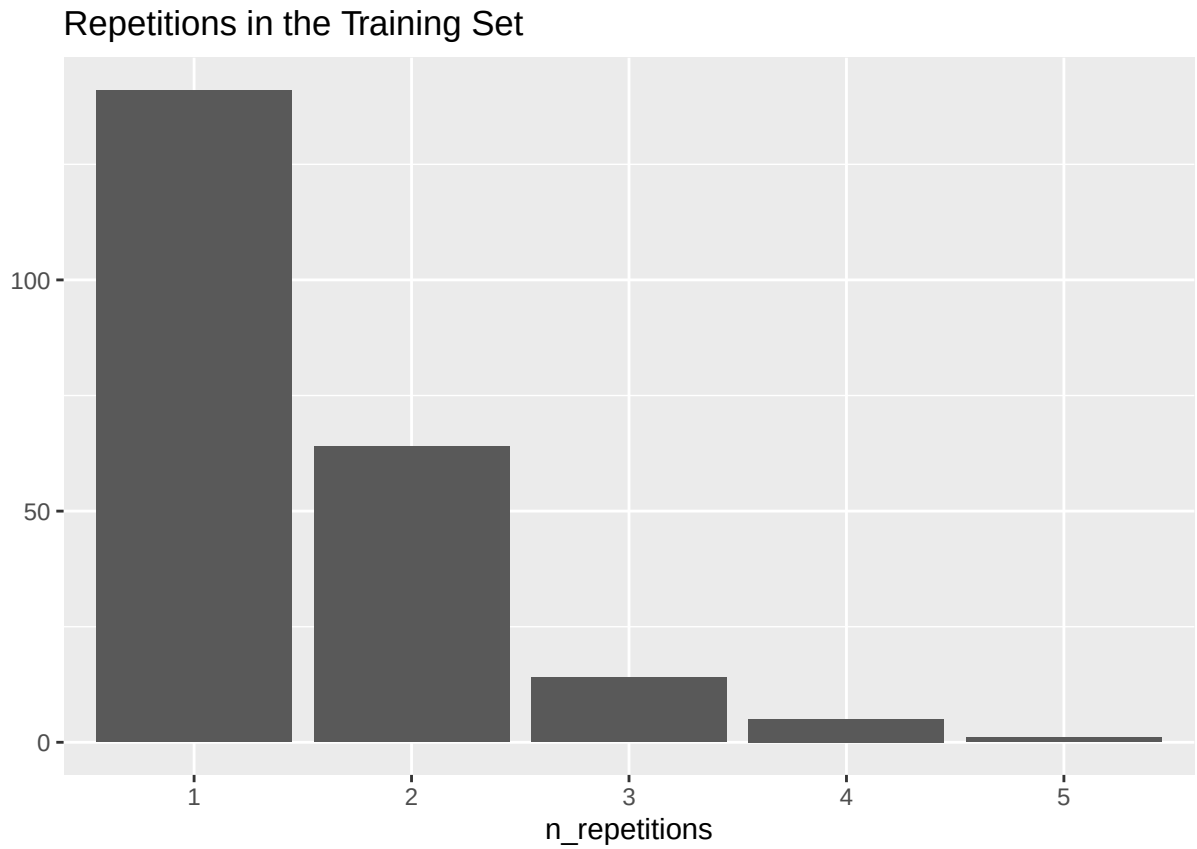


This represents the unknown moviess by the users, varying quite a lot.

Bootstrapping

The same user can be sampled more than once during `bootstraping`.

When the training set has the same size previously, this will be more users in the test set.



Respresents that most of the users that have sampled lower than four times.

K-Fold Cross-Validation Approach

K-Fold Cross-Validation yields is the most accurate, however, is the most resource intensive.

