MOVIE RECOMMENDATION SYSTEM, 20240

Overview: The aim of this project is to develop a movie recommender system using the principle of collaborative filtering, for recommending movies to a user. Principle on which the collaborative filtering recommender (CFR) works is- if two people liked same thing in the past, then there is high chance that they will like similar things in the future as well. For example- if the movies seen by user1 and user2 in the past are similar, and user1 has recently seen a movie which user2 has not seen yet, our recommendation system will recommend that movie to user2.

Link for the full R code: https://github.com/SURYA-LAMICHANEY/MovieRecommendationSystem

Required libraries:

```
library(recommenderlab)
library(ggplot2)
library(data.table)
library(reshape)
```

Dataset: I have used MovieLens dataset from Kaggle. The dataset consists of four csv file- links.csv, ratings.csv, tags.csv and movies.csv out of which I have used only two csv files for building the recommendation system- movies.csv and ratings.csv.

6 point summary and first few rows of movies data:

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

One important thing to note here is that one movie can fall into many genres.

6 point summary and first few rows of ratings data:

```
     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 <int>	movieId <int>	rating <dbl>	timestamp <int>
1	16	4.0	1217897793
1	24	1.5	1217895807
1	32	4.0	1217896246
1	47	4.0	1217896556
1	50	4.0	1217896523
1	110	4.0	1217896150

MovieId from both the data (movies and ratings) is used to match corresponding movies and ratings.

Data pre-processing:

Before proceeding to creating the recommender system some data pre-processing is required. I have created a matrix of movies with corresponding genres using a one-hot encoding.

```
Action Adventure Animation Children Comedy Crime Documentary Drama Fantasy
     0         1         1        1      1     0           0     0       1
     0         1         0        1      0     0           0     0       1
     0         0         0        0      1     0           0     0       0
     0         0         0        0      1     0           0     1       0
     0         0         0        0      1     0           0     0       0
     1         0         0        0      0     1           0     0       0
Film-Noir Horror Musical Mystery Romance Sci-Fi Thriller War Western
        0      0       0       0       0      0        0   0       0
        0      0       0       0       0      0        0   0       0
        0      0       0       0       1      0        0   0       0
        0      0       0       0       1      0        0   0       0
        0      0       0       0       0      0        0   0       0
        0      0       0       0       0      0        1   0       0
```

Here, 1 means the corresponding movie falls into that genre and 0 means it does not.

Creating a matrix for searching a movie by its genre:

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

Now, it will be very easy to search for a movie by its genre. For example if we want to search for an action movie, the model will simply iterate through the matrix and return all the movies for which there is 1 under the genre Action.

Converting ratings matrix into a realRatingMatrix type for recommendation:

For the ratings data to be used with recommenderlab, we need to create a sparse matrix of realRatingMatrix type out of rating matrix.

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

Parameters exploration of recommendation models:

I will be using two models namely Item Based Collaborative Filtering(IBCF) and User Based Collaborative Filtering(UBCF). The parameters of these models are below:

```
> recommenderModels$IBCF_realRatingMatrix$parameters
$k
[1] 30

$method
[1] "Cosine"

$normalize
[1] "center"

$normalize_sim_matrix
[1] FALSE

$alpha
[1] 0.5

$na_as_zero
[1] FALSE
```

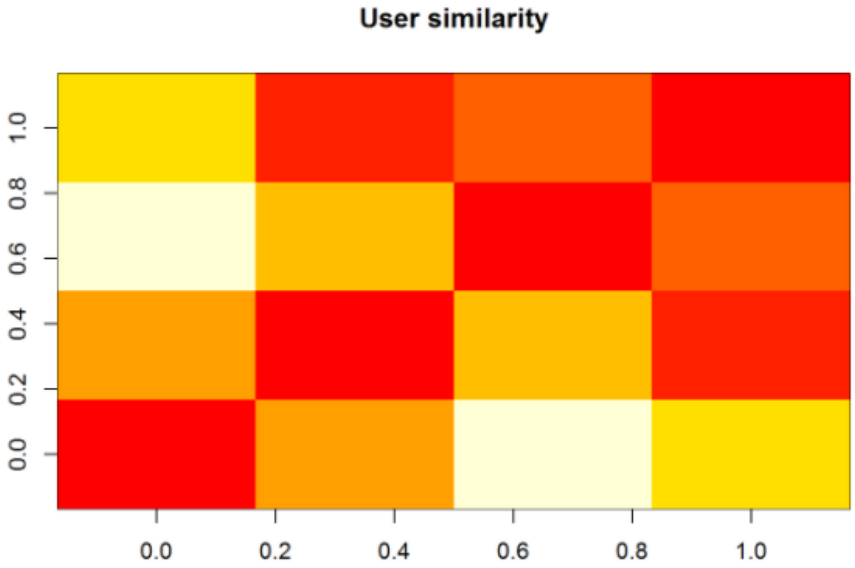
```
> recommenderModels$UBCF_realRatingMatrix$parameters
$method
[1] "cosine"

$nn
[1] 25

$sample
[1] FALSE

$weighted
[1] TRUE

$normalize
[1] "center"

