NAME: Hammad Ansari **ROLL NO: 2018450002**

Experiment No. 12

Title: Recommender System Using Collaborative Filtering

1. Introduction

   Collaborative filtering is a technique used by recommender systems for predicting the interests of one user based on the preference information of other users. This project is an implementation of a Movie Recommender System that uses the following techniques:

   - Item-Item Collaborative Filtering
   - User-User Collaborative Filtering

   This implementation uses the recommenderlab package in R. The MovieLense dataset is included with this package and is used here to train, predict, and evaluate the models.

   ```
   library(dplyr)

   library(ggplot2)

   library(knitr)

   library(recommenderlab)
   ```

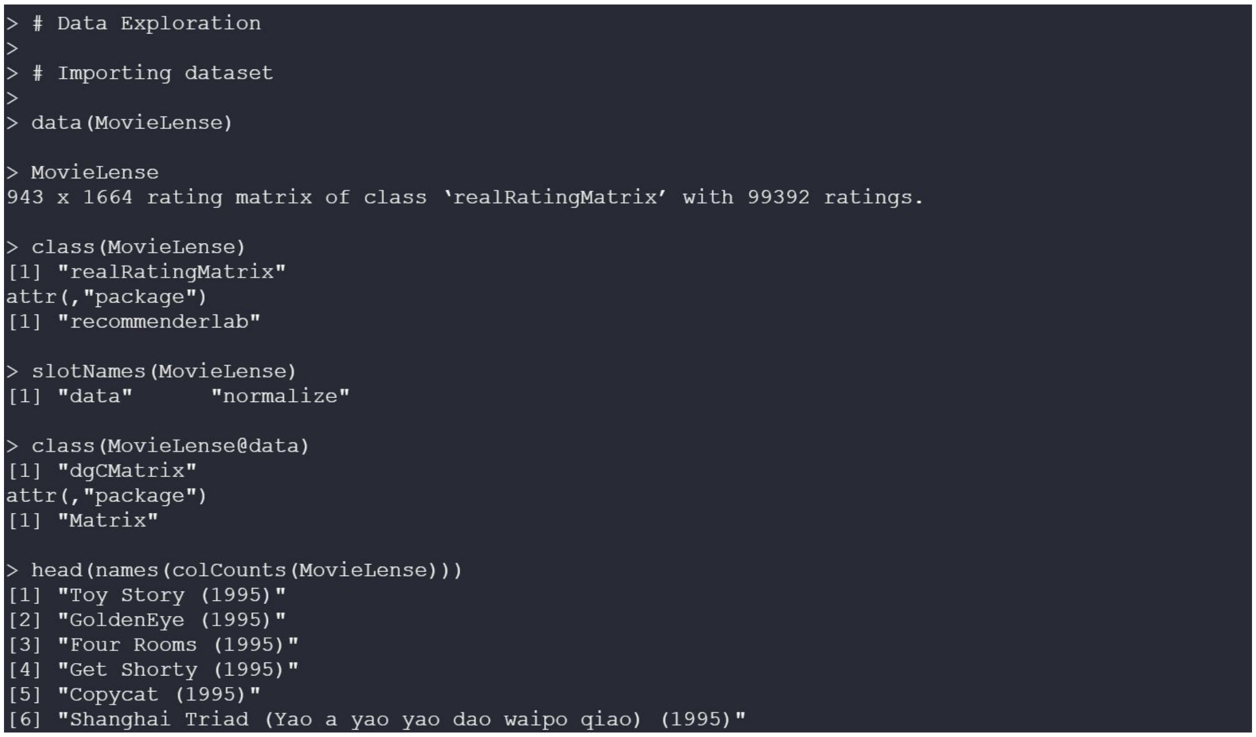
2. Data Exploration

   The starting point in collaborative filtering is a rating matrix in which rows correspond to users and columns correspond to items. This matrix is implemented in the MovieLense data object.