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

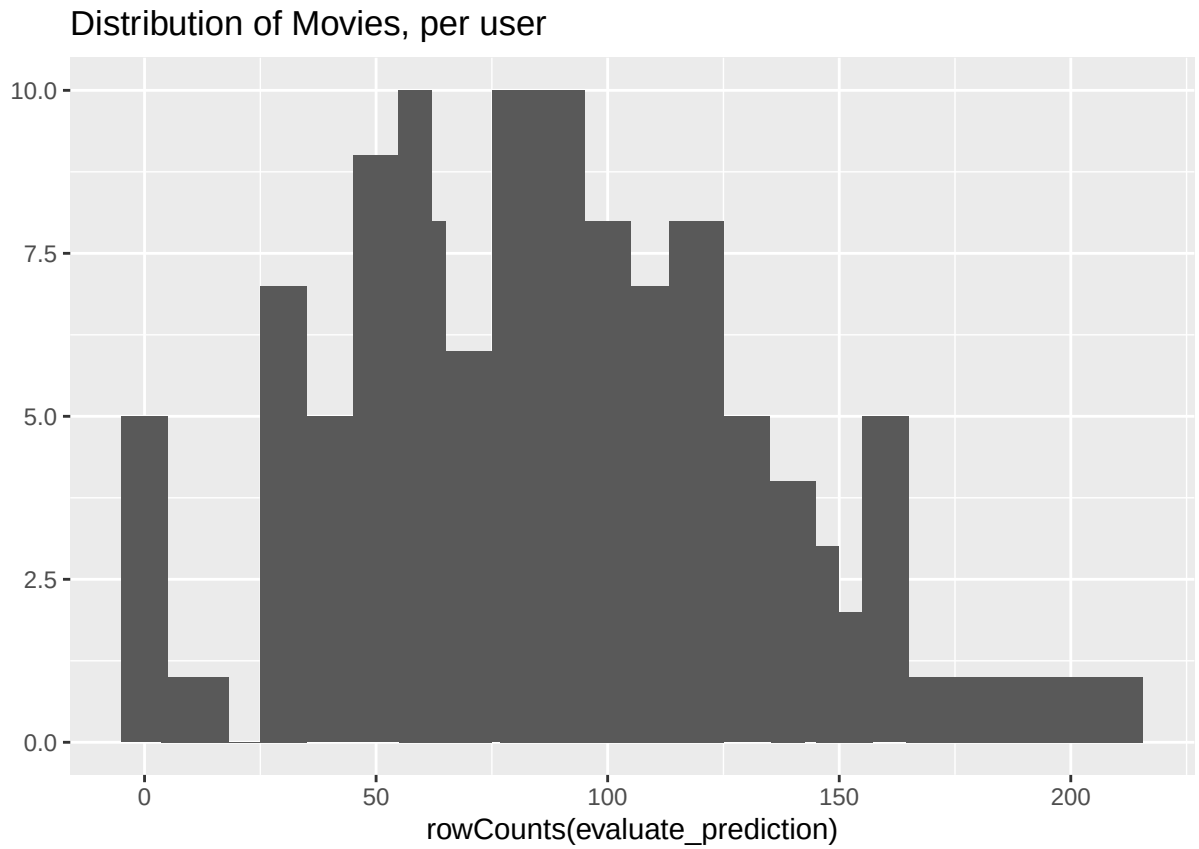
When using the K-Fold approach, results in four sets of the same size 315.

Evaluation of Ratings

K-Fold Approach is used for evaluation of the results:

- First, re-defining the evaluation sets.
- Build IBCF model
- Creating a matrix with predicted ratings.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Represents the distribution of movies per user in the matrix of predicted ratings.