$min_matching_items
[1] 0

$min_predictive_items
[1] 0
```

Exploring similarity data:

The similarity between the items or users are the key principles on which the collaborative filtering algorithms are based. There is a similarity function in recommenderlab for this purpose. Three methods which is used to calculate similarities are Jaccard, Cosine and Pearson.

Here, I have used cosine distance to measure similarity between the first four users. Below is the similarity matrix and its visualization.

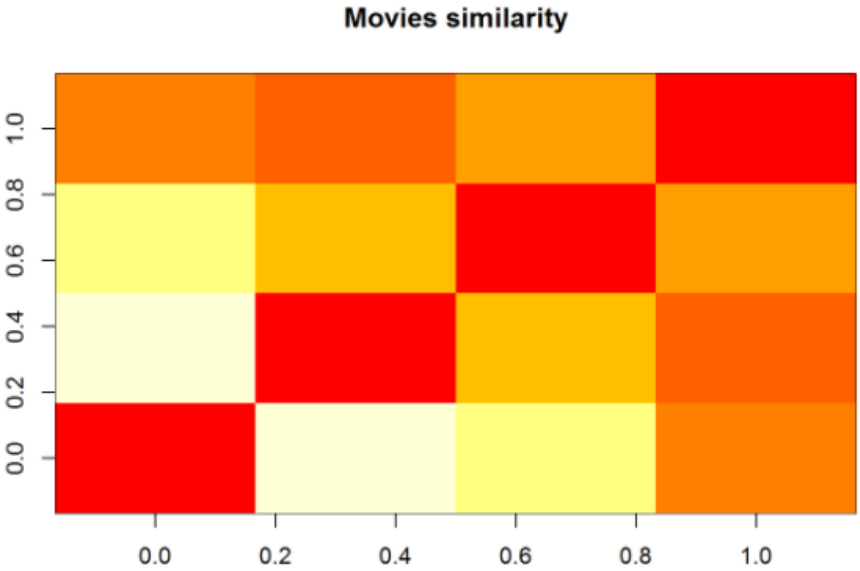
```
          1         2         3         4
1 0.0000000 0.9760860 0.9641723 0.9914398
2 0.9760860 0.0000000 0.9925732 0.9374253
3 0.9641723 0.9925732 0.0000000 0.9888968
4 0.9914398 0.9374253 0.9888968 0.0000000
```



In the above matrix, each row and column represents a user and each cell explains the similarity between two users. The darker the cell is, the more similarity among users.

Similarly I have computed the similarity between 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



Some more data exploration:

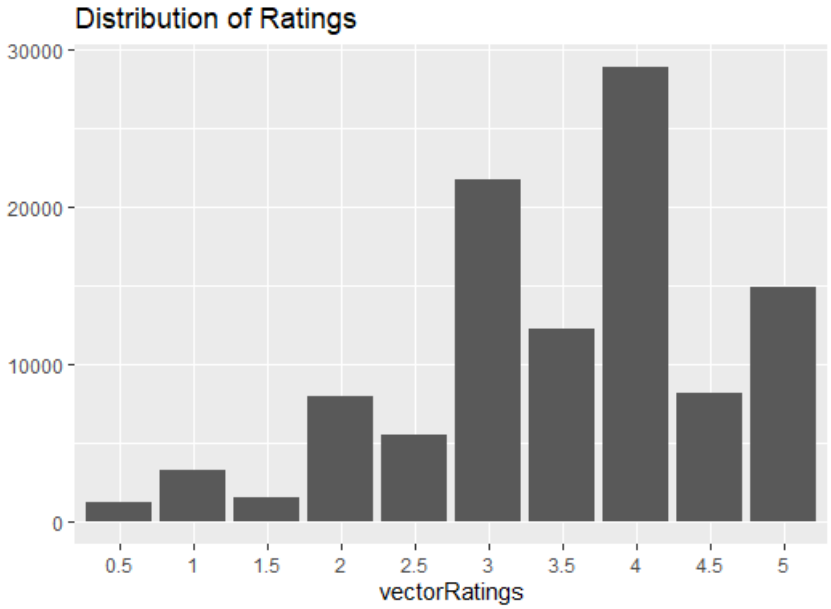
Exploration of values of ratings data-

```
> unique(vectorRatings)
 [1] 0.0 5.0 4.0 3.0 4.5 1.5 2.0 3.5 1.0 2.5 0.5
>
> tableRatings<-table(vectorRatings)
> tableRatings
vectorRatings
      0     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
6791761    1198    3258    1567    7943    5484   21729   12237   28880    8187   14856
>
```

So there are total 11 unique ratings values that is from 0.0 to 5.0. We can see the counts of each ratings value in the data. Rating value 0 means either missing values or not rated movies.

Distribution of Ratings:

Before visualizing I have removed 0 rating values because they really do not make any difference.



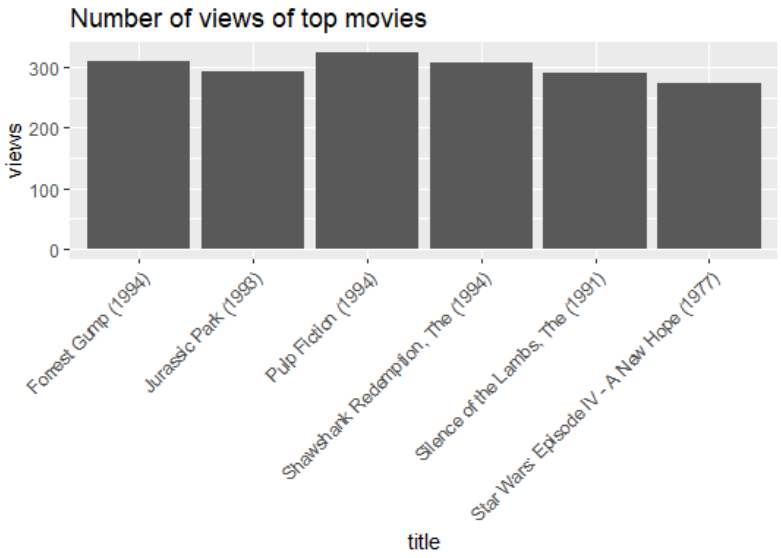
The highest number of movies have been rated 4. Most of the movies have been rated above 3.

Top viewed movies:

Following 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)
>
```