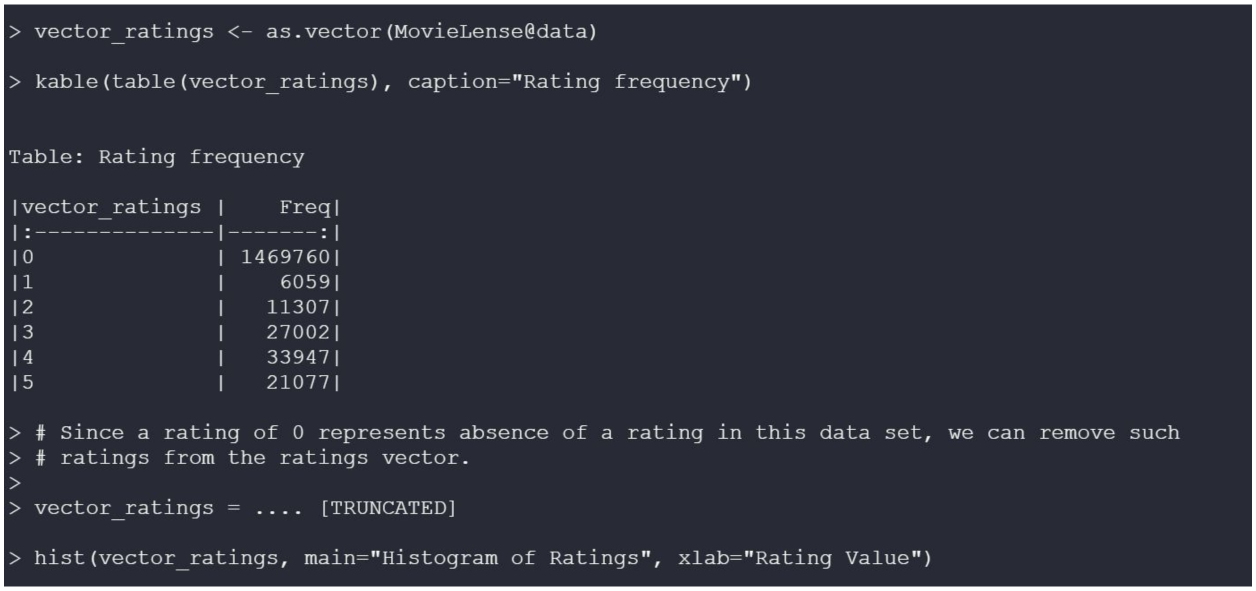
   We now load and inspect the MovieLense data object.

   ```
   # Importing dataset

   data(MovieLense)

   MovieLense

   class(MovieLense)
   ```