Compute Accuracy

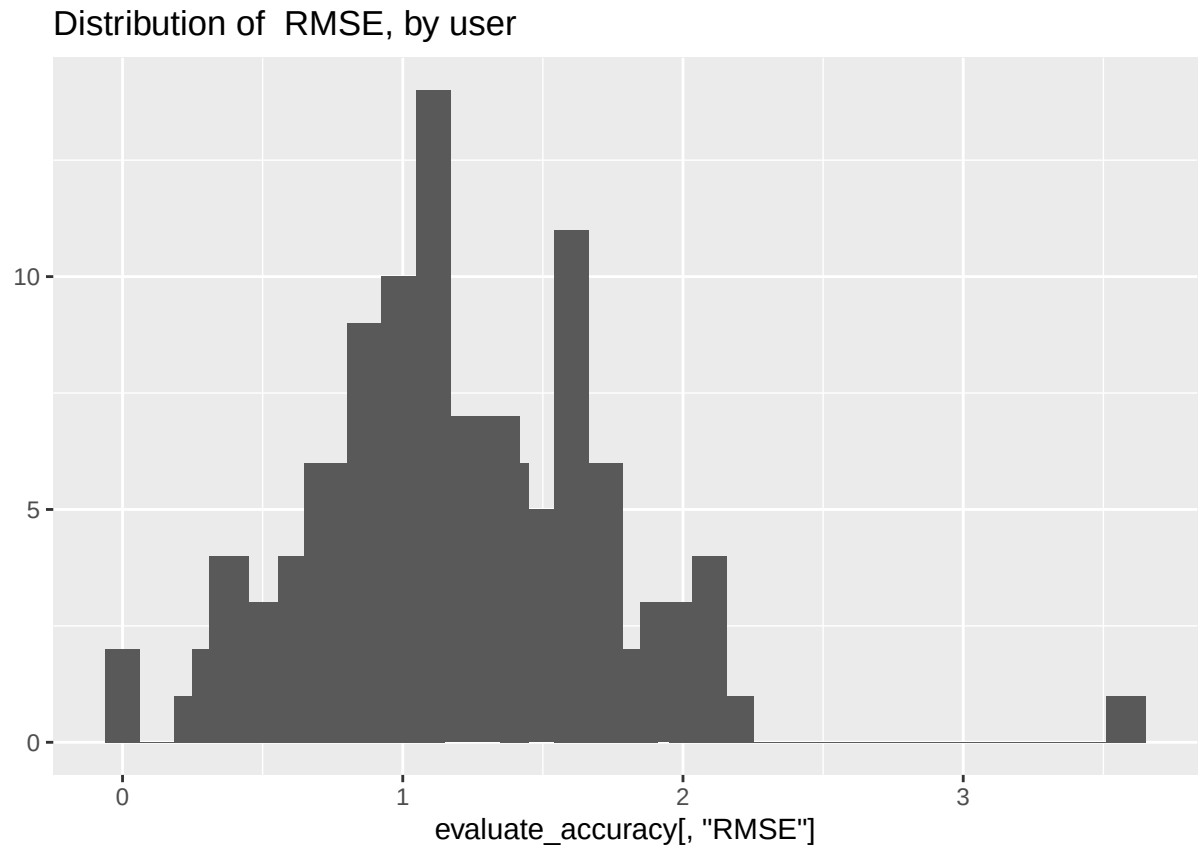
Most of the RMSEs (Root mean square errors) are in the range of 0.5 to 1.8:

```
##           RMSE       MSE      MAE
## [1,] 1.6583124  2.750000 1.250000
## [2,] 0.7071068  0.500000 0.500000
## [3,] 1.8719437  3.504173 1.627285
## [4,] 1.6955825  2.875000 1.083333
## [5,] 3.5707142 12.750000 3.500000
## [6,]       NaN       NaN      NaN
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 7 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 7 rows containing non-finite values (stat_bin).
```



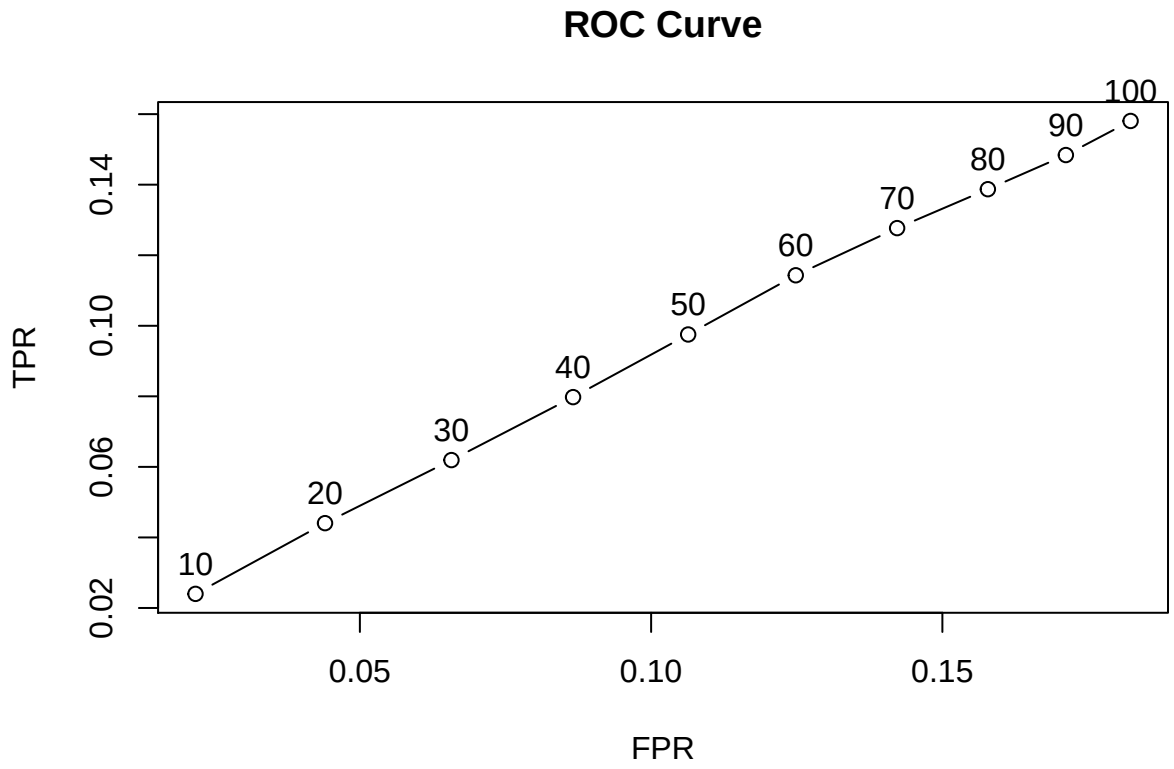

```
##      RMSE       MSE       MAE
## 1.2912642 1.6673632 0.9951832
```

