בס"ד

Big Data Hackathon Report

Group: A(L)IEN

Group members: Aviel Berkowitz, Itamar Cohen, Elchanan Bloom, Noam Simchon.

July 2022

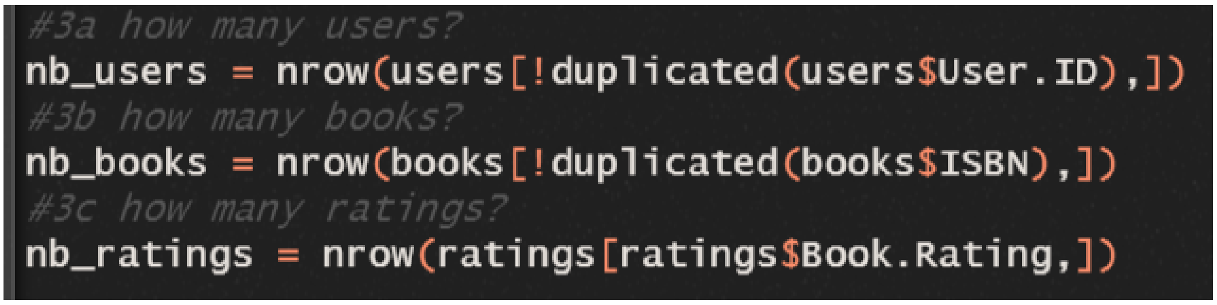
JCT, Jerusalem, Israel

GitHub Repository link

The data we received was stored in SQL tables and since R does not know how to handle SQL we extracted the data using python to mySQL format and then by using R's dbConnect library we converted the data to R tables.

After that, we have to do pre-processing, for example, by remove no relevant data like duplicates users with same id or books with same ISBN. For that, we use the command !duplicated() that returns strictly different rows according to one parameter. For users, we filter duplicates according to the User.ID column, while for books we filter according to ISBN column.

On this result, we use the function nrow() to returns lines of data frame

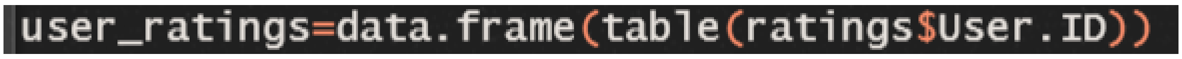


```
#3a how many users?
nb_users = nrow(users[!duplicated(users$User.ID),])
#3b how many books?
nb_books = nrow(books[!duplicated(books$ISBN),])
#3c how many ratings?
nb_ratings = nrow(ratings[ratings$Book.Rating,])
```

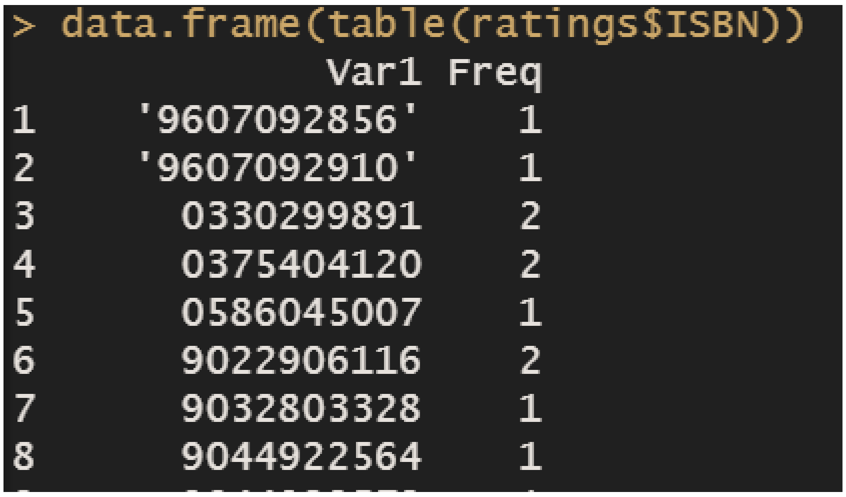
At this step, we have a clean data. Now we want to predict rate of book b for an user u in order to propose him this book. For that,we will implement the collaborative filtering algorithm. But now, we have lot of books which have not been rated by lot of users, that is meaning that when we will build the book-user matrix, we will receive matrix almost empty. So it missing data and when we predict ratings of 270 551 books for 278 858 users we will receive not plausible results. That it why, we want to keep only the most rated books and the most user-ratings.