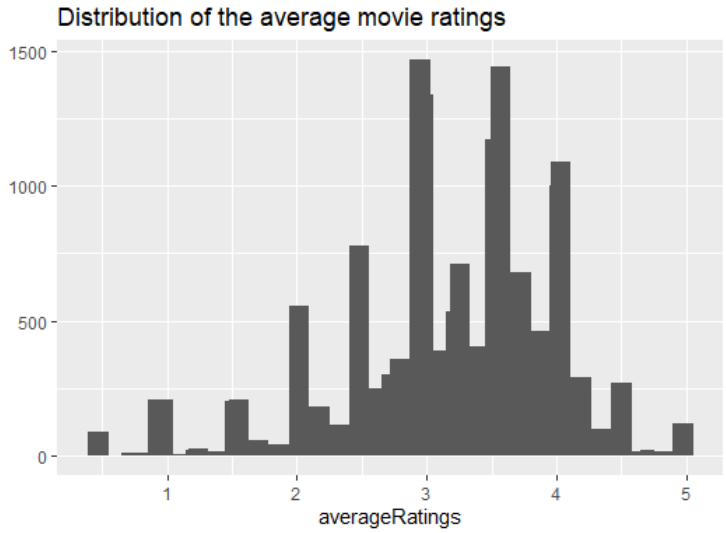
Frequency of most viewed movies

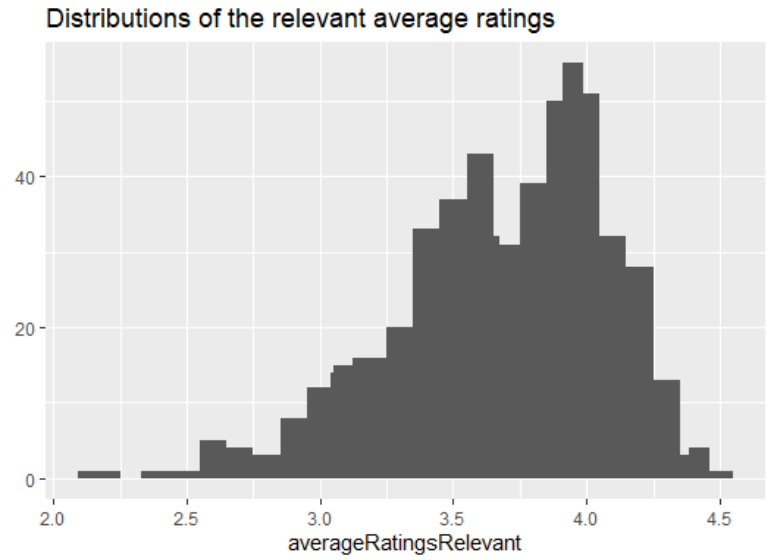


So all of the top viewed movies have been seen by more than 290 people.

Average movie rating distribution:

Here I am identifying top-rated movies by looking at their average ratings.



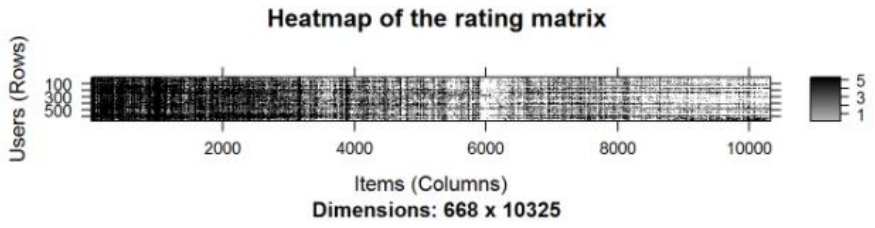


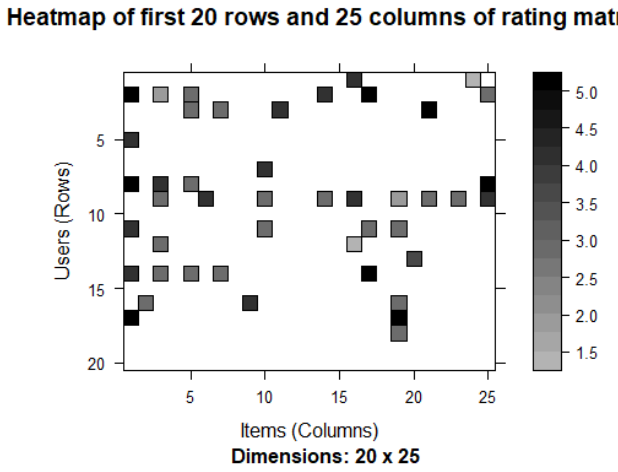
In the first plot, the highest average movie rating is around 3 but there are very few movies which are rated either 0 or 5. So these are outliers. In second plot, I have removed 0and 5 to keep just relevant ratings and now the highest average movie rating is around 4.

Rating matrix heatmap:

The whole matrix can be visualized by building a heatmap whose color represents the ratings. Each row represents a user, column a movie and cell its ratings.

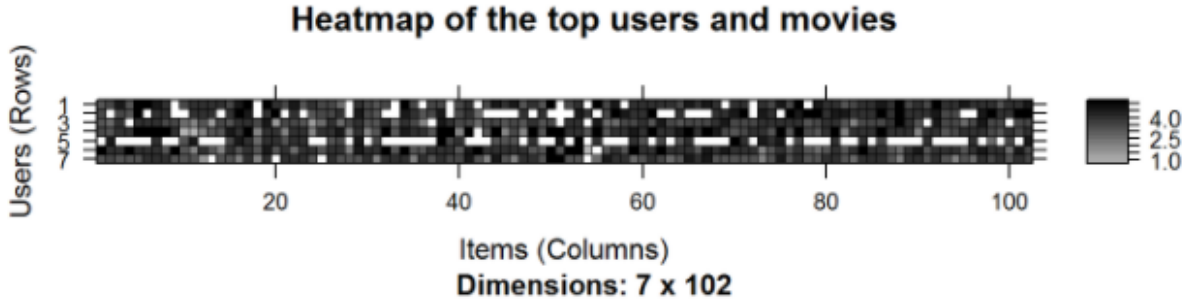
As we can see the first heatmap is very hard to read, I have created another heatmap which is built using first 20 users and 25 movies .





Some people have seen greater number of movies as compared to others. So instead of picking some random user and movie, I want to pick the most relevant movies and users. So I have identified the movies which was watched by many people and users who watched many movies. This can be achieved by identifying the minimum movies watched by a user, the minimum users for each movie and selecting the users and movies satisfying both the criteria.

```
> print("Minimum number of movies per user:")
[1] "Minimum number of movies per user:"
> minMovies
   99%
1198.17
> print("Minimum number of users per movie:")
[1] "Minimum number of users per movie:"
> minUsers
99%
115
```



Most of the users have seen all the top movies. Some rows are full of different dark shades which means those users have seen all the movies and given ratings accordingly. Many columns are darker than others which mean those movies got higher ratings.

