Movie Recommender System

- A Machine Learning Project using Python

Summer Internship Major Project

By Shreeyansh Das| SIC: 20BCSD50

Branch: Computer Sc. and Engnn.

Year: 20-2021 | Regn. No. - 2001209182

This project was a part of the Yearly Summer Internship which was undertaken during the month of September, 2021. Given the choice with 3 project ideas, I decided to choose the one which intrigued me quite well – to build a movie recommender system. We get to observe basically any recommender system in our day-to-day life. Be it, *YouTube* videos, *Netflix* series, *Instagram* Sponsored posts, *Facebook* Ads, etc. All of these recommendations are made possible by the implementation of recommender systems.

Recommender systems encompass a class of techniques and algorithms that can suggest "relevant" items to users. They predict future behaviour based on past data through a multitude of techniques including matrix factorization.

These are few to mention but in the real world we get to see a lot more applications. The document consists of following sections summarising various aspects of the project in an overview –

1. **Problem Formulation**
2. **Algorithms Followed**
3. **Dataset Used**
4. **Results and Discussion**
5. **References**

The following document is a concise summary of my Recommender System which was trained on almost 1, 000,000 (MovieLens | GroupLens) examples of *MovieLens* Raw Data and gives a recommendation of movie to a user based on his interests.

1. Problem Formulation

Example: Predict movie ratings

Suppose we have a set of movies and some people who rate then on a scale of 0 to 5. We have features x_1 *and* x_2 of each movies that represent some sort of linear function about the amount of action-ness and romantic-ness in each movie on a scale of 0 to 1. The ratings that they have provided are as follows:

Movie	**Alice**	**Bob**	**Carol**	**Dave**	x_1	x_2
Love at last	5	5	0	0	0.9	0
Romance Forever	5	?	?	0	1.0	0.01
Cute puppies	?	4	0	?	0.99	0
Fast and Furious	0	0	5	4	0.1	1.0
Kill/Bill	0	0	5	?	0	0.9

Next we have some metrics say,

n_u: $\#users$ $(4\ in\ this\ case)$

n_m: $\#movies$ $(5\ in\ this\ case)$

$r^{(i,j)} = 1$, if the user 'i' has rated movie 'j'

$y^{(i,j)} = rating$, that the user 'i' has provided to movie 'j'

$\theta^{(j)} =$ Parameter vector of 'j'

$\boldsymbol{x^{(i)}} =$ **Feature vector for movie 'i'**

∴ User j's prediction on 'i' will be $\left(\theta^{(j)}\right)^T\left(x^{(i)}\right)$ So here is the problem formulation: *given this data that has given these* $r^{(i,j)}$ *and the* $y^{(i,j)}$*, we have to look through the data and look at all the movie ratings that are missing and to try to predict what these values of the question marks should be.*

2. Algorithms Followed

Movie recommender systems are basically of 2 types – Content Based and Collaborative filtering.

- ### Content Based Engines

  The Content-Based Recommender relies on the similarity of the items being recommended. The basic idea is that if you like an item, then you will also like a "similar" item. Based on that data that a user provides, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate. We have certain features, say x_1 *and* x_2 of each movies that represent some sort of linear function about the amount of action-ness and romantic-ness in each movie on a scale of 0 to 1.

- ### Collaborative Filtering Engine

  An approach to building a recommender system is called collaborative filtering. The algorithm that we're talking about has a very interesting property that it does what is called feature learning and by that we mean that this will be an algorithm that can start to learn for itself what features to use. More specifically, it is based on the similarity in preferences, tastes and choices of two users. It analyses how similar the tastes of one user is to another and makes recommendations on the basis of that.