We create a new table named user_ratings from the ratings table by sum occurrence of users in the table. To get this frequency, we use the table function on User.ID column of ratings table. That is the complete command :

בס"ד

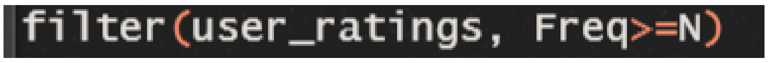


```
user_ratings=data.frame(table(ratings$User.ID))
```

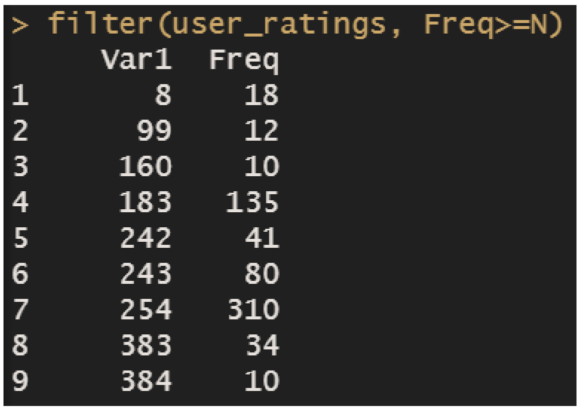


```
> data.frame(table(ratings$ISBN))
                  Var1 Freq
1     '9607092856'    1
2     '9607092910'    1
3       0330299891    2
4       0375404120    2
5       0586045007    1
6       9022906116    2
7       9032803328    1
8       9044922564    1
```

Now we filter the N most frequent, for instance I choose N = 10

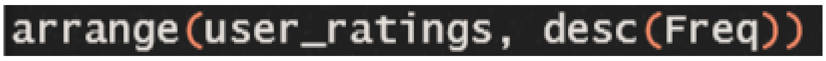


```
filter(user_ratings, Freq>=N)
```

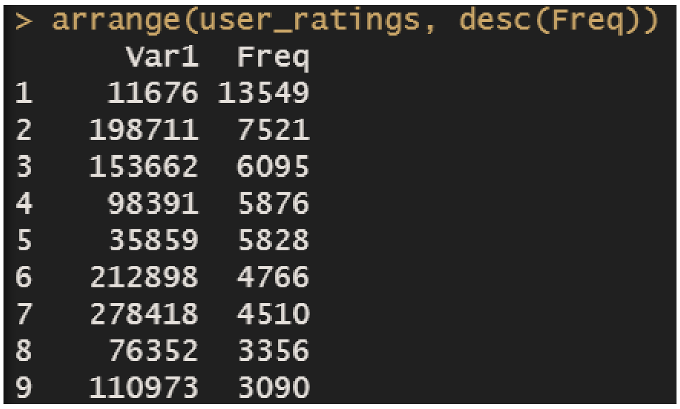


```
> filter(user_ratings, Freq>=N)
     Var1  Freq
1       8    18
2      99    12
3     160    10
4     183   135
5     242    41
6     243    80
7     254   310
8     383    34
9     384    10
```

