DSCI 6007 - Final Project
Fall 2020

Movie Recommendation by Collaborative Filtering using the Netflix Data

By:
Archana Dasharath Marol

Under the guidance of:
Prof. Vahid Behzadan

Table of Contents

Chapter One

1 MOTIVATION

Due to the predominance of the internet, more and more data is collected every day, so people now have too many options to choose from, so we are moving from the age of information to the age of recommendation. In the past, the number of movies that can be placed in a store was depended on the size of that store. Whereas, in the present days, the internet allows people to access abundant resources online. For instance, Netflix and Amazon has huge collection of movies. This has resulted in a new problem as people has a hard time selecting the items they want to see. This is where the recommender system comes in.

Big data is the driving force behind Recommendation systems. A typical Recommendation system cannot do its job without sufficient data and big data supplies plenty of user data such as past purchases, browsing history, and feedback for the Recommendation systems to provide relevant and effective recommendations. Hence, data engineering approaches are useful to address these problems.

The goal of this project is to analyze the NETFLIX data using SPARK and based on the outcomes of this analysis, develop a feasible and efficient implementation of the collaborative filtering algorithm in PYSPARK. In this project, around 100,000 movie ratings are predicted for users in a subset of the original NETFLIX data issued for the NETFLIX Prize. This challenge aimed at substantially improving the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences. The GitHub link for the project implementation is given at the end of this report.

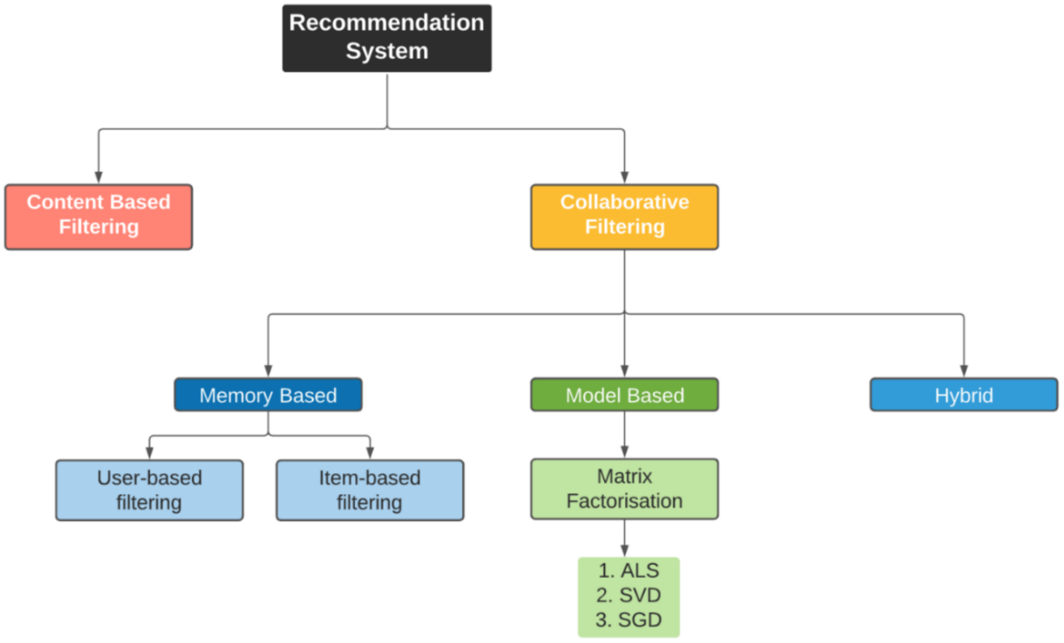
Chapter Two

2 INTRODUCTION AND APPROACH

Recommender System is an information filtering tool that seeks to predict which product a user will like, and based on that, recommends a few products to the users. The two widely used approaches for building a recommender system are 1) the content-based filtering 2) the collaborative filtering.

Content-based filtering analyses the nature of each item and aims to find the insights of the data to identify the user preferences.