Evaluation of Recommendations

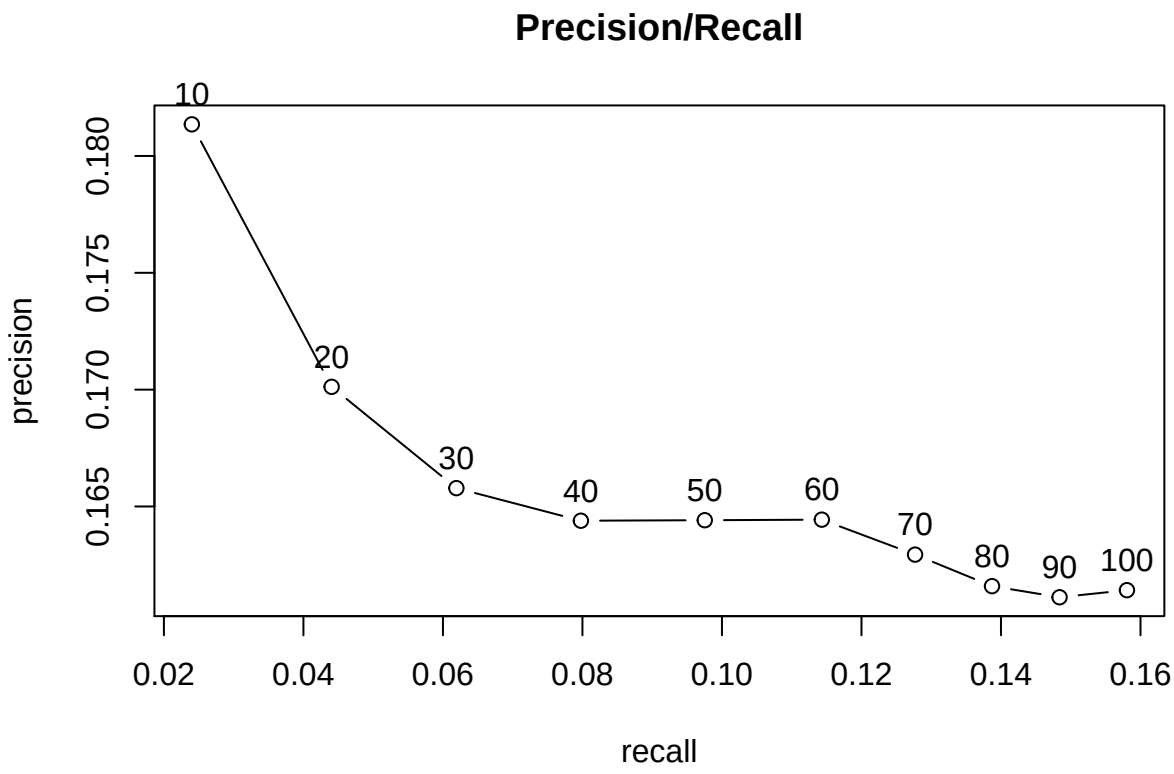
```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.26sec/0.04sec]
##   2  [0.33sec/0.01sec]
##   3  [0.26sec/0.03sec]
##   4  [0.26sec/0.04sec]
```

```
##          TP        FP       FN       TN precision     recall        TPR
## 10 1.942857  7.619048 78.44762 353.9905 0.2034653 0.02581728 0.02581728
## 20 3.676190 15.342857 76.71429 346.2667 0.1933359 0.04766793 0.04766793
## 30 5.190476 23.180952 75.20000 338.4286 0.1832300 0.06429239 0.06429239
## 40 6.638095 30.476190 73.75238 331.1333 0.1801943 0.07971827 0.07971827
## 50 7.961905 37.380952 72.42857 324.2286 0.1770836 0.09586570 0.09586570
## 60 9.200000 43.647619 71.19048 317.9619 0.1747005 0.10942461 0.10942461
##           FPR
## 10 0.02119761
## 20 0.04264550
## 30 0.06426196
## 40 0.08416630
## 50 0.10324298
## 60 0.12091208
```

```
##           TP        FP       FN       TN
## 10  7.066667  31.80000 295.1429 1433.990
## 20 13.228571  64.16190 288.9810 1401.629
## 30 19.209524  95.82857 283.0000 1369.962
## 40 25.047619 126.27619 277.1619 1339.514
## 50 30.838095 155.21905 271.3714 1310.571
## 60 36.266667 181.99048 265.9429 1283.800
```



