2020

COLLABORATIVE FILTERING ON NETFLIX DATA

PROJECT REPORT

SUNDA

Contents

1. Introduction & Motivation

In the present world, before investing on any product, users are looking for the reputation or rating of the product in the internet. Product companies are using this pattern to their advantage by giving recommendations to users when they shop for a product int heir platform. In a way this recommendation systems are useful to users also. This method will give users the list of products or items of their interest based on their past history.

So, users can stop hunting for new items to purchase and just rely on the recommendation systems. The loophole in this is that the user should have ample amount of past history for the recommendation system to work accurate. Entertainment media like Netflix, YouTube, Amazon Prime are the biggest platform for the recommendation systems.

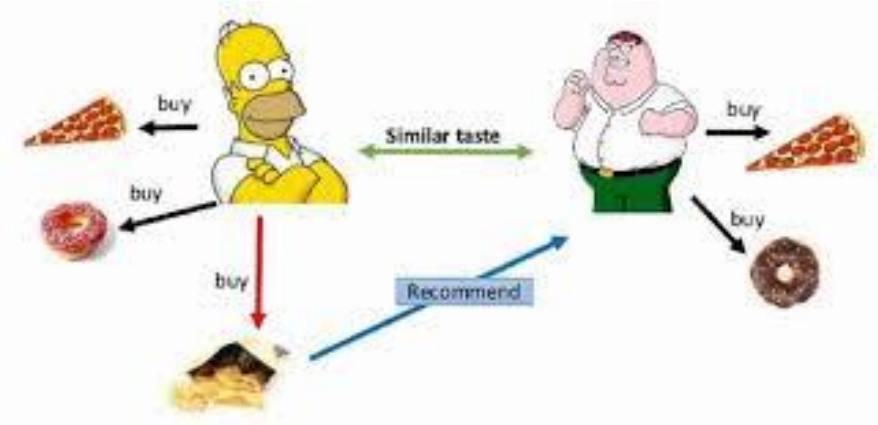

Figure 1: Recommendation Example

As an example, let us look at the task of movie recommendation. Suppose we have 1,000,000 users, and a list of the movies each user has watched (from a catalog of 500,000 movies). Our goal is to recommend movies to users. To solve this problem some method is needed to determine which movies are similar to each other. Here comes the concept of recommendation systems. The objective of this project is to build a recommendation system technique called collaborative filtering on Netflix data issued for the NETFLIX Prize.

2. Collaborative Filtering

Before going further, there is a need to understand collaborative filtering and how it works (methodology) and what are the end results of this algorithm. To achieve this, I approached Wikipedia, a free online encyclopedia. I read through the results and the useful required details to know about collaborative filtering are stated below:

2.1. Overview

Collaborative filtering (CF) is a technique used by recommender systems. Collaborative filtering has two senses, a narrow one and a more general one. In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. These predictions are specific to the user, but use information gleaned from many users.

In the more general sense, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc.

2.2. Methodology

Collaborative filtering algorithms often require (1) users' active participation, (2) an easy way to represent users' interests, and (3) algorithms that are able to match people with similar interests. Typically, the workflow of a collaborative filtering system is:

- A user expresses his or her preferences by rating items (e.g. books, movies or CDs) of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
- The system matches this user's ratings against other users' and finds the people with most "similar" tastes.
- With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item)

A key problem of collaborative filtering is how to combine and weight the preferences of user neighbors. Sometimes, users can immediately rate the recommended items. As a result, the system gains an increasingly accurate representation of user preferences over time.

3. System Requirements

For successful execution of this project, it should be done in Amazon Web Services (AWS) EMR (Elastic Map Reduce) clustering service. The chosen cluster configuration is:

EMR Cluster with 3 nodes (1 Master node & 2 Slave nodes). The cluster type is m4.xlarge with a EBS storage of .