And now we have to sort by the frequency column with arrange function



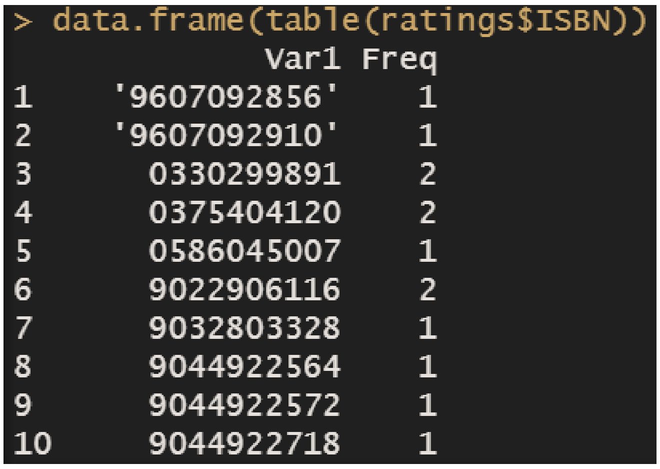
```
arrange(user_ratings, desc(Freq))
```



```
> arrange(user_ratings, desc(Freq))
       Var1  Freq
1     11676 13549
2    198711  7521
3    153662  6095
4     98391  5876
5     35859  5828
6    212898  4766
7    278418  4510
8     76352  3356
9    110973  3090
```

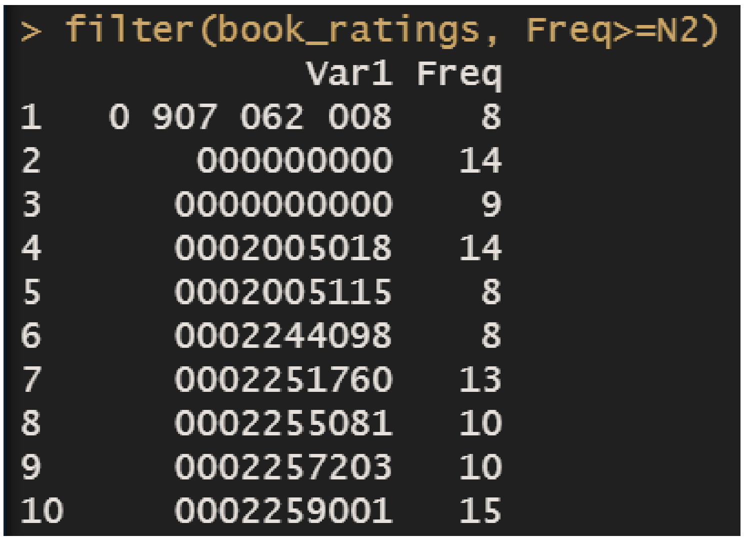
We can see in column Freq, that frequency values are sorted

For books, sone ones have only been rated very few times, so we keep only the top 10 rated-books and we will works only on these 10 books to implement algorithm of collaborative filtering. In order to find these top 10 rated books, we create a new table named book _ratings from the ratings table by sum occurrence of ISBN in the table. To get this frequency, we use the table function on ISBN column of ratings table. That is the complete command :



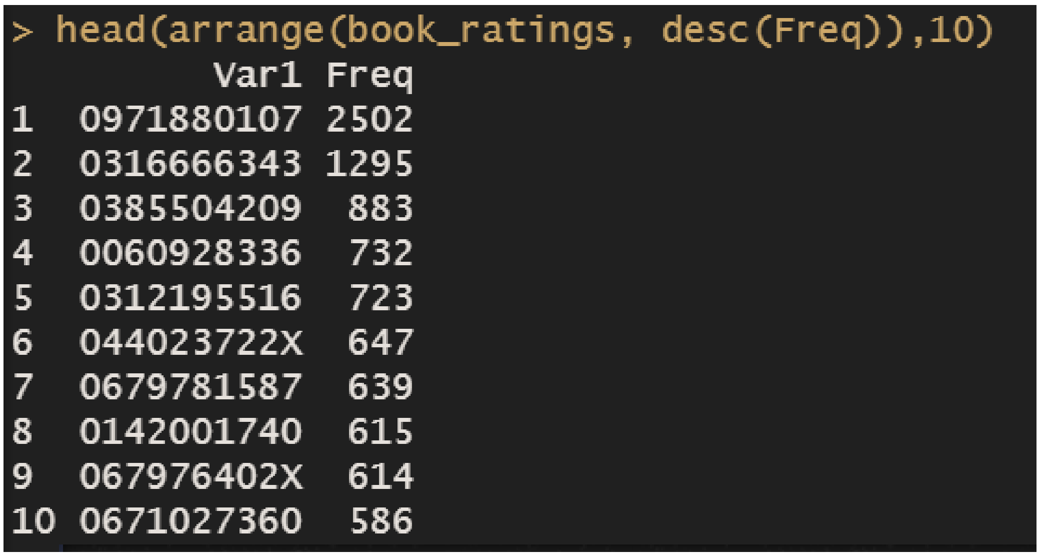
```
> data.frame(table(ratings$ISBN))
            Var1 Freq
1    '9607092856'    1
2    '9607092910'    1
3      0330299891    2
4      0375404120    2
5      0586045007    1
6      9022906116    2
7      9032803328    1
8      9044922564    1
9      9044922572    1
10     9044922718    1
```