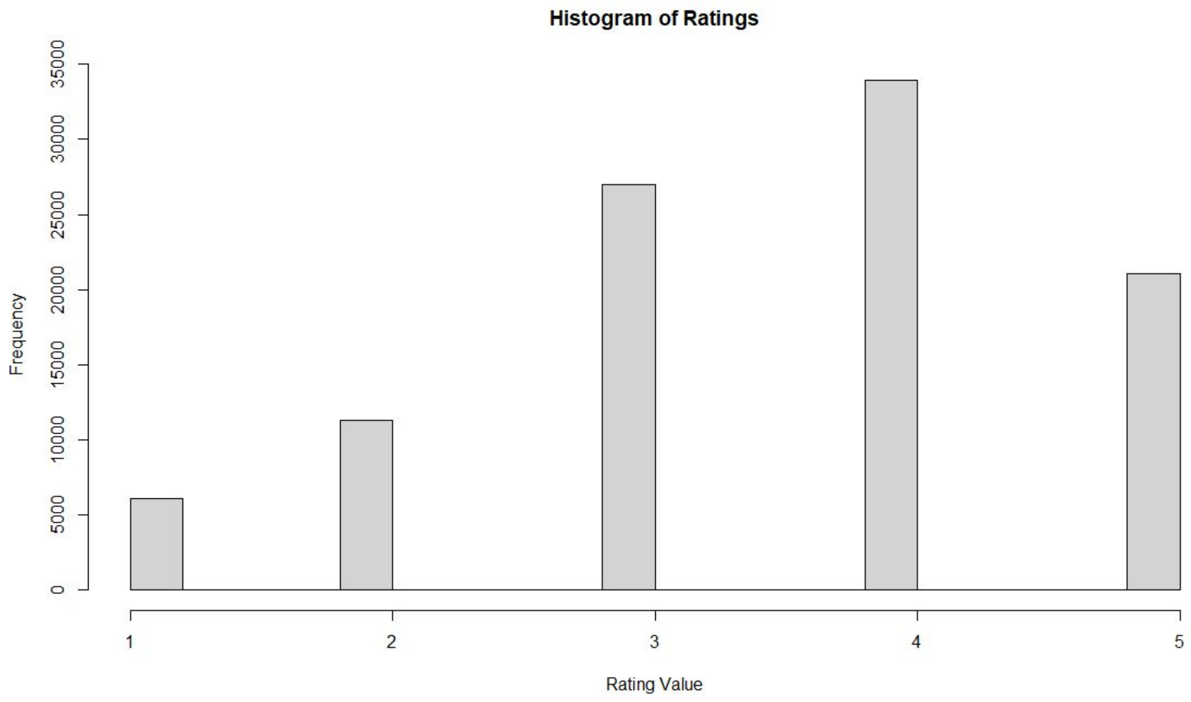
```
slotNames(MovieLense)

class(MovieLense@data)

head(names(colCounts(MovieLense)))

vector_ratings <- as.vector(MovieLense@data)

kable(table(vector_ratings), caption = "Rating frequency")

# Since a rating of 0 represents the absence of a rating in this data set, we can
remove such

# ratings from the ratings vector.

vector_ratings = vector_ratings[vector_ratings != 0]

hist(vector_ratings, main = "Histogram of Ratings", xlab = "Rating Value")
```



```
> # Data Exploration
>
> # Importing dataset
>
> data(MovieLense)

> MovieLense
943 x 1664 rating matrix of class ‘realRatingMatrix’ with 99392 ratings.

> class(MovieLense)
[1] "realRatingMatrix"
attr(,"package")
[1] "recommenderlab"

> slotNames(MovieLense)
[1] "data"      "normalize"

> class(MovieLense@data)
[1] "dgCMatrix"
attr(,"package")
[1] "Matrix"

> head(names(colCounts(MovieLense)))
[1] "Toy Story (1995)"
[2] "GoldenEye (1995)"
[3] "Four Rooms (1995)"
[4] "Get Shorty (1995)"
[5] "Copycat (1995)"
[6] "Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)"
```



```
> vector_ratings <- as.vector(MovieLense@data)

> kable(table(vector_ratings), caption="Rating frequency")


Table: Rating frequency

|vector_ratings |    Freq|
|:--------------|-------:|
|0              | 1469760|
|1              |    6059|
|2              |   11307|
|3              |   27002|
|4              |   33947|
|5              |   21077|

> # Since a rating of 0 represents absence of a rating in this data set, we can remove such
> # ratings from the ratings vector.
>
> vector_ratings = .... [TRUNCATED]

> hist(vector_ratings, main="Histogram of Ratings", xlab="Rating Value")
```

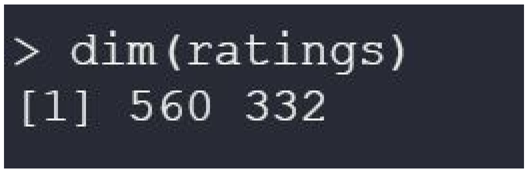


3. Data Preparation

   # Minimum threshold

   ratings = MovieLense[rowCounts(MovieLense) > 50, colCounts(MovieLense) > 100]

   dim(ratings)



```
> dim(ratings)
[1] 560 332
```

   # Normalizing the data

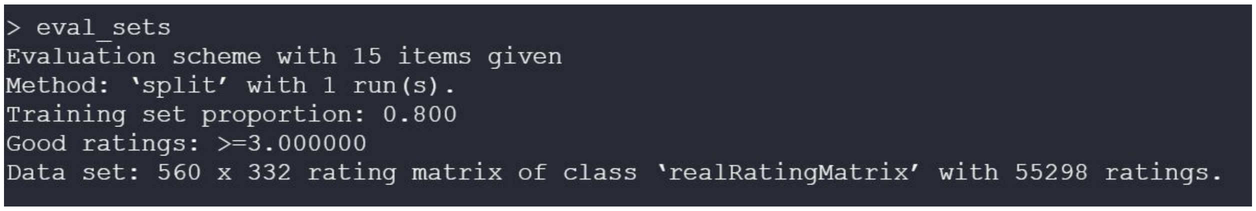
   ratings.n = normalize(ratings)

   ratings.n.vec = as.vector(ratings.n@data)