4. Dataset Details

The dataset used in this project is NETFLIX data issued for the NETFLIX Prize. The data is provided in two separate files for training and test datasets. The details of these datasets are as follows:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3255352 entries, 0 to 3255351
Data columns (total 3 columns):
 #   Column   Dtype
---  ------   -----
 0   MovieID  int64
 1   UserID   int64
 2   Rating   float64
dtypes: float64(1), int64(2)
memory usage: 74.5 MB
```

Figure 2: Training Dataset details

From the above result, we can state that the data has about 3255352 samples and the features of the dataset are Movie ID, User ID & Rating of the movie given by the user. This dataset does not have any null values in it. Now, let's have a look at test dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100478 entries, 0 to 100477
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   MovieID  100478 non-null  int64
 1   UserID   100478 non-null  int64
 2   Rating   100478 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 2.3 MB
```

Figure 3: Test Dataset details

From the above result, we can state that the test data has about 100478 samples and the features of the dataset are Movie ID, User ID & Rating of the movie given by the user. This dataset also does not have any null values in it.

Apart from these two data files, the dataset also includes one more file of Movie details. The details of the file are as shown below:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17770 entries, 0 to 17769
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   MovieID  17770 non-null  int64
 1   Year     17763 non-null  float64
 2   Name     17770 non-null  object
dtypes: float64(1), int64(1), object(1)
memory usage: 416.6+ KB
```

Figure 4: Movie Dataset details

From the above result, we can state that the movie data has about 17770 samples and the features of the dataset are Movie ID, Year of the movie release& the Name of the movie. This dataset also does not have any null values in it.

To see the distribution of data in the respective dataset, I have used matplotlib & pandas.plotting packages. By using these packages, the distribution of data of both training set & testing set are shown below in two bar & scatter plots:

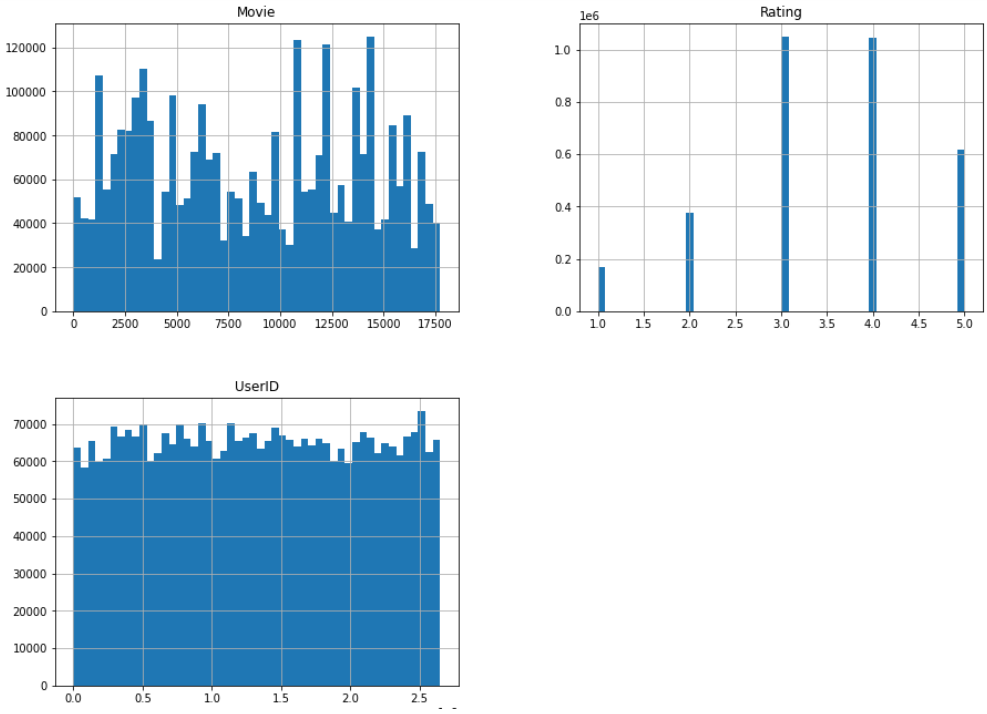

Figure 5: Train Data Distribution – Bar Plot

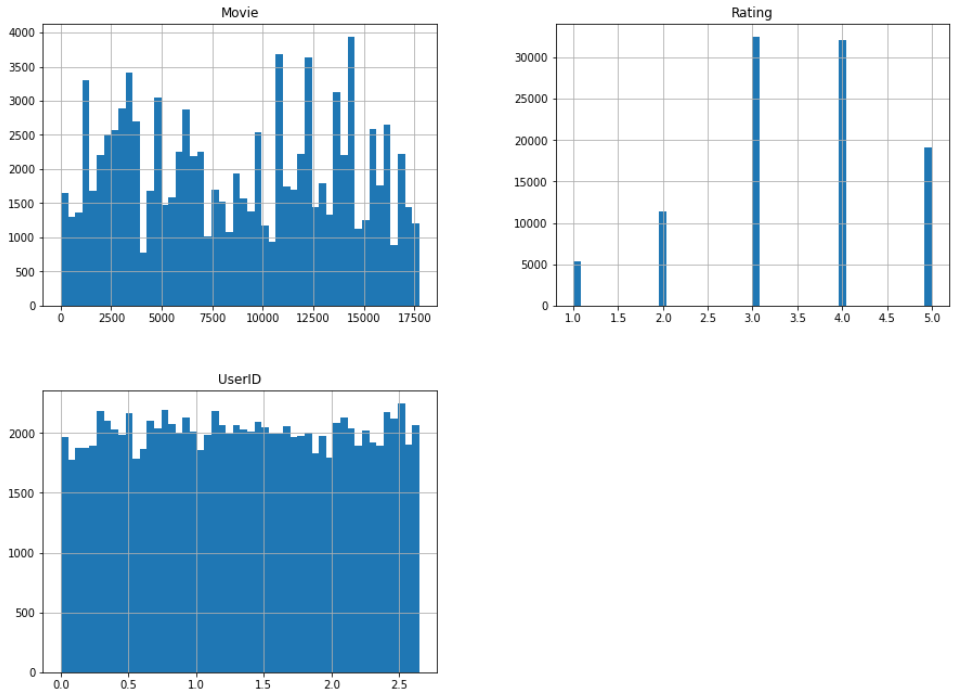

Figure 6: Test Data Distribution – Bar Plot

The distribution of each feature with itself & other features can be plotted by using scatter_matrix plot. And the respective scatter plots of train & test sets are as follows:

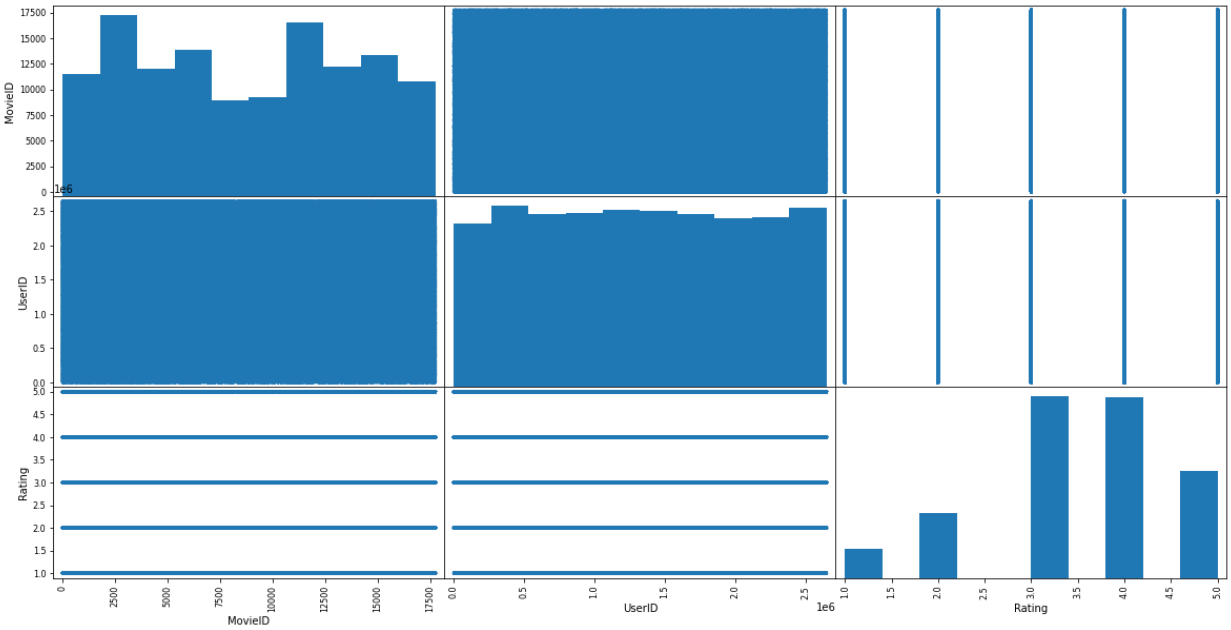

Figure 7: Train set Data Distribution – Scatter Matrix

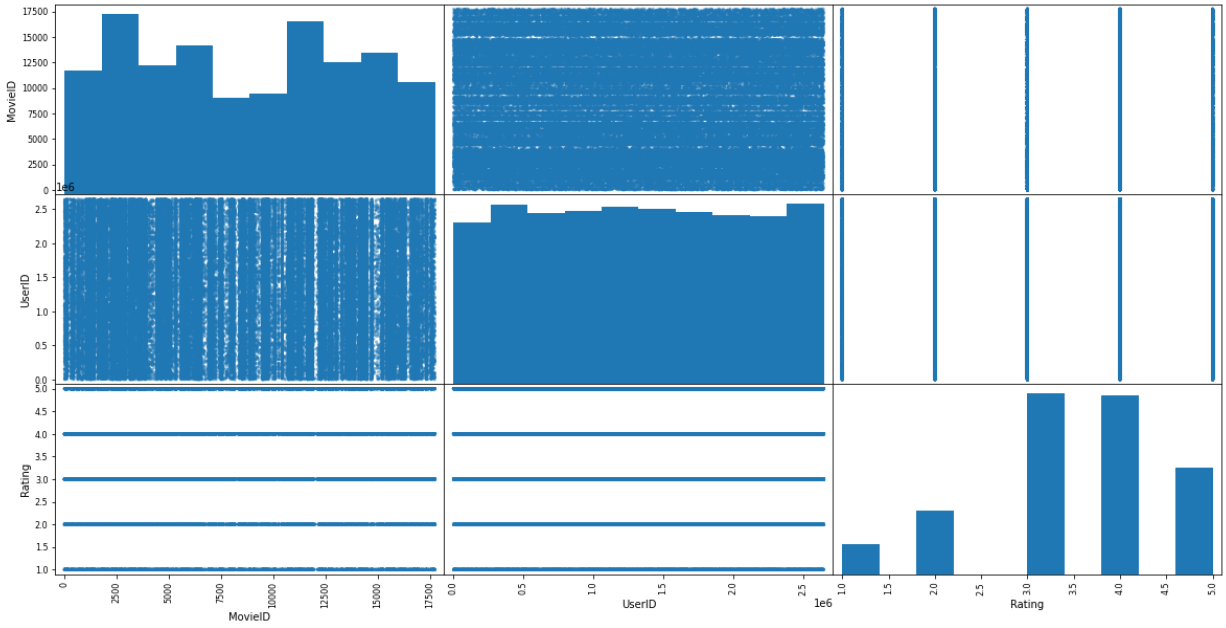

Figure 8: Test Data Distribution – Bar Plot

So far, we have seen about Collaborative Filtering of recommendation systems, understood the Netflix data and saw some visualization of the train & test datasets. Now, comes analyzing task of the dataset.

4. Analyzing the Netflix Data

Initially, the duplicated rows (if any) of training & test datasets are dropped. Then for each feature in the datasets, the number of unique values are computed. The obtained results are as follows:

```
TRAINING DATA:

Count of Unique Movies : 1821
Count of Unique Users : 28978
Count of Unique Ratings : 5
```

Figure 9: Train Data Unique Count of Features

```
TEST DATA:

Count of Unique Movies : 1701
Count of Unique Ratings : 5
Count of Unique Users : 27555
```

Figure 10: Test Data Unique Count of Features

Then, from the UserID feature, a value is selected, and a particular rating is chosen. Now, with these details, on joining both the training & movie datasets, Movie ID and Movie name for the selected UserID & Rating is displayed. The result is shown below:

```
MovieID
1615                The Flintstones: Season 2
2290                    Film School: Season 1
2660                       Wrath of the Ninja
2861                 The Silence of the Lambs
2913                                   Kasoor
2955                       Lie Down with Dogs
4385            The Return of the Living Dead
5562       Woodstock: 3 Days of Peace & Music
6281                                   Madman
6287                        Essence of Echoes
7445     The Office: Series 1: Bonus Material
7511                         Poor White Trash
7577                Who's the Boss?: Season 1
10662                Kipper: Friendship Tails
10947                              Easy Riders
11673                   The Name of the Rose
11812                              Cold & Dark
11837                                    Omagh
12497       Goosebumps: Welcome to Dead House
13614                       The Great Silence
14185                              Bitter Moon
14209                  Grave of the Fireflies
14716                    Confusion of Genders
14807            Viva La Bam: Seasons 2 and 3
Name: Name, dtype: object
```

Figure 11: User 2080843 Rating 5 Movie List

5. Collaborative Filtering Implementation

In order to implement Collaborative Filterting on the Netflix Data, I came across alibrary called Surprise (https://surprise.readthedocs.io/en/stable/index.html). Surprise is an easy-to-use Python scikit for recommender systems.

Surprise has a set of built-in algorithms and datasets. In its simplest form, it only takes a few lines of code to run a cross-validation procedure. Few of the methods are trained with the Netflix data. For each model, cross-validation is applied with 5 folds and the final result is obtained by calculating mean of the results.

The first implemented model is a very popular model, SVD and the obtained results are as follows:

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9990  0.9909  0.9854  0.9947  0.9933  0.9927  0.0045
MAE (testset)     0.7929  0.7858  0.7805  0.7886  0.7862  0.7868  0.0041
Fit time          4.39    4.48    4.35    4.31    4.38    4.38    0.06
Test time         0.24    0.13    0.13    0.14    0.23    0.17    0.05
{'fit_time': (4.38528847694397,
  4.482245922088623,
  4.352317810058594,
  4.311546087265015,
  4.38419246673584),
 'test_mae': array([0.79294374, 0.78575855, 0.78045018, 0.78856928, 0.78621783]),
 'test_rmse': array([0.99901304, 0.99085052, 0.9854397 , 0.99471545, 0.99329606]),
 'test_time': (0.2404794692993164,
  0.1331629753112793,
  0.12585735321044922,
  0.1353905200958252,
  0.23271512985229492)}
```

Figure 12: SVD Model Results – 5 Fold Cross-validation

The next model trained is KNNBasic with Alternating Least Squares (ALS) baseline and the obtained results are as follows:

```
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.1509  1.1573  1.1641  1.1552  1.1614  1.1578  0.0046
MAE (testset)     0.8869  0.8932  0.8981  0.8888  0.8947  0.8924  0.0040
Fit time          0.01    0.03    0.03    0.03    0.03    0.03    0.01
Test time         0.36    0.24    0.22    0.36    0.25    0.29    0.06
{'fit_time': (0.013135671615600586,
  0.033365488052368164,
  0.0333867073059082,
  0.03391838073730469,
  0.033263444900512695),
 'test_mae': array([0.88694215, 0.89321006, 0.89805615, 0.88882871, 0.89472959]),
 'test_rmse': array([1.15094658, 1.1573207 , 1.16409794, 1.15524554, 1.16137992]),
 'test_time': (0.36482834815979004,
  0.2434403896331787,
  0.2236182689666748,
  0.3562657833099365,
  0.2485244274139404)}
```