Now we filter the N most frequent, for instance I choose N = 7



```
> filter(book_ratings, Freq>=N2)
           Var1 Freq
1   0 907 062 008    8
2       000000000   14
3      0000000000    9
4      0002005018   14
5      0002005115    8
6      0002244098    8
7      0002251760   13
8      0002255081   10
9      0002257203   10
10     0002259001   15
```

And now we have to sort all books from the book_ratings table by frequency



```
> head(arrange(book_ratings, desc(Freq)),10)
         Var1 Freq
1  0971880107 2502
2  0316666343 1295
3  0385504209  883
4  0060928336  732
5  0312195516  723
6  044023722X  647
7  0679781587  639
8  0142001740  615
9  067976402X  614
10 0671027360  586
```

Next, we deleted the books that existed in the Rating database but did not exist in the Book database with the following code:

```
ratings <- ratings[(ratings$ISBN %in% books$ISBN),]
```

Then we proceeded to the same sort on the IDs:

```
ratings <- ratings[(ratings$User.ID %in% users$User.ID),]
```

Due to the huge amount of data we had to deal with we decided to reduce the amount of data and leave only the data that contributes more to building the model. We thought users who rated less than 3 times and books rated less than 3 times were not helpful to us so we deleted them from the data. This should be done in iterations until there are no users / books left that have not been rated / rated at least 3 times. (Iterations are needed since let's say we have a user who rated 3 times books rated only by him so those books will drop from the data and then he is left with 0 ratings so we need to add iterations where this user will drop from the data)

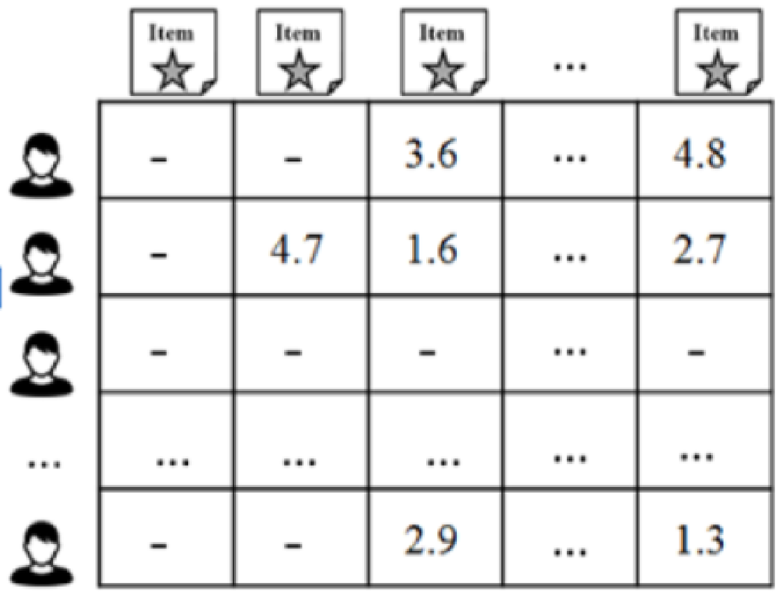
```
min.rating.user <- 8
min.rating.book <- 8


real.rating.matrix <- as(ratings, "realRatingMatrix")



while(min(rowCounts(real.rating.matrix)) < min.rating.user ||
min(colCounts(real.rating.matrix)) < min.rating.book){
  real.rating.matrix <-
real.rating.matrix[rowCounts(real.rating.matrix) >= min.rating.user,
colCounts(real.rating.matrix) >= min.rating.book]
```

Then we created a RealRating Matrix.