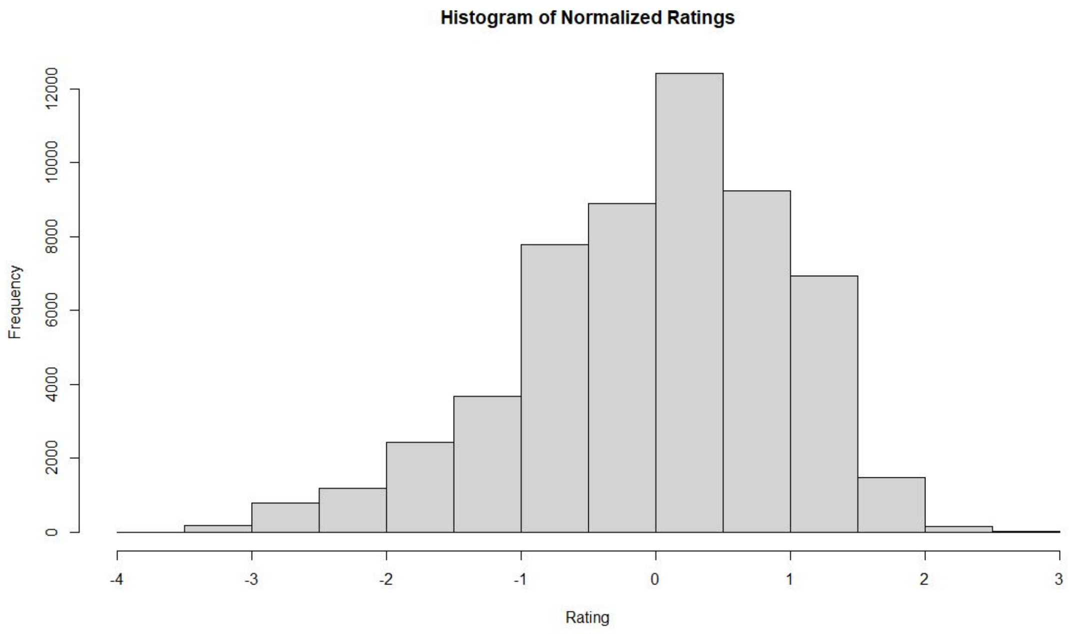
   ratings.n.vec = ratings.n.vec[ratings.n.vec != 0]

   hist(ratings.n.vec, main = "Histogram of Normalized Ratings", xlab = "Rating")

```
# Splitting data for test and train

percent_train = 0.8

#min(rowCounts(ratings.n))

items_to_keep = 15 # items to use for each user

rating_threshold = 3 # good rating implies >=3

n_eval = 1 # number of times to run eval

eval_sets = evaluationScheme(data = ratings, method = "split",

                train = percent_train, given = items_to_keep,

                goodRating = rating_threshold, k = n_eval)

eval_sets
```