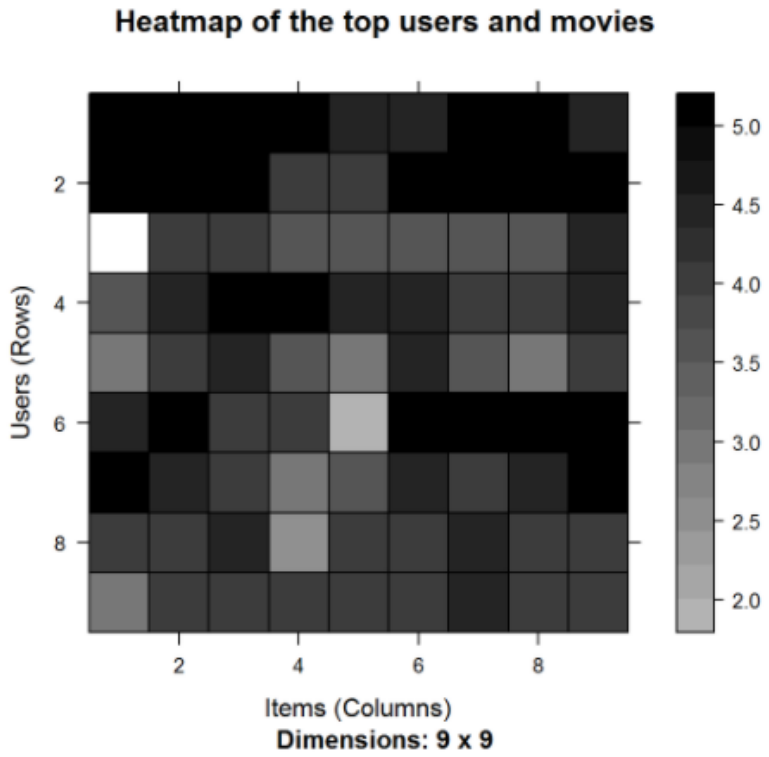
Preparation of data:

The process of data preparation consists of selecting relevant data, normalizing it and binarizing it. For selecting the most relevant data, I am taking minimum users for every rated movies as 50 and minimum views for every movie as 50.

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

So the selection of data using above conditions fetched me 447 movies and 420 users.

Looking at the top 2% users and movies from the most relevant data;



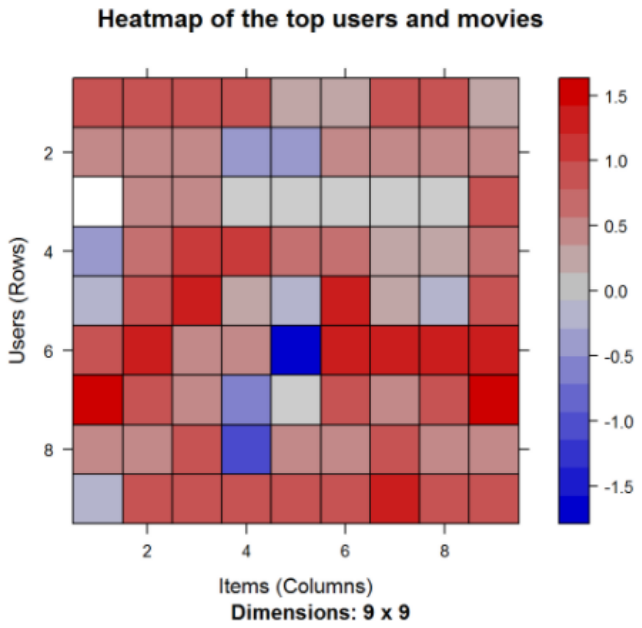
Some users have rated all the movies high. Users 1 and 2 have rated movie1 as 5 but user 3 has rated movie1 less than 2.

Data normalization:

Since many users have given high or low ratings for movies they watched, it can bias the result. So I have normalized the data such that average rating per user is 0.

```
> ratingsMoviesNorm<-normalize(ratingsMovies)
> sum(rowMeans(ratingsMoviesNorm) > 0.00001)
[1] 0
```

So the sum is 0, as expected.

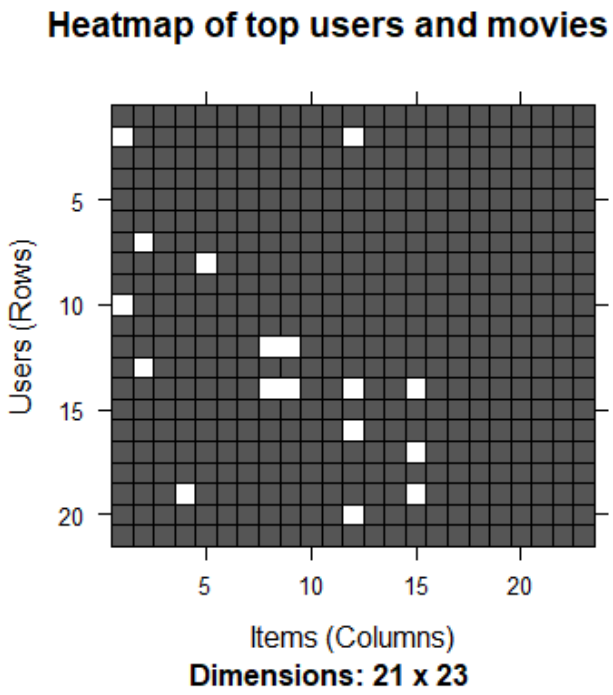


Data binarization:

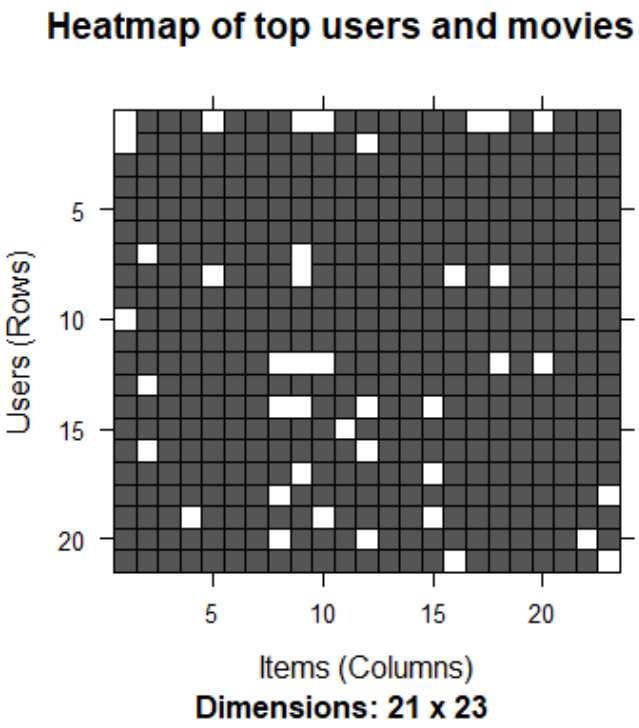
Since there are recommendation models which work only on binary data, it is wise to convert the data into binary values by defining a table with only 1s and 0s. 0 can be treated as a missing value or a bad rating.

So I defined two matrices for two different approaches and visualize 5% portion of each.

First approach- for a watched movie, the matrix is defined equal to 1:



Second approach- if any cell has more rating than a defined value(3 in this case), define a matrix equal to 1:



Second heatmap contains more white cells which means there more movies that are not rated or poorly rated than those movies which were not seen by users.

Collaborative filtering model(ITEM-based)

Defining training/test sets- I have divided the whole dataset set into 80/20 training set/test set.

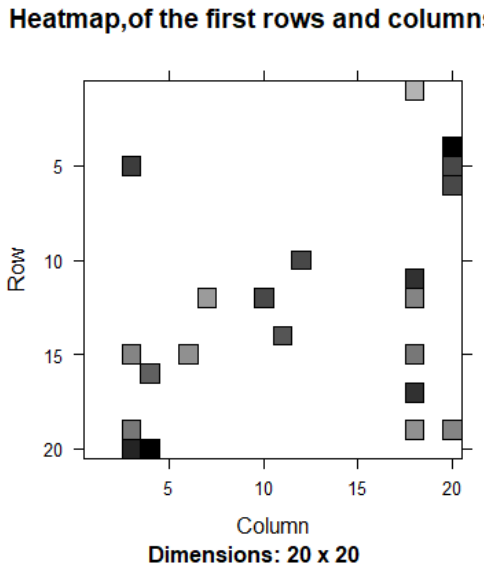
Recommendation model:

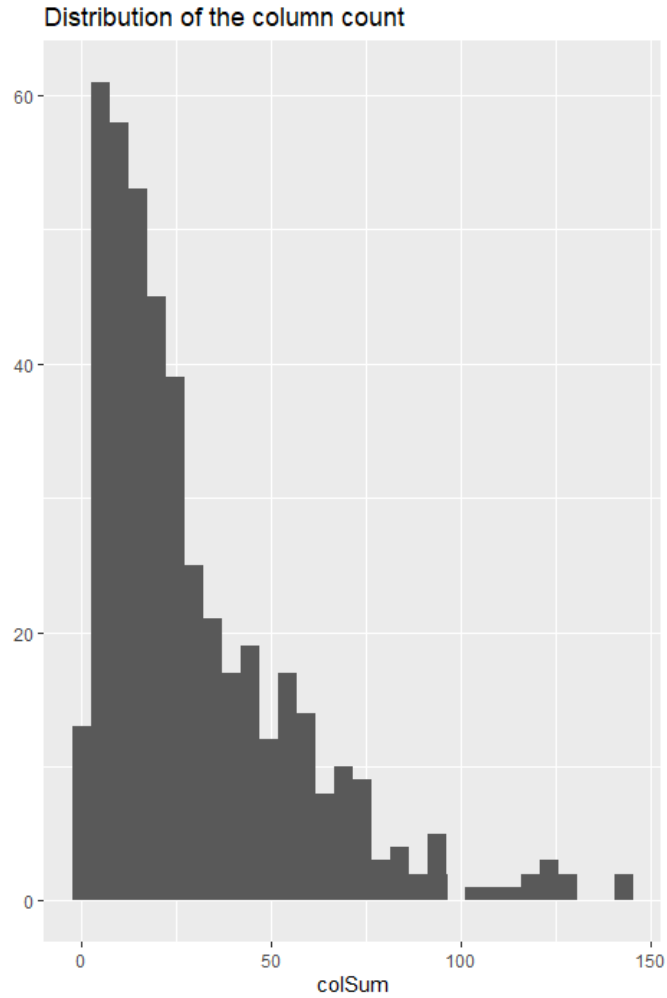
In IBCF model, the default parameters are - k which is the number of items to be taken for calculating the similarity among them in the first step. Then the algorithm finds k most similar items for each item and keeps them as integers. Another default parameter is – method which is a similarity function. It can be cosine, pearson or Jaccard. I have taken k=30 and method =cosine.

```
> reccModel<-Recommender(data=dataTrain,
+                        method="IBCF",
+                        parameter=list(method="Cosine",k=30))
> reccModel
Recommender of type 'IBCF' for 'realRatingMatrix'
learned using 342 users.
```

Exploring the model:

```
> class(modelDetails$sim)
[1] "dgCMatrix"
attr(,"package")
[1] "Matrix"
> dim(modelDetails$sim)
[1] 447 447
```





The model itself creates a similarity matrix called dcgMatrix. The dimension of this matrix is equal to the number of items that is 447x447. Most of the values are 0 in the heatmap of first 20 items because each row contains only k(30) elements that are greater than 0. The number of non zero elements for columns depends on the number of times the corresponding item was included in the top k of another movie. Therefore the matrix need not be necessarily symmetric.