RealRatingMatrix implements a rating matrix with real valued ratings stored in sparse format defined in package Matrix. Sparse matrices in Matrix typically do not store 0s explicitly, however for realRatingMatrix we use these sparse matrices such that instead of 0s, NAs are not explicitly stored.

Why a Sparse Matrix?

In fact, we have databases that contain the ISBNs of the books, as well as the users who have rated these books. However, we do not have a table that directly matches users with the items and the score given, like that :



	Item	Item	Item	...	Item
	-	-	3.6	...	4.8
	-	4.7	1.6	...	2.7
	-	-	-	...	-
...	...	...	...	...	...
	-	-	2.9	...	1.3

To do this we create a matrix which applies this correspondence. However, we are faced with a problem: there are a lot of empty elements in this matrix because not all the books have been rated by the users. The point is, the array is far too large to process because it keeps even empty elements in memory. To solve this problem we must create a SPARSE MATRIX whose specificity is not to use memory for NAN type elements. Rating Matrix use sparse matrix system.

After that we separate our data using the K-fold validation system.

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

In our case we apply a k = 5 and split the data in train 0.8 and validation 0.2.

```
sets <- evaluationScheme(data = real.rating.matrix, method =
"split",
                         train = 0.8, given = min.rating.user,
                         goodRating = 5, k = 5)
```

After this step:

We created two recommenders (user-based and item-based collaborative filtering) using the training data:

UBCF:

```
UB_recommender <- Recommender(data = getData(sets, "train"),
                              method = "UBCF", parameter = NULL)


UB_prediction <- my.predict(UB_recommender@model,
                            newdata = getData(sets, "known"),
                            n = 10,
                            type = "ratings")
```

In the UB_prediction we choise the n = 10 user with more activity.

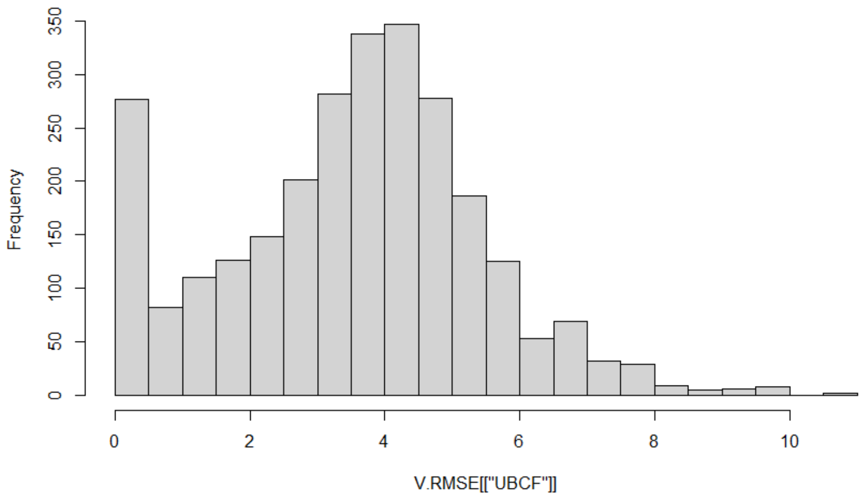
IBCF:

For IBCF we had to take less data, so we took users/books rated at least 8 times.

```
IB_recommender <- Recommender(data = getData(sets, "train"),
                              method = "IBCF", parameter = NULL)

IB_prediction <- predict(IB_recommender,
                           newdata = getData(sets, "known"),
                           n = 10,
                           type = "ratings")
```