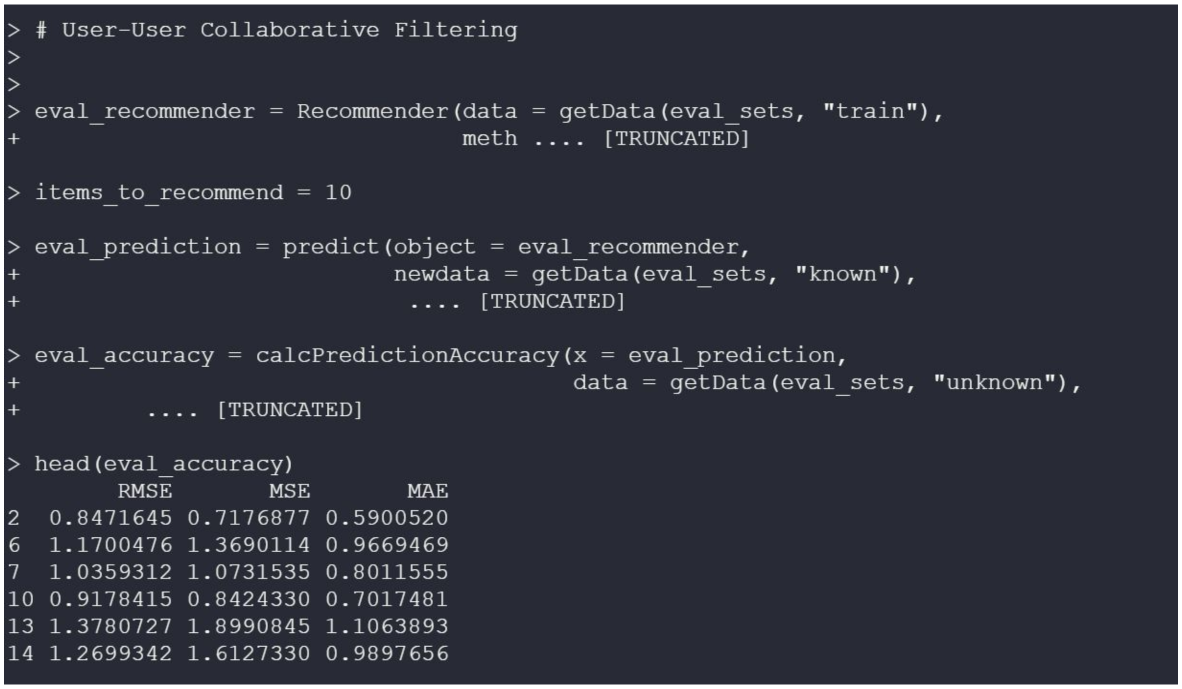


```
> eval_sets
Evaluation scheme with 15 items given
Method: ‘split’ with 1 run(s).
Training set proportion: 0.800
Good ratings: >=3.000000
Data set: 560 x 332 rating matrix of class ‘realRatingMatrix’ with 55298 ratings.
```



4. User-User Collaborative Filtering

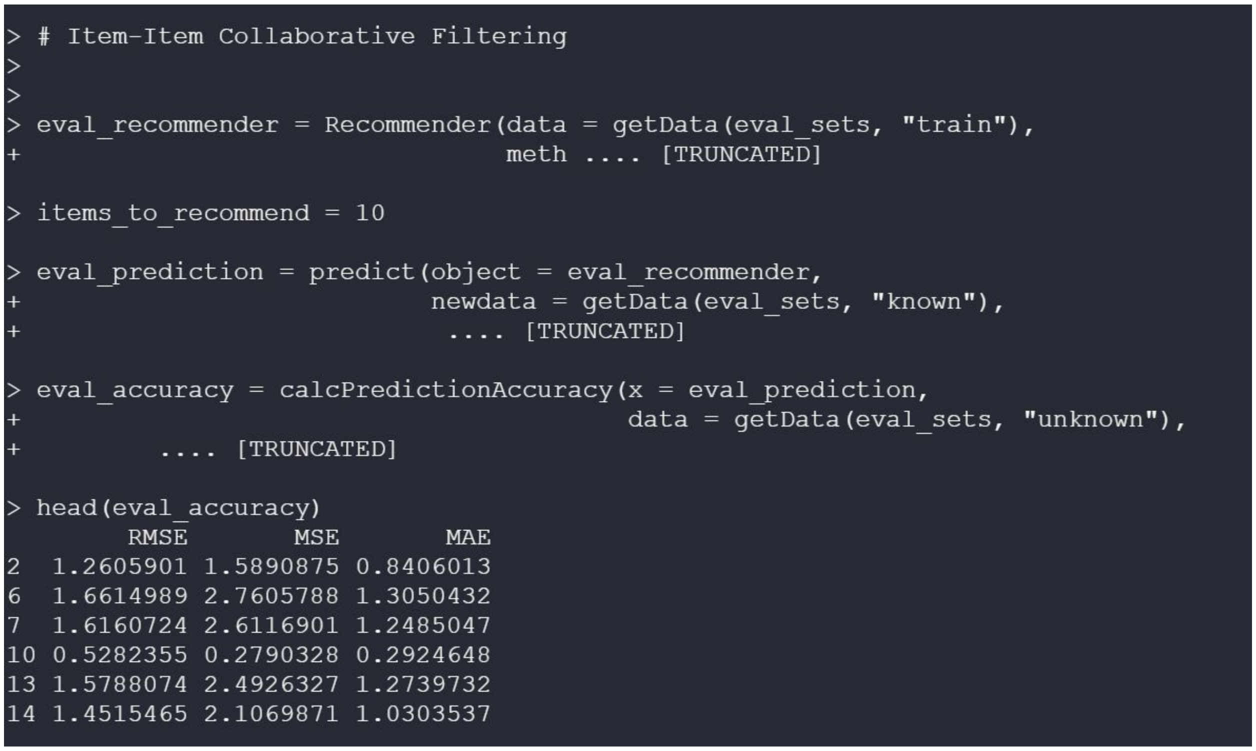
```
# User-User Collaborative Filtering

eval_recommender = Recommender(data = getData(eval_sets, "train"),
                               method = "UBCF", parameter = NULL)

items_to_recommend = 10

eval_prediction = predict(object = eval_recommender,
                          newdata = getData(eval_sets, "known"),
                          n = items_to_recommend,
                          type = "ratings")

eval_accuracy = calcPredictionAccuracy(x = eval_prediction,
                                       data = getData(eval_sets, "unknown"),
                                       byUser = TRUE)

head(eval_accuracy)
```



```
> # User-User Collaborative Filtering
>
>
> eval_recommender = Recommender(data = getData(eval_sets, "train"),
+                                meth .... [TRUNCATED]

> items_to_recommend = 10

> eval_prediction = predict(object = eval_recommender,
+                           newdata = getData(eval_sets, "known"),
+                            .... [TRUNCATED]

> eval_accuracy = calcPredictionAccuracy(x = eval_prediction,
+                                        data = getData(eval_sets, "unknown"),
+         .... [TRUNCATED]

> head(eval_accuracy)
        RMSE       MSE       MAE
2  0.8471645 0.7176877 0.5900520
6  1.1700476 1.3690114 0.9669469
7  1.0359312 1.0731535 0.8011555
10 0.9178415 0.8424330 0.7017481
13 1.3780727 1.8990845 1.1063893
14 1.2699342 1.6127330 0.9897656
```