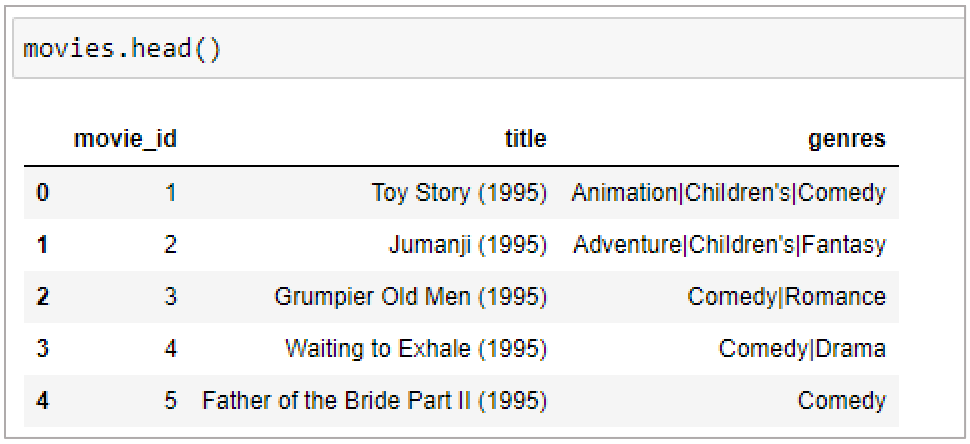
- ### Which one to choose?

  Content based engines generally works well when it's easy to determine the context/properties of each item. A content based recommender works with data that the user provides, here explicitly movie ratings for the MovieLens dataset. As we can imagine it can be very difficult and time consuming and expensive to actually try to get someone to, watch each movie and tell us how romantic each movie and how action packed is each movie, and often we'll want even more features than just these two. So, to avoid such time consuming and computationally expensive process we will choose Collaborative filtering approach. In this method, since the feature vector of each movie is unknown, the algorithm will determine them on its own.

3. Dataset Used

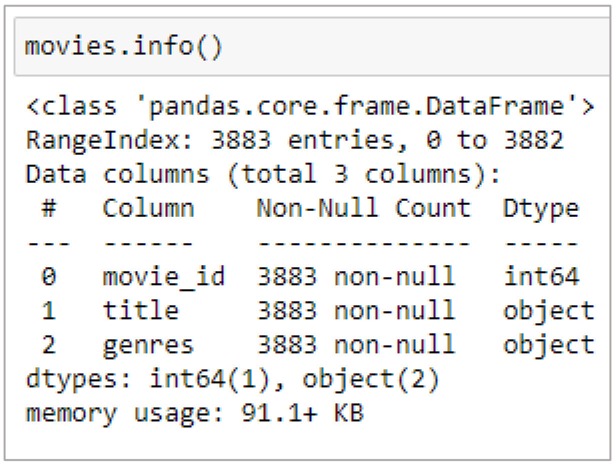
One of the most common datasets that is available on the internet for building a Recommender System is the MovieLens Dataset. This version of the dataset that I'm working with contains **1,000,209 anonymous ratings** of approximately **3,900 movies** made by **6,040 MovieLens users** who joined MovieLens in 2000. This 1M version was released on February 2003. Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided. The original data are contained in three files, “movies.dat”, “ratings.dat” and “users.dat”, which have been converted to csv’s for easy use.

- The “**movies.csv”** contains 3883 entries (indexed from 0) with 3 features, namely, ‘movie_id’, ‘title’ and ‘genre’. Titles are identical to titles provided by the IMDB (including year of release). Genres are pipe-separated (‘|’) and are selected from the following genres:
  - Action
  - Adventure
  - Animation
  - Children's
  - Comedy
  - Crime
  - Documentary
  - War
  - Western
  - Drama
  - Fantasy
  - Film-Noir
  - Horror
  - Musical
  - Mystery
  - Romance
  - Sci-Fi
  - Thriller



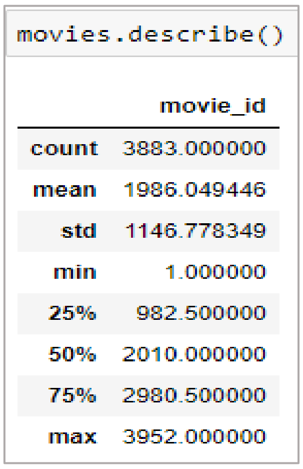
```
movies.head()
```

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy



```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   movie_id   3883 non-null   int64 
 1   title      3883 non-null   object
 2   genres     3883 non-null   object
dtypes: int64(1), object(2)
memory usage: 91.1+ KB
```



```
movies.describe()
```

	movie_id
count	3883.000000
mean	1986.049446
std	1146.778349
min	1.000000
25%	982.500000
50%	2010.000000
75%	2980.500000
max	3952.000000