The distribution of the column count shows that very few movies are there which are actually similar to many other movies.

Implementing the designed system using the dataset: Now we are ready for recommending the movies to the users using test data. Each user will be recommended 10 movies.

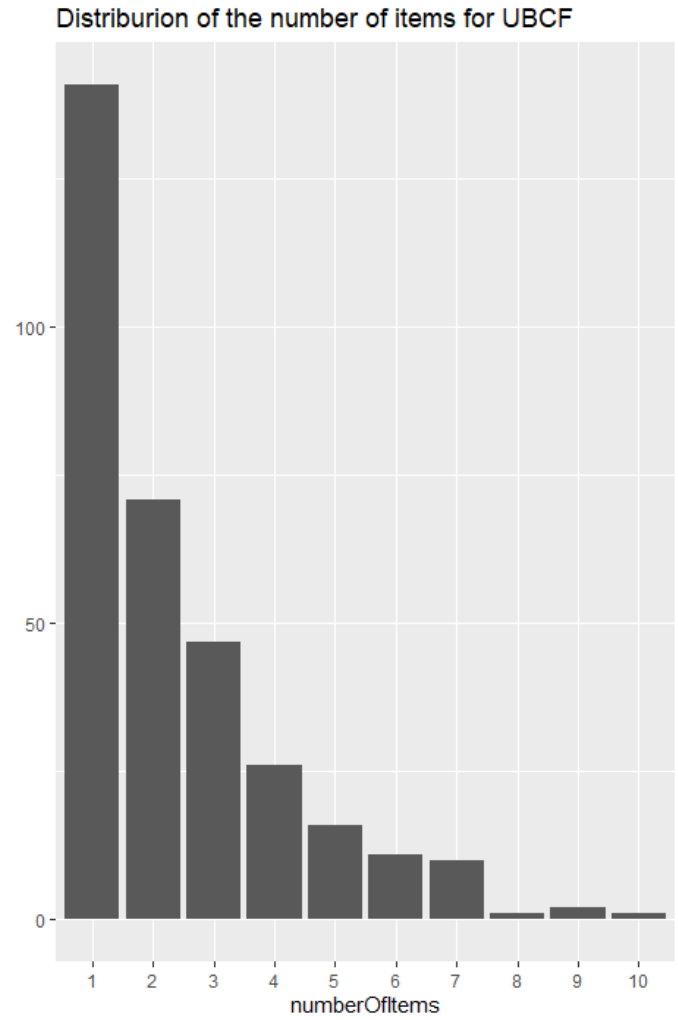
For each user, rated movies are extracted by the algorithm. For each movie, the algorithm uses the similarity matrix to find all the similar items for it. After running the algorithm, we get:

```
> reccPredicted<-predict(object = reccModel,
+                        newdata=dataTest,
+                        n=nRecommended)
> reccPredicted
Recommendations as 'topNList' with n = 10 for 78 users.
```

So the top 10 recommendations for the first user is:

```
 [1] "Toy Story (1995)"          "Grumpier Old Men (1995)"   "Broken Arrow (1996)"       "First Knight (1995)"
 [5] "Judge Dredd (1995)"        "Stargate (1994)"           "Santa Clause, The (1994)"  "Demolition Man (1993)"
 [9] "Lawrence of Arabia (1962)" "Fantasia (1940)"
```

Now, I have identified the most recommended movies.



	Movie title	No. of items
3	Grumpier Old Men (1995)	10
34	Babe (1995)	9
95	Broken Arrow (1996)	9
2	Jumanji (1995)	8

Most of the movies have been recommended a few times only.

USER-based collaborative filtering model

Building the recommendation system:

The default parameters of UBCF model is – method which is a similarity function and nn which is number of similar users. I have used default values for these parameters on the training set.

```
Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 342 users.
> modelDetails <- getModel(reccModel)
> #names(model_details)
> modelDetails$data
342 x 447 rating matrix of class 'realRatingMatrix' with 31067 ratings.
Normalized using center on rows.
```

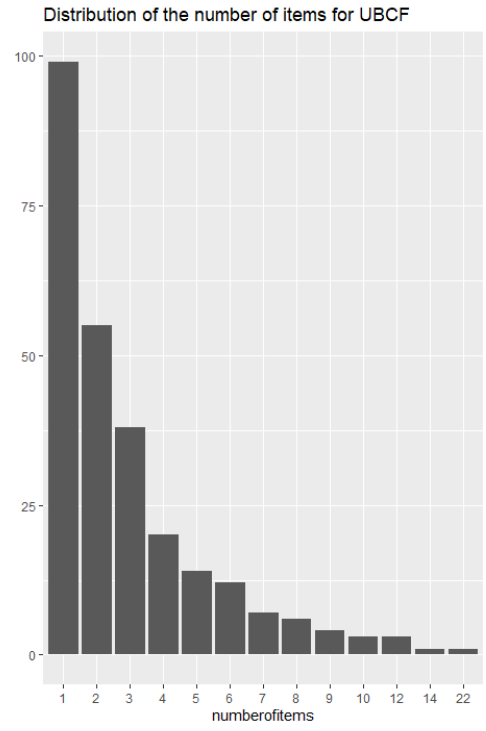
Implementing the designed system using the test set:

Similar to IBCF, I have determined the top 10 recommendation for each user in the test set.

```
Recommendations as 'topNList' with n = 10 for 78 users.
```

The top 10 recommendations for first user is:

	Movie title	No of items
5669	Bowling for Columbine (2002)	22
1343	Cape Fear (1991)	14
62	Mr. Holland's Opus (1995)	12
920	Gone with the Wind (1939)	12



The distribution of the number of items for UBCF shows that compared to IBCF, some movies got recommended much more than the others .