5. Item-Item Collaborative Filtering

```
# Item-Item Collaborative Filtering

eval_recommender = Recommender(data = getData(eval_sets, "train"),

                               method = "IBCF", parameter = NULL)

items_to_recommend = 10

eval_prediction = predict(object = eval_recommender,

                          newdata = getData(eval_sets, "known"),

                          n = items_to_recommend,

                          type = "ratings")

eval_accuracy = calcPredictionAccuracy(x = eval_prediction,

                                       data = getData(eval_sets, "unknown"),

                                       byUser = TRUE)

head(eval_accuracy)
```



```
> # Item-Item Collaborative Filtering
>
>
> eval_recommender = Recommender(data = getData(eval_sets, "train"),
+                                 meth .... [TRUNCATED]

> items_to_recommend = 10

> eval_prediction = predict(object = eval_recommender,
+                           newdata = getData(eval_sets, "known"),
+                            .... [TRUNCATED]

> eval_accuracy = calcPredictionAccuracy(x = eval_prediction,
+                                        data = getData(eval_sets, "unknown"),
+         .... [TRUNCATED]

> head(eval_accuracy)
        RMSE       MSE       MAE
2  1.2605901 1.5890875 0.8406013
6  1.6614989 2.7605788 1.3050432
7  1.6160724 2.6116901 1.2485047
10 0.5282355 0.2790328 0.2924648
13 1.5788074 2.4926327 1.2739732
14 1.4515465 2.1069871 1.0303537
```

6. Evaluating Models using different Similarity Parameters

   # Evaluating Models using different Similarity Parameters

   models_to_evaluate = list(IBCF_cos = list(name = "IBCF", param = list(method = "cosine")),