- The “**users.csv**” file contains 6040 entries (indexed from 0 to 6039) with 5 features namely ‘user_id’, ‘gender’, ‘zipcode’, ‘age_desc’ and ‘occ_desc’.
  - Gender is denoted by an "M" for male and "F" for female
  - Age is chosen from the following ranges:

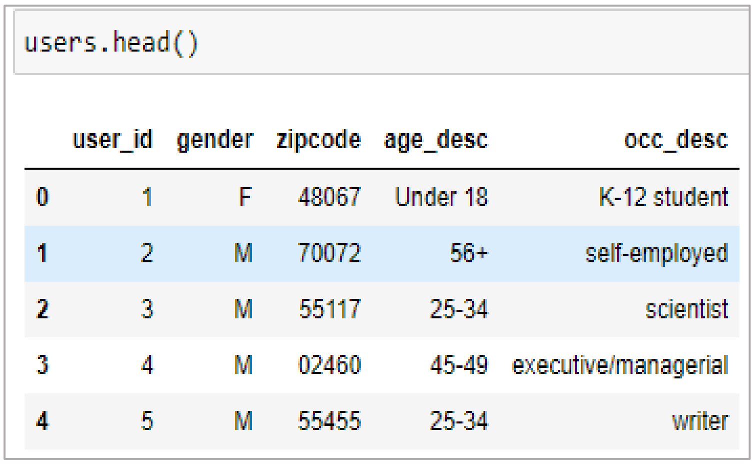
    1: "Under 18", 18: "18-24",

    25: "25-34", 35: "35-44", 45: "45-49",

    50: "50-55", 56: "56+"

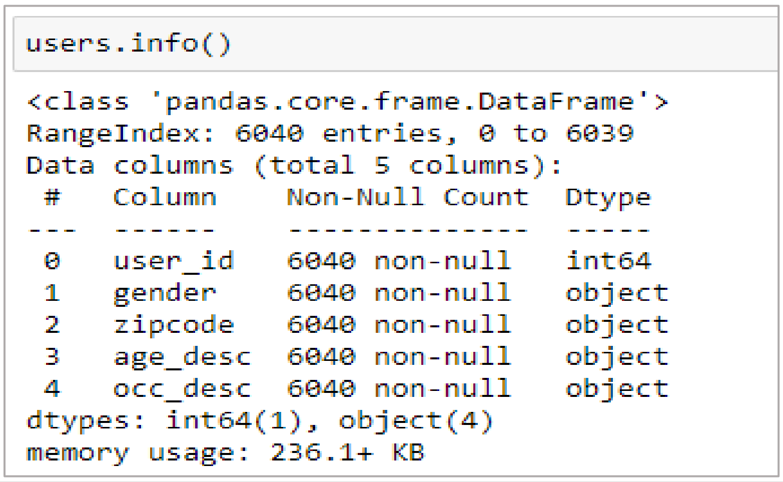
  - Occupation is chosen from the following choices:

0: "other" or not specified	11: "lawyer"
1: "academic/educator"	12: "programmer"
2: "artist"	13: "retired"
3: "clerical/admin"	14: "sales/marketing"
4: "college/grad student"	15: "scientist"
5: "customer service"	16: "self-employed"
6: "doctor/health care"	17: "technician/engineer"
7: "executive/managerial"	18: "tradesman/craftsman"
8: "farmer"	19: "unemployed"
9: "homemaker"	20: "writer"
10: "K-12 student"	



```
users.head()
```