ROC



From the above graphs, if a small percentage of rated movies is recommended, the precision decreases. At the same time, the higher percentage of rated movies that are recommended, the higher the recall.

Model Comparisons

In order to compare different models, create a basline measure out of the 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

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.23sec/0.03sec]
##   2  [0.2sec/0.02sec]
##   3  [0.2sec/0.05sec]
##   4  [0.24sec/0.01sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.28sec/0.03sec]
##   2  [0.3sec/0.03sec]
##   3  [0.28sec/0.03sec]
##   4  [0.2sec/0.03sec]
## UBCF run fold/sample [model time/prediction time]
```

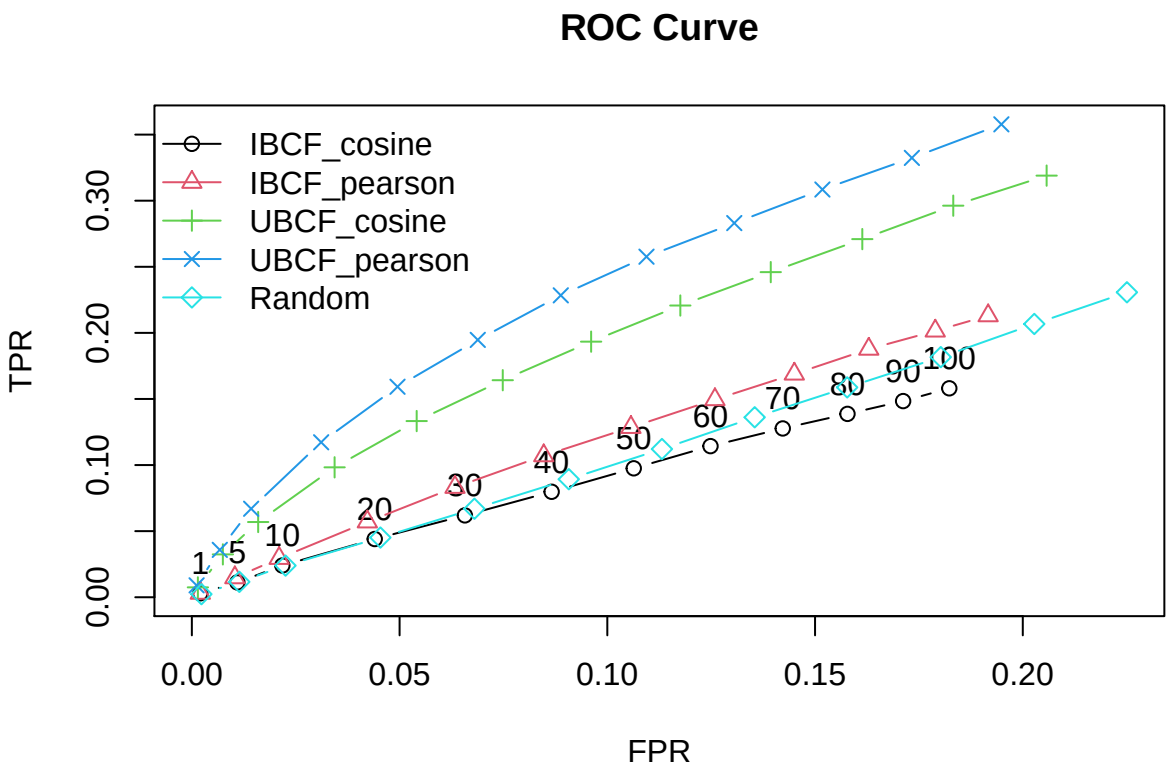
```
##   1  [0sec/0.13sec]
##   2  [0sec/0.11sec]
##   3  [0sec/0.12sec]
##   4  [0.02sec/0.1sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.02sec/0.12sec]
##   2  [0.02sec/0.11sec]
##   3  [0sec/0.14sec]
##   4  [0sec/0.14sec]
## RANDOM run fold/sample [model time/prediction time]
##   1  [0sec/0.03sec]
##   2  [0sec/0.03sec]
##   3  [0sec/0.03sec]
##   4  [0sec/0.03sec]
```

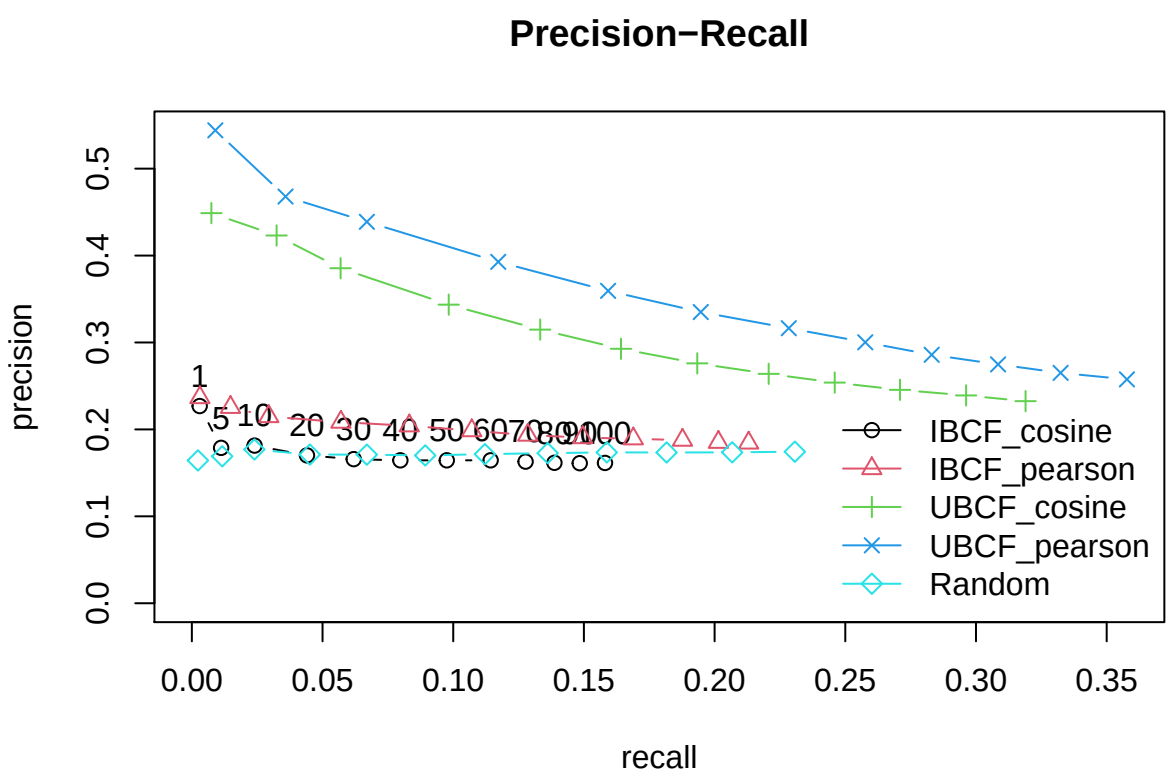
```
##  IBCF_cosine IBCF_pearson  UBCF_cosine UBCF_pearson       Random
##         TRUE         TRUE         TRUE         TRUE         TRUE
```