Figure 13: KNNBasic Model Results – 5 Fold Cross-validation

The next model trained is KNNBaseline with Stochastic Gradient Descent (SGD) baseline and the obtained results are as follows:

```
Evaluating RMSE, MAE of algorithm KNNBaseline on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0670  1.0715  1.0622  1.0553  1.0654  1.0643  0.0054
MAE (testset)     0.8222  0.8299  0.8196  0.8162  0.8244  0.8225  0.0046
Fit time          0.25    0.28    0.28    0.31    0.29    0.28    0.02
Test time         0.32    0.47    0.31    0.46    0.32    0.38    0.07
{'fit_time': (0.2506563663482666,
  0.27977991104125977,
  0.27826762199401855,
  0.3060760498046875,
  0.28669261932373047),
 'test_mae': array([0.82219316, 0.82986171, 0.81962939, 0.81620185, 0.82442852]),
 'test_rmse': array([1.06701539, 1.07146791, 1.06215629, 1.05528087, 1.06539398]),
 'test_time': (0.32434654235839844,
  0.47165679931640625,
  0.30911803245544434,
  0.46346092224121094,
  0.31964778900146484)}
```

Figure 14: KNNBaseline Model Results – 5 Fold Cross-validation

The final model trained on the dataset is KNNWithMeans model and the obtained results are as follows:

```
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0677  1.0619  1.0726  1.0692  1.0687  1.0680  0.0035
MAE (testset)     0.8263  0.8254  0.8317  0.8302  0.8319  0.8291  0.0027
Fit time          0.03    0.06    0.05    0.05    0.05    0.05    0.01
Test time         0.26    0.42    0.26    0.27    0.27    0.30    0.06
{'fit_time': (0.026952028274536133,
  0.055382490158081055,
  0.04926943778991699,
  0.0488283634185791,
  0.05002737045288086),
 'test_mae': array([0.82632707, 0.82541597, 0.83172037, 0.83020007, 0.8318913 ]),
 'test_rmse': array([1.0676821 , 1.06194838, 1.07257637, 1.06922525, 1.06869472]),
 'test_time': (0.2561209201812744,
  0.42163562774658203,
  0.2613227367401123,
  0.265561580657959,
  0.2739732265472412)}
```

Figure 15: KNNWithMeans Model Results – 5 Fold Cross-validation

The final mean results of all the models are as follows:

Model	test_rmse	test_mae	fit_time	test_time
SVD	0.992663	0.786788	4.383118	0.173521
KNNBasic	1.157798	0.892353	0.029414	0.287335
KNNBaseline	1.064263	0.822463	0.280295	0.377646
KNNWithMeans	1.068025	0.829111	0.046092	0.295723

Figure 16: Final Results of all Models

From the above table we can say that SVD model has better performance than the rest of the models. Using surprise library one can build a custom prediction algorithm [5]. This interest me a lot about this library.

6. Conclusion

Recommendation Systems became a part of online platforms. These systems improve their product marketing and helps in the development of their firm. Working on a recommendation systems enlightens how we are seeing recommendation in the online platforms like Amazon, Hotstar etc. All these platforms have millions or may be billions of records of data for recommendation to a user. In turn the number of users using the platform may also be in millions, in the case of Netflix. Building a recommendation system in standard Jupyter IDE does not work this huge amount of data. Implementing the recommendation system on AWS EMR cluster makes is workable and satisfies the user with the appropriate recommendations.

7. References

[1] https://en.wikipedia.org/wiki/Collaborative_filtering

[2] http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

[3] https://blog.dominodatalab.com/recommender-systems-collaborative-filtering/

[4] https://www.kaggle.com/jieyima/netflix-recommendation-collaborative-filtering

[5] https://surprise.readthedocs.io/en/stable/building_custom_algo.html