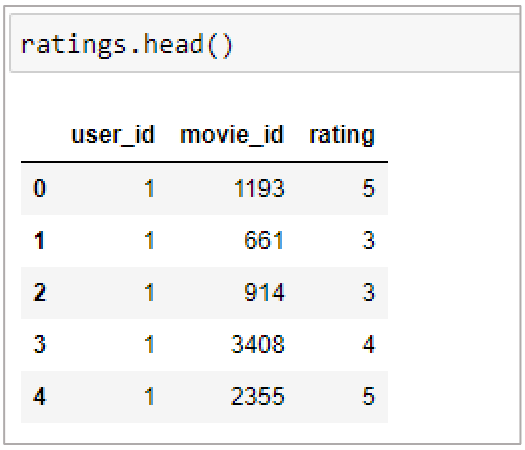
	user_id	gender	zipcode	age_desc	occ_desc
0	1	F	48067	Under 18	K-12 student
1	2	M	70072	56+	self-employed
2	3	M	55117	25-34	scientist
3	4	M	02460	45-49	executive/managerial
4	5	M	55455	25-34	writer



```
users.info()
```

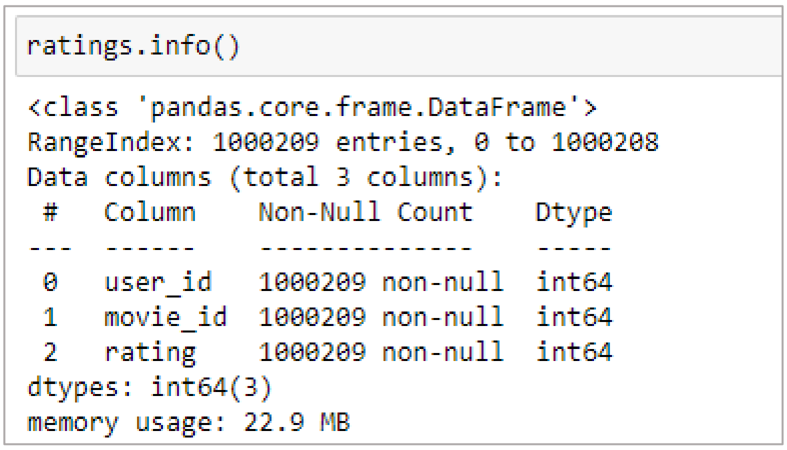
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   user_id   6040 non-null   int64 
 1   gender    6040 non-null   object
 2   zipcode   6040 non-null   object
 3   age_desc  6040 non-null   object
 4   occ_desc  6040 non-null   object
dtypes: int64(1), object(4)
memory usage: 236.1+ KB
```

- The “**ratings.csv**” file contains 1000209 entries with 3 columns namely ‘user_id’, ‘movie_id’ and ‘rating’. Hence, there are 1M ratings for different user and movie combinations.
  - User IDs range between 1 and 6040
  - Movie IDs range between 1 and 3952
  - Ratings are made on a 5-star scale (whole-star ratings only)
  - Timestamp is represented in seconds
  - Each user has at least 20 ratings



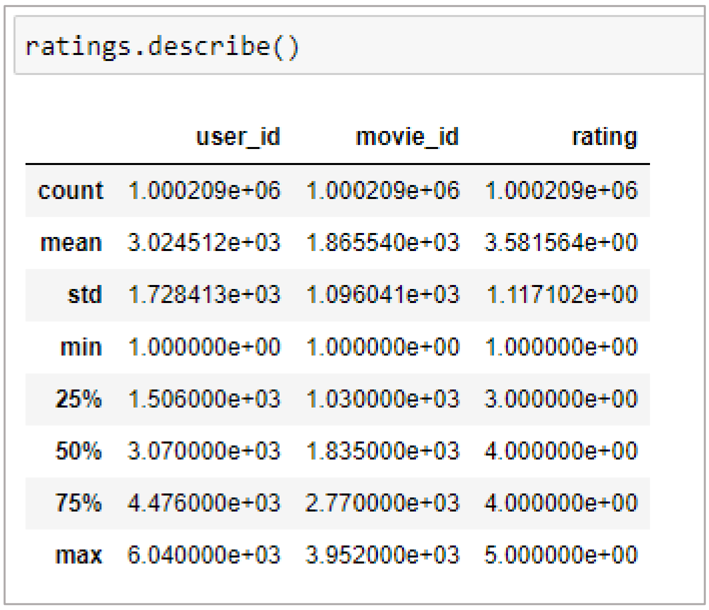
```
ratings.head()
```

	user_id	movie_id	rating
0	1	1193	5
1	1	661	3
2	1	914	3
3	1	3408	4
4	1	2355	5