```
##    precision      recall         TPR         FPR
## 1  0.2268721 0.003004595 0.003004595 0.002030434
## 5  0.1787458 0.011207418 0.011207418 0.010958775
## 10 0.1813536 0.024003595 0.024003595 0.021781025
## 20 0.1701202 0.044045828 0.044045828 0.044018202
## 30 0.1657848 0.061929032 0.061929032 0.065719518
## 40 0.1643905 0.079778227 0.079778227 0.086603492
```

Plot - Best Fit Model

ROC curves & Precision/Recall Curves.


ROC Curve
IBCF_cosine
IBCF_pearson
UBCF_cosine
UBCF_pearson
Random
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
40
50
60
70
80
90
100



The UBCF with cosine distance performs the best out of all the models.

Optimiziation

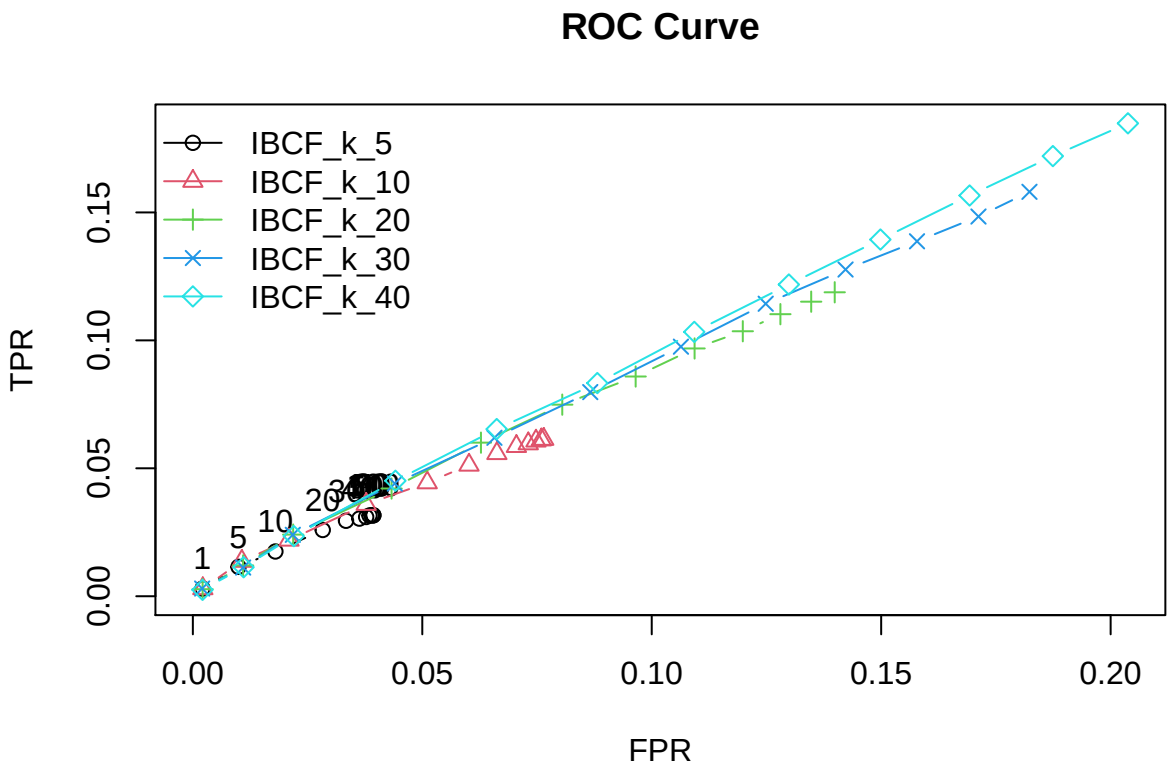
IBCF takes in consideration the k-nearest neighbor.

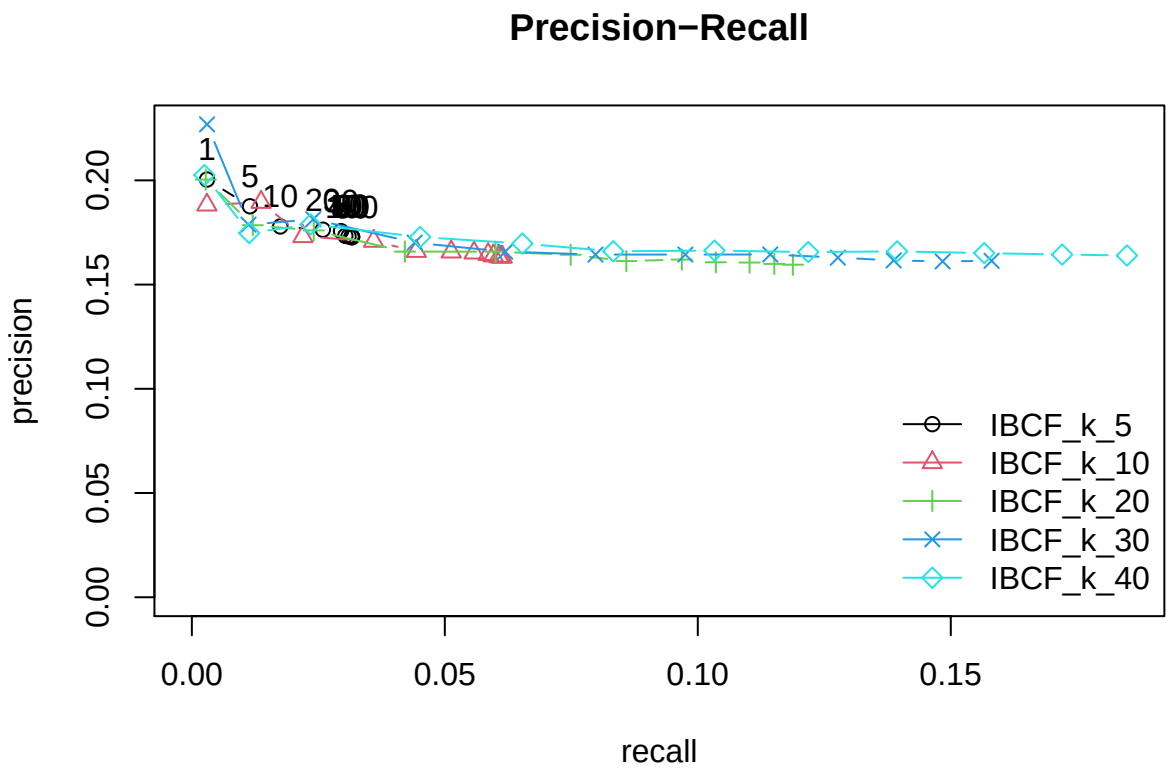
Tuning parameters ranging between 5 and 40.

Construct IBCF/Cosine Models with different values of the k-nearest neighbor.

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.25sec/0.01sec]
##   2  [0.23sec/0.03sec]
##   3  [0.3sec/0.01sec]
##   4  [0.28sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.27sec/0.01sec]
##   2  [0.2sec/0.02sec]
##   3  [0.19sec/0.01sec]
##   4  [0.22sec/0.01sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.17sec/0.03sec]
##   2  [0.25sec/0.02sec]
##   3  [0.28sec/0.02sec]
##   4  [0.27sec/0.01sec]
## IBCF run fold/sample [model time/prediction time]
```

```