Collaborative filtering, on the other hand, does not require any information about the items or the user themselves. It recommends the item based on user past experience and behavior. The key idea behind Collaborative Filtering is that similar users share similar interest, people with similar interest tends to like similar items. Hence those items are recommended to similar set of users.

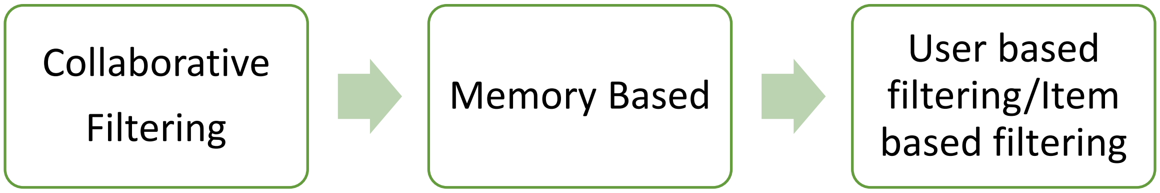


Collaborative Filtering algorithm has the ability to do feature learning on its own, which means that it can start to learn for itself what features to use.

- Memory Based: It basically identifies the clusters of users to calculate the interactions of one specific user to predict the interactions of other similar users. The second thought process will be identifying the items clusters rated by user A and predicting user's interaction with item B. Memory based methods fail while dealing with large sparse matrices.
  - User-user: In order to make a new recommendation to a user, user-user method roughly tries to identify users with the most similar "interactions profile" (nearest neighbours) in order to suggest items that are the most popular among these neighbours (and that are "new" to our user). This method is said to be "user-centred" as it represent users based on their interactions with items and evaluate distances between users.
  - Item-item: To make a new recommendation to a user, the idea of item-item method is to find items like the ones the user already "positively" interacted with. Two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is said to be "item-centred" as it represent items based on interactions users had with them and evaluate distances between those items.
- Model Based: spark.ml currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.ml uses the alternating least squares (ALS) algorithm to learn these latent factors. ALS is basically a Matrix Factorization approach to implement a recommendation algorithm you decompose your large user/item matrix into lower dimensional user factors and item factors.
- Hybrid: Consolidated both types of information with the aim of avoiding problems that are generated when working with one kind of Recommender systems.

Some of the advantages of using Collaborative Filtering is that it takes other user's ratings into consideration, it doesn't need to study or extract information from the recommended item, and it adapts to the user's interests which might change over time.

Approach for the project:



The project follows the below approaches:

- Similarity measure CORRELATION
- Prediction method weighted
- User-user model
- Evaluation- Mean Absolute Error and Root Mean Squared Error

Evaluation Methods:

There are many evaluation metrics but one of the most popular metric used to evaluate accuracy of predicted ratings are:

1) Root Mean Squared Error (RMSE)
2) Mean Absolute Error (MAE)

MAE is average of the differences between values predicted by a model or an estimator and the values observed. Meaning, it is measure of difference between the Actual and predicted values.

RMSE is just the square root of MSE. The predicted values can positive or negative as they under or overestimates the actual value. Squaring the residuals, averaging the squares, and taking the square root gives us the RMSE error.

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

Chapter Three

3 SYSTEM CONFIGURATION

The goal of this project is to build a recommendation system for the NETFLIX data using SPARK.

1. Let’s first see how to set the EMR Cluster in AWS account to analyze and implement the approach in a Jupyter notebook of EMR. Below are the steps to be followed:
   - Login to the AWS Account and click on ‘Amazon EMR’
   - Create a EC2 key pair
   - Click on ‘Create cluster’ and use the below configurations:
     - Give a cluster name
     - Choose Spark: Spark 2.4.7 on Hadoop 2.10.1 YARN and Zeppelin 0.8.2
     - Choose m4.xlarge or more
     - Choose the EC2 key pair created previously
     - Click on ‘Create cluster’
     - Wait till the cluster is created (till it shows ‘waiting’)
   - Once the cluster is created, open the Security Groups for the Master node, and add 2 inbound rules for SSH (anywhere) and Custom TCP Port 8888 (anywhere)
   - SSH to the master node, the username is hadoop
   - Run the below commands into the terminal:
     - sudo pip3 install pyyaml ipython jupyter ipyparallel pandas boto -U
     - export PYSPARK_DRIVER_PYTHON=/usr/local/bin/jupyter
     - export PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=0.0.0.0 --port=8888"
     - source ~/.bashrc
     - pyspark