Comparision of results of IBCF and UBCF models:

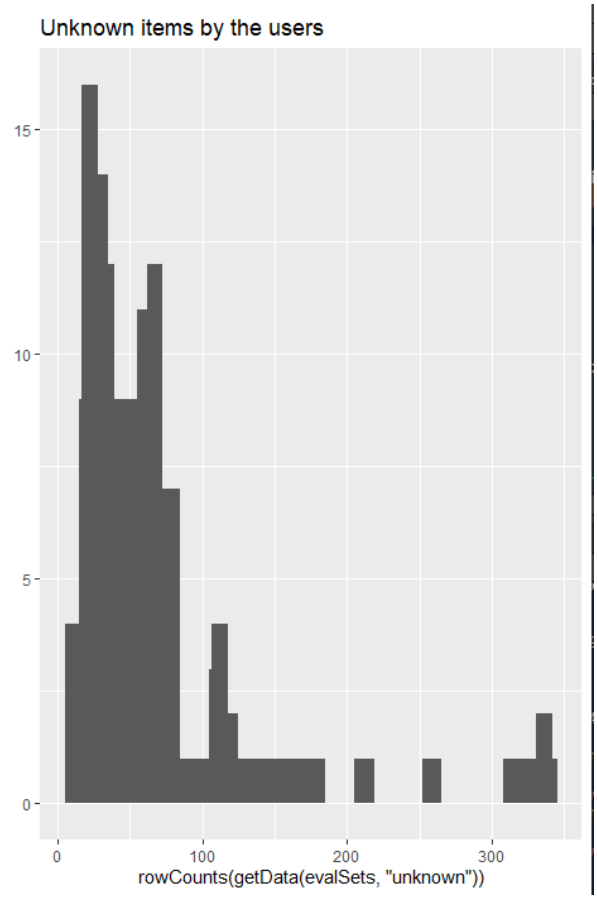
Steps:

a) Preparing data for evaluation of models
b) Evaluating the performance
c) Choosing the best model
d) Optimizing model parameters

Preparing data for evaluation of model:

Again I have split the data into 80% training set and 20% test set.

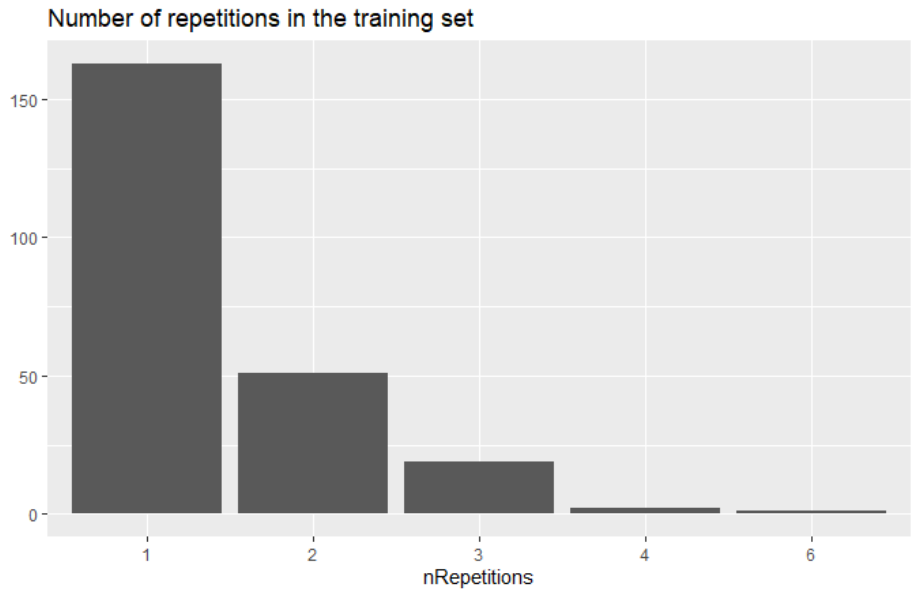
```
Evaluation scheme with 5 items given
Method: 'split' with 1 run(s).
Training set proportion: 0.800
Good ratings: >=3.000000
Data set: 420 x 447 rating matrix of class 'realRatingMatrix' with 38341 ratings.
>
> getData(evalSets,"train")
336 x 447 rating matrix of class 'realRatingMatrix' with 31967 ratings.
> getData(evalSets,"known")
84 x 447 rating matrix of class 'realRatingMatrix' with 420 ratings.
> getData(evalSets,"unknown")
84 x 447 rating matrix of class 'realRatingMatrix' with 5954 ratings.
>
```



The distribution unknown items by the user shows the number of movies which is not seen by the users.

Bootstraping the data:

One more way of splitting the data is bootstrapping.



The distribution shows that there are fewer than six times that any movie have been sampled.

Validating the models with cross-validation:

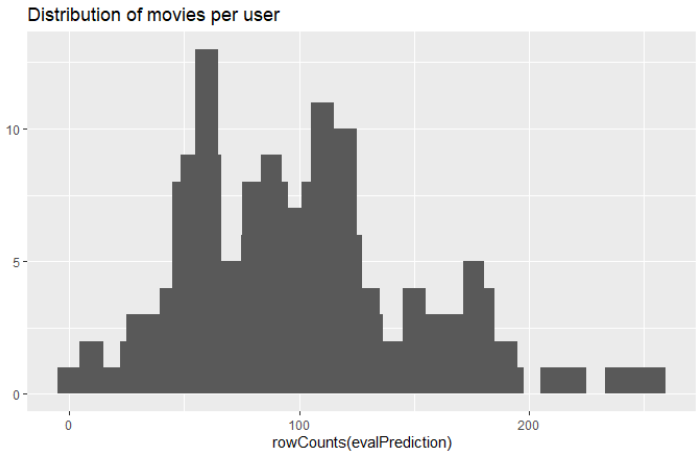
Though it is computationally complex, the k-fold cross-validation is the most accurate one. Here, we will split the data into k parts, take a part as test set and evaluate the accuracy. We do this for all k parts of data and find the average accuracy.

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

Here, I have taken k=4, which has divided the data into 4 chunks of 315 values each.

Evaluating the ratings:

Redefining the evaluation set, then building a IBCF model and creating a matrix with predicted ratings.



The plot shows the distribution of movies per user in the matrix of predicted ratings.