```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 3 columns):
 #   Column    Non-Null Count    Dtype
---  ------    --------------    -----
 0   user_id   1000209 non-null  int64
 1   movie_id  1000209 non-null  int64
 2   rating    1000209 non-null  int64
dtypes: int64(3)
memory usage: 22.9 MB
```



```
ratings.describe()
```

	user_id	movie_id	rating
count	1.000209e+06	1.000209e+06	1.000209e+06
mean	3.024512e+03	1.865540e+03	3.581564e+00
std	1.728413e+03	1.096041e+03	1.117102e+00
min	1.000000e+00	1.000000e+00	1.000000e+00
25%	1.506000e+03	1.030000e+03	3.000000e+00
50%	3.070000e+03	1.835000e+03	4.000000e+00
75%	4.476000e+03	2.770000e+03	4.000000e+00
max	6.040000e+03	3.952000e+03	5.000000e+00

4. Results and Discussions

There are 2 main types of Collaborative Filtering algorithms:

4.1 User Based CF –

User based collaborative filtering (a.k.a. user-user collaborative filtering) is based on the interests of similar users. It recommends those items to a user that another user with a similar taste has rated. The implementation goes as follows –

Step 1: Find the average rating given by target user

$$\overline{r_i} = \frac{\sum_p r_{ip}}{\sum p}$$

Average rating of user ‘i’ over all items ‘p’ excluding non-rated item

Step 2: Find Similarity Factor of target user with all users

The formula to calculate similarity between two users ‘a’ and ‘b’ for item ‘p’ is given by the Pearson correlation coefficient–

$$\boxed{Sim(a,b) = \frac{(r_{ap} - \overline{r_a}) \times (r_{bp} - \overline{r_b})}{\sqrt{\sum(r_{ap} - \overline{r_a})^2}\sqrt{\sum(r_{bp} - \overline{r_b})^2}}}$$

, where r_{up} = rating of user ‘u’ for item ‘p’

$\overline{r_a}$ = Average rating given by user ‘a’

We do this computation for the targeted user with all other users.

Step 3: Predict the non-rated movie for the target user

The target user might be very similar to some users and may not be much similar to others. Hence, the ratings given to a particular item by the more similar users should be given more weightage than those given by less similar users and so on. This problem can be solved by using a weighted average approach. In this approach, we multiply the rating of each user with a *similarity factor* calculated using the above mention formula. Thus the missing rating is of user ‘u’ for item ‘p’ is calculated as,