- Use the token and open in browser like this : http://ec2-XXX-XXX-XXXXX.comput1.amazonaws.com:8888/tree?token=000111222333444555666777888999aaabbbcccdddeeefff#)
- The link opens the jupyter notebook
- Once the jupyter notebook is ready to use, the data can be imported using the s3 bucket created before with the datasets

Uploading the data into the S3 bucket:

- Login to the AWS Account and click on 'Amazon S3'
- Create a bucket and upload the dataset into the bucket
- This bucket can be accessed in the jupyter notebook created previously to load the data into the notebook and use it further for analysis

2. In this project, PySpark is used as coding language. Pyspark is the collaboration of Apache Spark and Python. Apache Spark is an open-source cluster-computing framework, built around speed, ease of use, and streaming analytics whereas Python is a general-purpose, high-level programming language.

Chapter Four

4 BIG DATA APPLICATION/DATSET

4.1. Analyzing the Netflix Data:

This dataset is a subset of the data provided as part of the Netflix Prize. TrainingRatings.txt and TestingRatings.txt are respectively the training set and test set. Each of them has lines having the format: MovieID, UserID, Rating. Each row represents a rating of a movie by some user. Ratings are integers from 1 to 5. The training set has 3,255,352 million ratings, and the test set has 100,478.

These datasets were stored in S3 bucket and then were imported into the jupyter notebook created using EMR cluster. The datasets were analyzed using pypark commands. The following analysis was implemented to understand the data.

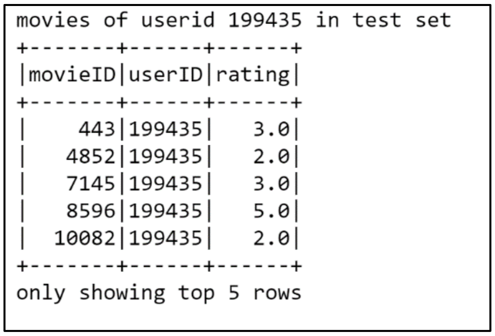
a) **Distinct items and distinct users are there in the test set**:

The test dataset contains 1,701 movies and 27,555 users.

b) Overlappingitems and Overlappingusers:

Estimated average overlap of items for users

- A userid= 199435 is picked from the test dataset.
- In the test set, movie ids are extracted for which the user has rated. There were 8

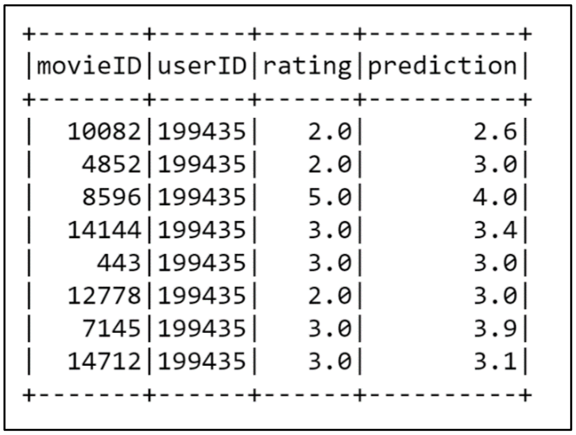


```
movies of userid 199435 in test set
+-------+------+------+
|movieID|userID|rating|
+-------+------+------+
|    443|199435|   3.0|
|   4852|199435|   2.0|
|   7145|199435|   3.0|
|   8596|199435|   5.0|
|  10082|199435|   2.0|
+-------+------+------+
only showing top 5 rows
```

- Then all the users in the training set who have rated the same movies are extracted.