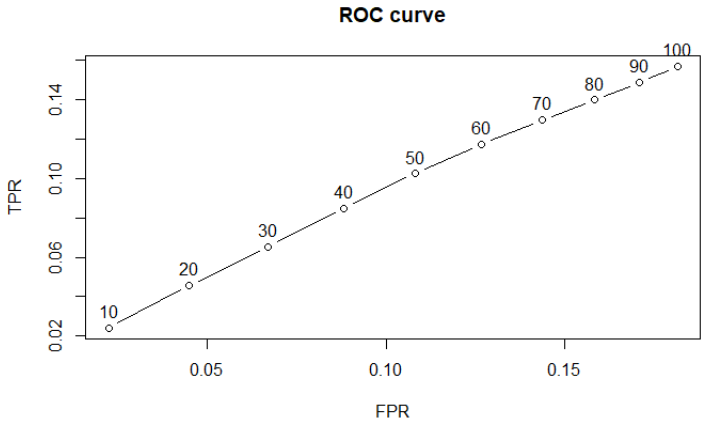
```
          RMSE       MSE       MAE
[1,] 1.2595682 1.5865119 1.1367307
[2,] 2.8577380 8.1666667 2.3333333
[3,] 0.6742827 0.4546572 0.4579801
[4,] 0.9013878 0.8125000 0.8750000
[5,] 1.0000002 1.0000004 0.7779857
[6,] 1.0000000 1.0000000 1.0000000
```

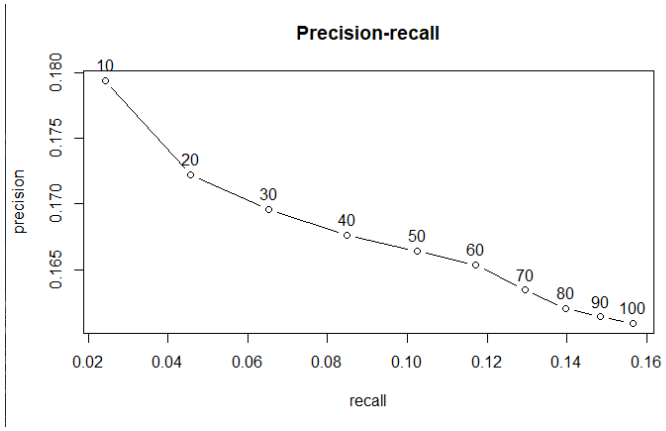
Here, for each user I have calculated their accuracy measures. The range of Root Mean Square Error is from 0.6 to 2.8. For the whole model, having one performance index is better. So I have calculated the average indices by specifying byUser as FALSE.

```
     RMSE       MSE       MAE
1.2514455 1.5661158 0.9136473
```

Evaluating the recommendations:

```
          TP       FP       FN       TN
10  7.114286  32.57143 295.1524 1433.162
20 13.619048  65.54286 288.6476 1400.190
30 19.990476  97.76190 282.2762 1367.971
40 25.990476 128.80000 276.2762 1336.933
50 31.666667 157.97143 270.6000 1307.762
60 36.780952 185.22857 265.4857 1280.505
```





The ROC curve shows that the test is not so accurate because the line is almost diagonal. Precision-recall plot shows that precision is very high at the beginning then it slowly decreases.

Comparing the models:

Following are the different model definitions for comparison:

IBCF using Cosine as distance function,

IBCF using Pearson correlation as the distance function,

UBCF using Cosine as the distance function

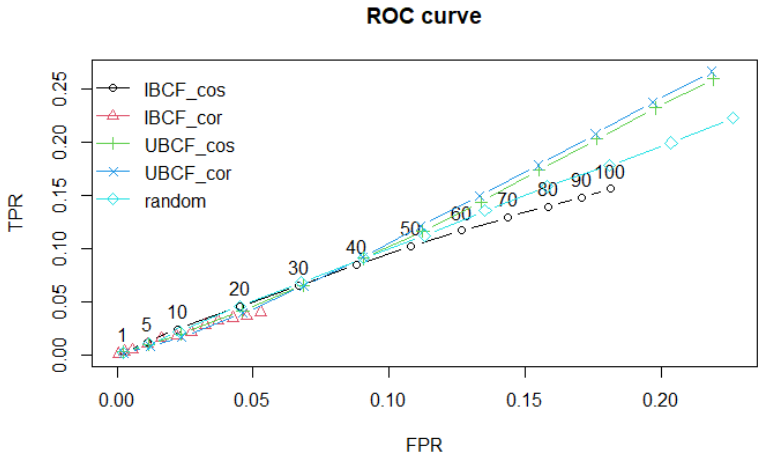
UBCF using Pearson correlation as the distance function.

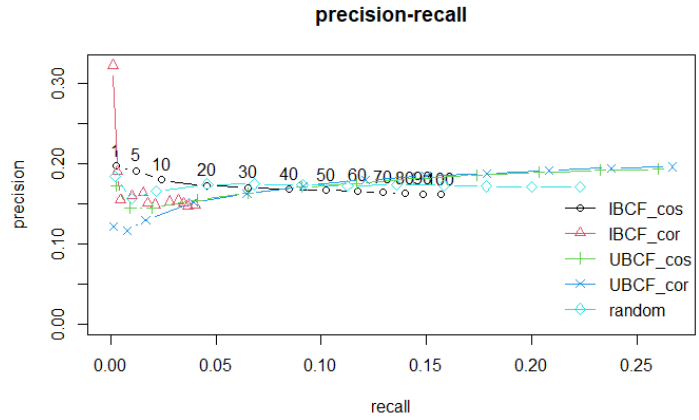
Random recommendation to act as a baseline.

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

Identifying the suitable model:

I have compared the above mentioned 5 models by plotting the ROC-curve and precision-recall. Whichever model gives the higher values will be best model.





Both the ROC-curve and precision-recall shows that UBCF with Cosine distance is the most suitable model.

Conclusion:

A user based collaborative filtering system is the most suitable model for creating a movie recommender system in my case. I have used the same model to create the recommender system using R Shiny for designing a dashboard.