##   1  [0.25sec/0.02sec]
##   2  [0.18sec/0.03sec]
##   3  [0.19sec/0.01sec]
##   4  [0.19sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.17sec/0.03sec]
##   2  [0.21sec/0.01sec]
##   3  [0.22sec/0.02sec]
##   4  [0.19sec/0.02sec]
```





The ROC Curve's Plot shows `k` having the biggest $AUC = 10$.

In additon. another good candidate is 5, because it can never have a high TPR.

Furthermore, the IBCF with k = 5 only recommends a few movies similar to the ratings.

Therefore, it cannot be the model used for recommendations.

Based on the Precision/Recall Plot, k should equal 10 to achieve the highest recall.

In that event, if we are more interested in the Precision, we set k to 5.

Conslusion

User-Based Collaborative Filtering

Strengths

- User-Based Collaborative Filtering provides recommendations that are complimentary to the item the user are observing.
- This prvides stronger Recommendations than an Item-Based Recommender.

Weaknesses

- User-based Collaborative Filtering is a memeory intensive Collaborative Filtering that uses all user data in the database to create Recommendations.
- However, comparing the pairwise correlation of every user in your dataset is not scalable. The more users the more time it would take to compute the Recommendations.

Shiny Application

For the project, the developed and evaluated a Collaborative Filtering Recommender System was implemented for the Recommendation of Movies.

The deployment of a Shiny R Application demonstrates the User-Based Collaborative Filtering Approach for the Recommendation Model.

Implementation

What2Watch

Appendix

Source Code