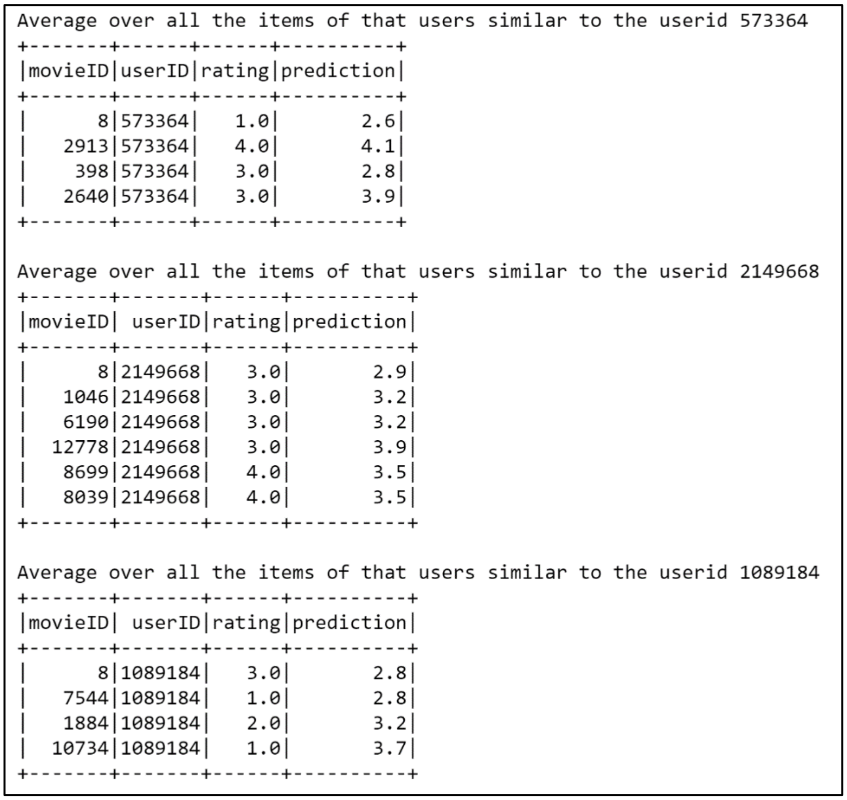
prediction

- The movieids are grouped by averaging the rating column to get average overlap of items rated by the users in the training set for users in the test set. The average column will be out predicted column
- Then this is joined with the original dataset, to see the predicted and actual values



```
+-------+------+------+----------+
|movieID|userID|rating|prediction|
+-------+------+------+----------+
|  10082|199435|   2.0|       2.6|
|   4852|199435|   2.0|       3.0|
|   8596|199435|   5.0|       4.0|
|  14144|199435|   3.0|       3.4|
|    443|199435|   3.0|       3.0|
|  12778|199435|   2.0|       3.0|
|   7145|199435|   3.0|       3.9|
|  14712|199435|   3.0|       3.1|
+-------+------+------+----------+
```

The same approach was **repeated for three more userids** and their predicted values are shown:



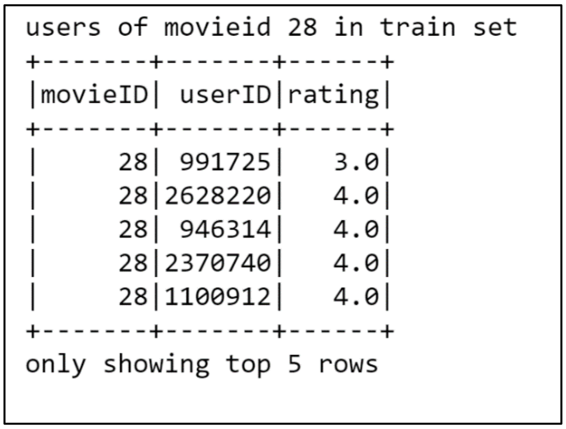
```
Average over all the items of that users similar to the userid 573364
+-------+------+------+----------+
|movieID|userID|rating|prediction|
+-------+------+------+----------+
|      8|573364|   1.0|       2.6|
|   2913|573364|   4.0|       4.1|
|    398|573364|   3.0|       2.8|
|   2640|573364|   3.0|       3.9|
+-------+------+------+----------+

Average over all the items of that users similar to the userid 2149668
+-------+-------+------+----------+
|movieID| userID|rating|prediction|
+-------+-------+------+----------+
|      8|2149668|   3.0|       2.9|
|   1046|2149668|   3.0|       3.2|
|   6190|2149668|   3.0|       3.2|
|  12778|2149668|   3.0|       3.9|
|   8699|2149668|   4.0|       3.5|
|   8039|2149668|   4.0|       3.5|
+-------+-------+------+----------+

Average over all the items of that users similar to the userid 1089184
+-------+-------+------+----------+
|movieID| userID|rating|prediction|
+-------+-------+------+----------+
|      8|1089184|   3.0|       2.8|
|   7544|1089184|   1.0|       2.8|
|   1884|1089184|   2.0|       3.2|
|  10734|1089184|   1.0|       3.7|
+-------+-------+------+----------+
```

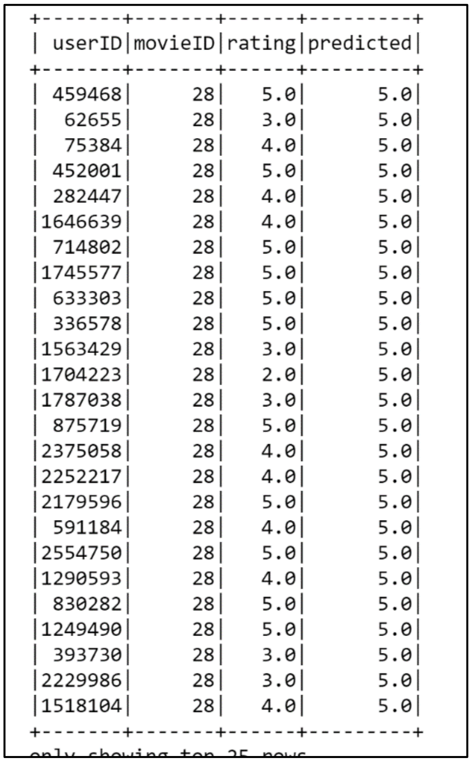
Estimated average overlap of users for items

- A item id= 28 is picked from the test dataset.
- In the test set, user ids are extracted who has rated the movie 28.