$$\boxed{r_{up} = \overline{r_u} + \frac{\sum_{i \in users} sim(u,i) \times r_{ip}}{\sum_{i \in users} |sim(u,i)|}}$$

4.2 Item Based CF –

Item based collaborative filtering (a.k.a. item-item collaborative filtering) is based on similar items rather than users. Here we try to find movie's look-alike based on a user's taste. The algorithm matches each of the user's rated items to similar items, then combines those similar items into a recommendation list. The implementation goes as follows –

Step 1: Find item-to-item similarity

There are many methods to find similarity between 2 items like cosine similarity, which is the cosine angle between the 2 items' vectors. There is also spearman similarity that measures the spearman correlation between the items. The cosine similarity is given by,

$$\boxed{Sim(\overrightarrow{A} \cdot \vec{B}) = \frac{\overrightarrow{A} \cdot \vec{B}}{|\vec{A}||\vec{B}|} = \cos(\theta_{AB})}$$

Step 2: Predict recommendation

Here, items (rated already) most similar to the item that hasn't been rated are found and predictions are generated using these similar items. The rating is computed for a particular item using weighted sum of the ratings of the other similar products using the formula –

$$\boxed{Rating(U, I_i) = \frac{\sum_j rating(U, I_j) \times S_{ij}}{\sum_j S_{ij}}}$$

Where I_i = Movie that hasn't been rated yet

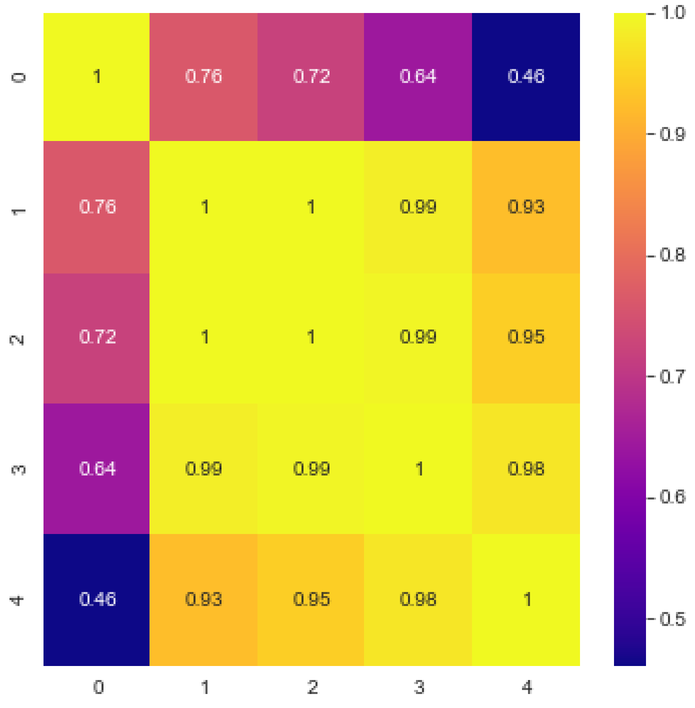
I_j = Movie that has been rated

S_{ij} = Similarity between movie 'I' and movie 'j'

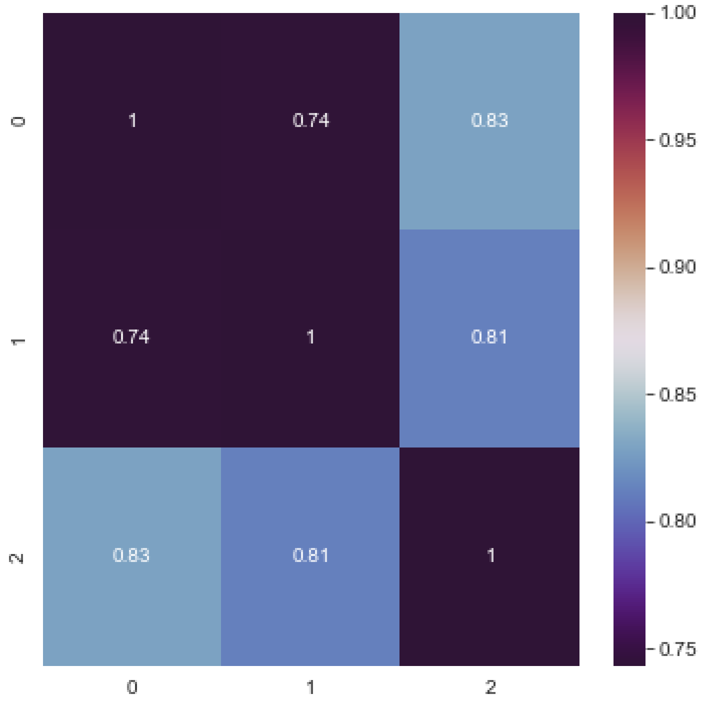
Henceforth, it is easily concluded that Item based Collaborative Filtering is easier to implement than User based Collaborative filtering. The computation costs are also low which is clearly visible. In either scenario, we build a *similarity matrix*. For user-user collaborative filtering, the **user-similarity matrix** will consist of some distance metrics that measure the similarity between any two pairs of users. Likewise, the **item-similarity matrix** will measure the similarity between any two pairs of items.

Heatmaps

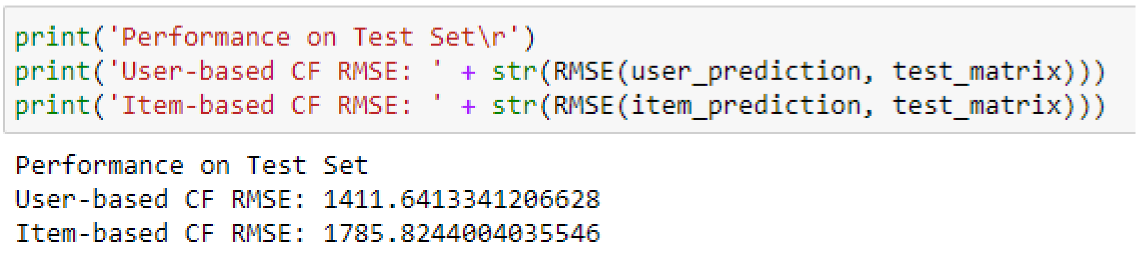
User-Based Pearson Similarity



Item-Based Cosine Similarity

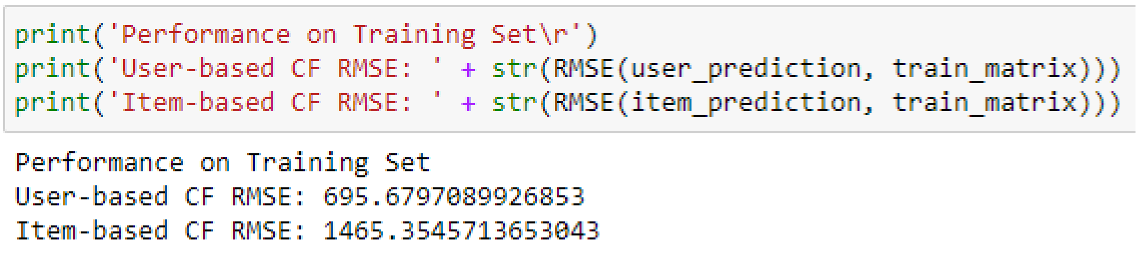


Evaluation Metrics