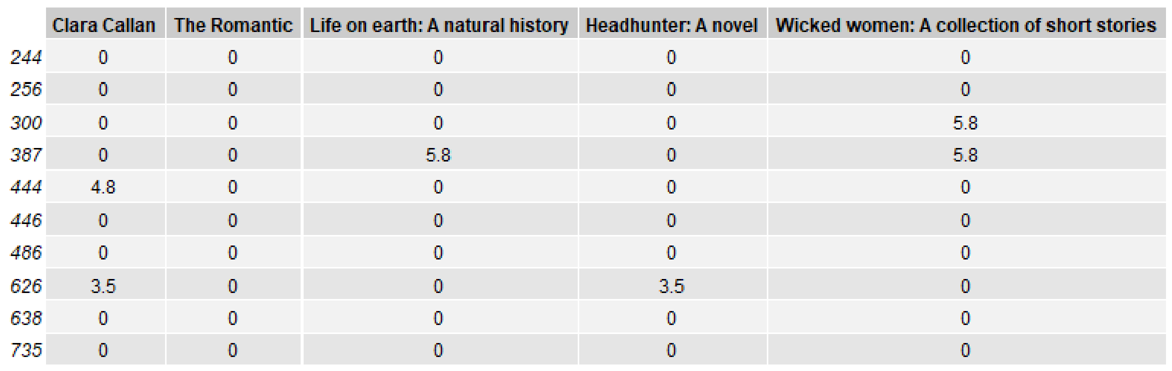
<u>UBCF Results:</u>

RMSE histogram



UBCF Total RMSE: **3.8122**

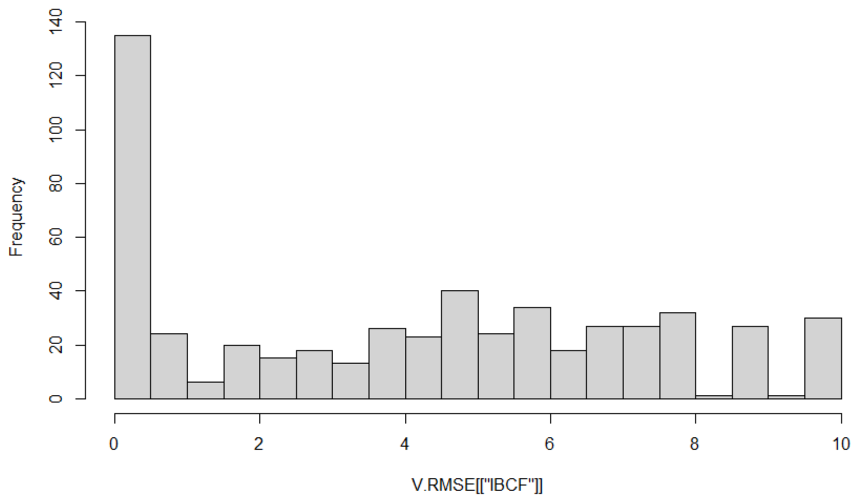
UBCF Table:



	Clara Callan	The Romantic	Life on earth: A natural history	Headhunter: A novel	Wicked women: A collection of short stories
244	0	0	0	0	0
256	0	0	0	0	0
300	0	0	0	0	5.8
387	0	0	5.8	0	5.8
444	4.8	0	0	0	0
446	0	0	0	0	0
486	0	0	0	0	0
626	3.5	0	0	3.5	0
638	0	0	0	0	0
735	0	0	0	0	0

IBCF Results:

RMSE histogram



IBCF Total RMSE: **4.9664**

IBCF Table:

	Clara Callan	The Forgetting Room: A Fiction (Byzantium Book)	Spadework	The Piano Man's Daughter	Angelas Ashes
254	0	0	0	0	0
388	0	0	0	0	7
408	0	0	0	0	0
487	0	0	0	0	0
505	0	0	0	0	0
619	0	0	0	0	0
638	0	0	0	0	0
735	0	0	0	0	0
876	0	0	0	0	0
1075	5	0	0	0	0