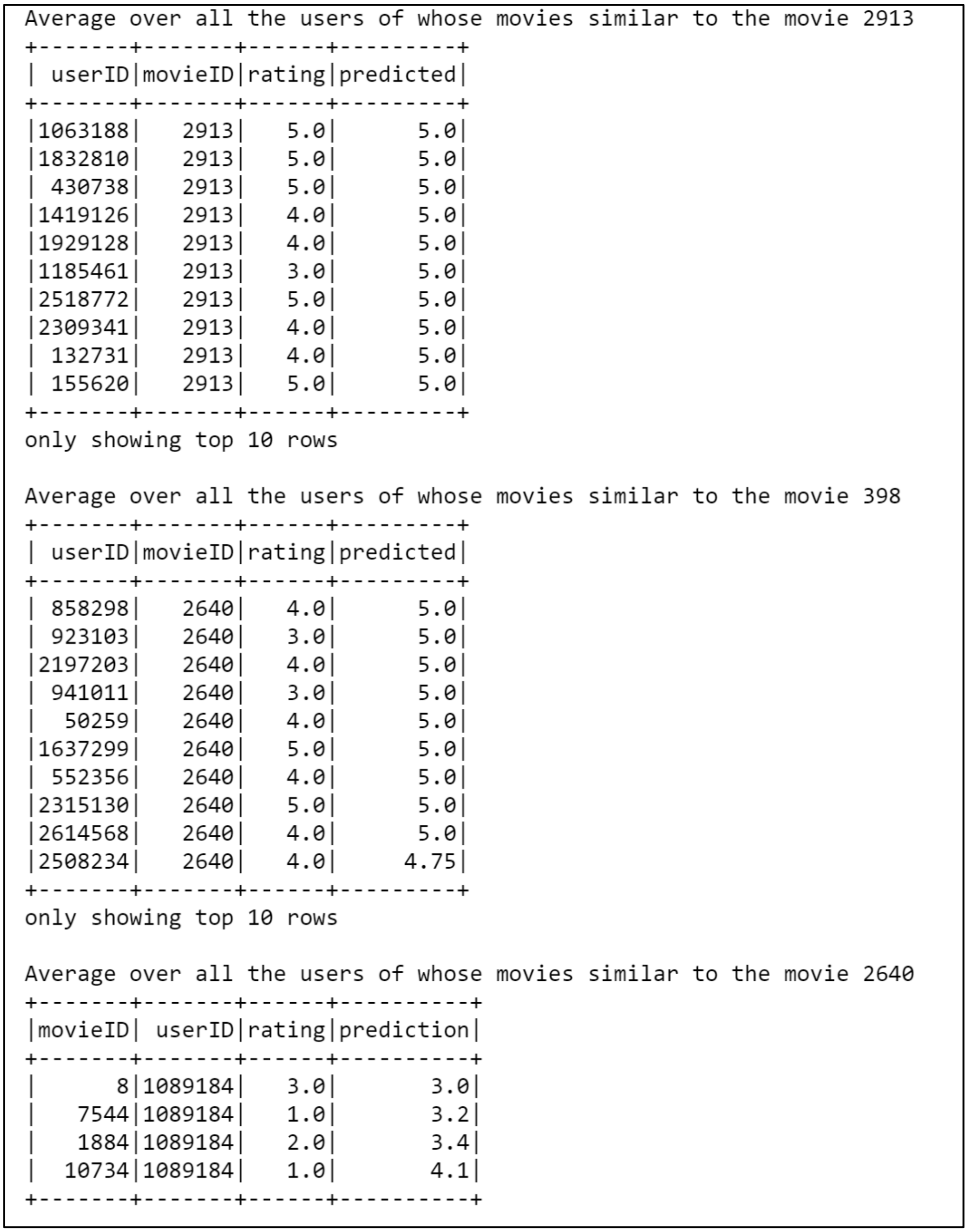
```
users of movieid 28 in train set
+-------+-------+------+
|movieID| userID|rating|
+-------+-------+------+
|     28| 991725|   3.0|
|     28|2628220|   4.0|
|     28| 946314|   4.0|
|     28|2370740|   4.0|
|     28|1100912|   4.0|
+-------+-------+------+
only showing top 5 rows
```

- Then all the movies in the training set that are watched by the same users are extracted
- Lastly, the userids are grouped by averaging the rating column to get the average overlap of users that rated items in the training set for items appearing in the test set



```
+-------+-------+------+---------+
| userID|movieID|rating|predicted|
+-------+-------+------+---------+
| 459468|     28|   5.0|      5.0|
|  62655|     28|   3.0|      5.0|
|  75384|     28|   4.0|      5.0|
| 452001|     28|   5.0|      5.0|
| 282447|     28|   4.0|      5.0|
|1646639|     28|   4.0|      5.0|
| 714802|     28|   5.0|      5.0|
|1745577|     28|   5.0|      5.0|
| 633303|     28|   5.0|      5.0|
| 336578|     28|   5.0|      5.0|
|1563429|     28|   3.0|      5.0|
|1704223|     28|   2.0|      5.0|
|1787038|     28|   3.0|      5.0|
| 875719|     28|   5.0|      5.0|
|2375058|     28|   4.0|      5.0|
|2252217|     28|   4.0|      5.0|
|2179596|     28|   5.0|      5.0|
| 591184|     28|   4.0|      5.0|
|2554750|     28|   5.0|      5.0|
|1290593|     28|   4.0|      5.0|
| 830282|     28|   5.0|      5.0|
|1249490|     28|   5.0|      5.0|
| 393730|     28|   3.0|      5.0|
|2229986|     28|   3.0|      5.0|
|1518104|     28|   4.0|      5.0|
+-------+-------+------+---------+
only showing top 25 rows
```