   IBCF_cor = list(name = "IBCF", param = list(method = "pearson")),

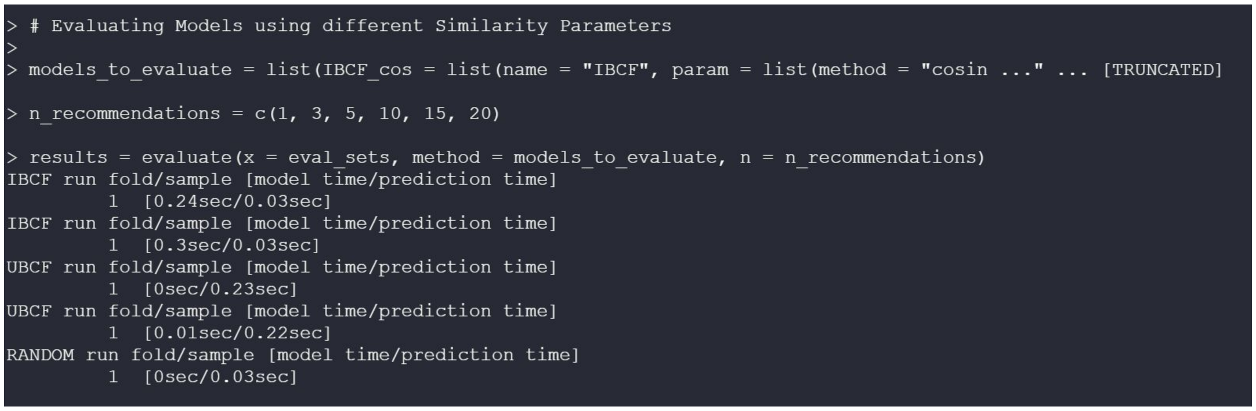
   UBCF_cos = list(name = "UBCF", param = list(method = "cosine")),

   UBCF_cor = list(name = "UBCF", param = list(method = "pearson")),

   random = list(name = "RANDOM", param = NULL))

   n_recommendations = c(1, 3, 5, 10, 15, 20)

   results = evaluate(x = eval_sets, method = models_to_evaluate, n = n_recommendations)


```
> # Evaluating Models using different Similarity Parameters
>
> models_to_evaluate = list(IBCF_cos = list(name = "IBCF", param = list(method = "cosin ..." ... [TRUNCATED]

> n_recommendations = c(1, 3, 5, 10, 15, 20)

> results = evaluate(x = eval_sets, method = models_to_evaluate, n = n_recommendations)
IBCF run fold/sample [model time/prediction time]
	 1  [0.24sec/0.03sec]
IBCF run fold/sample [model time/prediction time]
	 1  [0.3sec/0.03sec]
UBCF run fold/sample [model time/prediction time]
	 1  [0sec/0.23sec]
UBCF run fold/sample [model time/prediction time]
	 1  [0.01sec/0.22sec]
RANDOM run fold/sample [model time/prediction time]
	 1  [0sec/0.03sec]
```

7. Comparing the Collaborative Filtering Models

   # Comparing the Collaborative Filtering Models

   # Draw ROC curve

   plot(results, y = "ROC", annotate = 1, legend = "topleft")

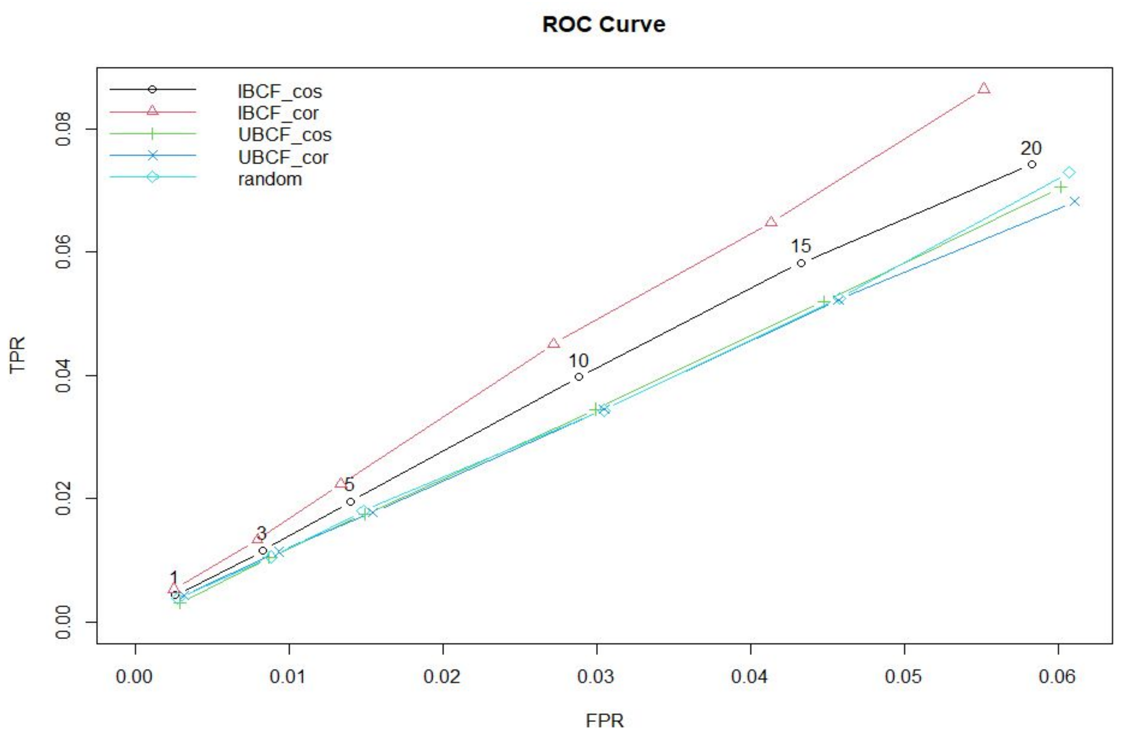
   title("ROC Curve")