```
print('Performance on Test Set\r')
print('User-based CF RMSE: ' + str(RMSE(user_prediction, test_matrix)))
print('Item-based CF RMSE: ' + str(RMSE(item_prediction, test_matrix)))
```

```
Performance on Test Set
User-based CF RMSE: 1411.6413341206628
Item-based CF RMSE: 1785.8244004035546
```



```
print('Performance on Training Set\r')
print('User-based CF RMSE: ' + str(RMSE(user_prediction, train_matrix)))
print('Item-based CF RMSE: ' + str(RMSE(item_prediction, train_matrix)))
```

```
Performance on Training Set
User-based CF RMSE: 695.6797089926853
Item-based CF RMSE: 1465.3545713653043
```

$$RMSE = \sqrt{\frac{1}{N}\sum(x_i - \hat{x}_i)^2}$$

RMSE of training of model is a metric which measure how much the signal and the noise is explained by the model. The model surely does way better on training set when compared to test set. Model seems to have over-fit the training data. Overall, Memory-based Collaborative Filtering is easy to implement and produce reasonable prediction quality. However, there are some drawback of this approach:

- It doesn't address the well-known cold-start problem, i.e., when new user or new item enters the system.
- It can't deal with sparse data, meaning it's hard to find users that have rated the same items.
- It suffers when new users or items that don't have any ratings enter the system.
- It tends to recommend popular items.

The term ***collaborative filtering*** refers to the observation that when we run this algorithm with a large set of users, what all of these users are effectively doing are sort of collaboratively--or collaborating to get better movie ratings for everyone because with every user rating some subset with the movies, every user is helping the algorithm a little bit to learn better features, and then by helping-- by rating a few movies myself, I will be helping. The system learns better features and then these features can be used by the system to make better movie predictions for everyone else. And so there is a sense of collaboration where every user is helping the system learn better features for the common good. This is what collaborative filtering essentially means.

5. References

- GeeksforGeeks – Machine Learning - (Machine Learning - GeeksforGeeks)
- Machine Learning by Andrew Ng via Stanford Online – (Machine Learning | Coursera)
- GitHub Repository (GitHub - khanhnamle1994/movielens: 4 different recommendation engines for the MovieLens dataset.)
- Wikipedia.org