The same approach was **repeated for three more movieids** and their predicted values are shown:



```
Average over all the users of whose movies similar to the movie 2913
+-------+-------+------+---------+
| userID|movieID|rating|predicted|
+-------+-------+------+---------+
|1063188|   2913|   5.0|      5.0|
|1832810|   2913|   5.0|      5.0|
| 430738|   2913|   5.0|      5.0|
|1419126|   2913|   4.0|      5.0|
|1929128|   2913|   4.0|      5.0|
|1185461|   2913|   3.0|      5.0|
|2518772|   2913|   5.0|      5.0|
|2309341|   2913|   4.0|      5.0|
| 132731|   2913|   4.0|      5.0|
| 155620|   2913|   5.0|      5.0|
+-------+-------+------+---------+
only showing top 10 rows

Average over all the users of whose movies similar to the movie 398
+-------+-------+------+---------+
| userID|movieID|rating|predicted|
+-------+-------+------+---------+
| 858298|   2640|   4.0|      5.0|
| 923103|   2640|   3.0|      5.0|
|2197203|   2640|   4.0|      5.0|
| 941011|   2640|   3.0|      5.0|
|  50259|   2640|   4.0|      5.0|
|1637299|   2640|   5.0|      5.0|
| 552356|   2640|   4.0|      5.0|
|2315130|   2640|   5.0|      5.0|
|2614568|   2640|   4.0|      5.0|
|2508234|   2640|   4.0|     4.75|
+-------+-------+------+---------+
only showing top 10 rows

Average over all the users of whose movies similar to the movie 2640
+-------+-------+------+----------+
|movieID| userID|rating|prediction|
+-------+-------+------+----------+
|      8|1089184|   3.0|       3.0|
|   7544|1089184|   1.0|       3.2|
|   1884|1089184|   2.0|       3.4|
|  10734|1089184|   1.0|       4.1|
+-------+-------+------+----------+
```

c) Best approach to implement the collaborative filtering

There are several methods of finding similar users such as JACCARD, COSINE, NORMALISATION, and the one we will be using here is going to be based on the Correlation Function. It is used to measure the strength of a linear association between two variables. It is used to measure the strength of a linear association between two variables. The formula for finding this coefficient between sets X and Y with N values can be seen in the image below.

Why are we using Correlation?

Pearson correlation is invariant to scaling, i.e. multiplying all elements by a nonzero constant or adding any constant to all elements. For example, if you have two vectors X and Y,then, pearson(X, Y) == pearson(X, 2 * Y + 3). This is a pretty important property in recommendation systems because for example two users might rate two series of items totally different in terms of absolute rates, but they would be similar users (i.e. with similar ideas) with similar rates in various scales.

The values given by the formula vary from r = -1 to r = 1, where 1 forms a direct correlation between the two entities (it means a perfect positive correlation) and -1 forms a perfect negative correlation.

In our case, a 1 means that the two users have similar tastes while a -1 means the opposite.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Jaccard distance is not a suitable?