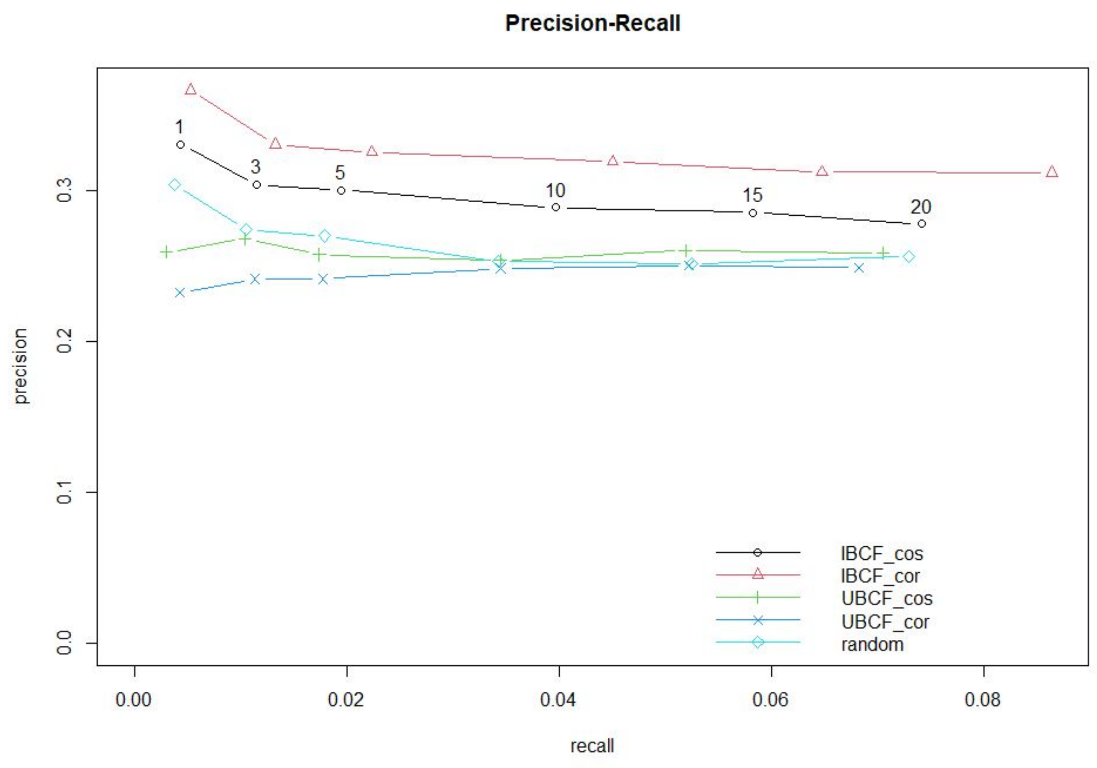
   # Draw precision / recall curve

   plot(results, y = "prec/rec", annotate = 1)

   title("Precision-Recall")

NAME: Hammad Ansari ROLL NO: 2018450002





8. Conclusions

We see that UBCF’s accuracy is higher than that of IBCF. UBCF using Pearson Correlation outperforms all other models. On the other hand, UBCF has a greater computational cost and requires more resources. There also exist hybrid systems that integrate both UBCF and IBCF approaches [6]. It is also worth noting that both UBCF and IBCF have limitations – for example when handling users who have made no purchases or items without a single purchase (the cold-start problem).