Jaccard distance is not a suitable measure for the kind of data we are considering, because We could ignore values in the matrix and focus only on the sets of items rated meaning it takes the union and intersections of user A and B. If our problem statement was only about watching movies and not about ratings, then Jaccard distance would be a good choice to use. But, in this project, we are dealing with detailed ratings (1 being lowest and 5 being highest), the Jaccard distance loses important information.

d) Applying Correlation Coefficient and rerunning the code

User Based recommendation:

The process for creating a User Based recommendation system is as follows:

- First select a user with the movies the user has watched from the test data, then filter out only that the user has given 5 rating, this will help for better recommendation
- Based on his rating to movies, find the top X neighbours/ similar users in the train dataset
- Get the watched movie record of the user for each neighbour.
- Calculate a similarity score using the formula
- Recommend the items with the highest score

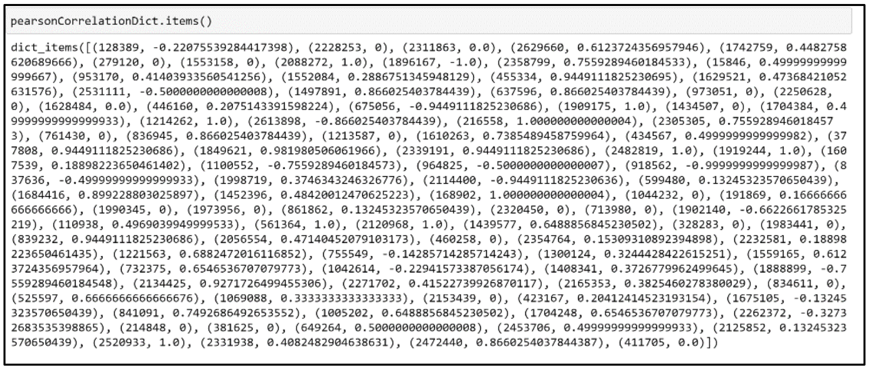
Implementation:

- ❖ Let's collect all the movies that a user id= 199435 has watched from test data. Then getting all the users who has watched the same movies from the train data.

Dataframe1 = dataset of movies userid= 199435 has watched

Dataframe2 = each group of the users who has watched the same movies

- ❖ Creating a loop for every user group in our Dataframe2 and then:
  - o Get the rating scores for the movies that the group and Dataframe1 have in common
  - o And then store them in a temporary buffer variable in a list format to facilitate future calculations
  - o Put each user group rating in a list format
  - o Calculating the Pearson Correlation between input user and subset group. Then storing the Pearson Correlation in a dictionary, where the key is the user Id and the value is the coefficient. pearsonCorrelationDict = {}. Below is the dictionary from jupyter notebook:



```
pearsonCorrelationDict.items()
```

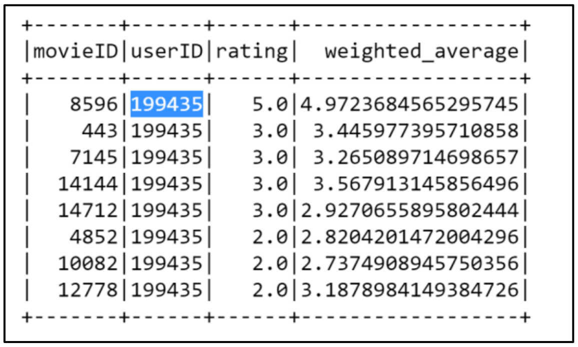
```
dict_items([(128389, -0.22075539284417398), (2228253, 0), (2311863, 0.0), (2629660, 0.6123724356957946), (1742759, 0.4482758
620689666), (279120, 0), (1553158, 0), (2088272, 1.0), (1896167, -1.0), (2358799, 0.7559289460184533), (15846, 0.49999999999
999667), (953170, 0.41403933560541256), (1552084, 0.2886751345948129), (455334, 0.9449111825230695), (1629521, 0.47368421052
631576), (2531111, -0.5000000000000008), (1497891, 0.866025403784439), (637596, 0.866025403784439), (973051, 0), (2250628,
0), (1628484, 0.0), (446160, 0.2075143391598224), (675056, -0.9449111825230686), (1909175, 1.0), (1434507, 0), (1704384, 0.4
9999999999999933), (1214262, 1.0), (2613898, -0.866025403784439), (216558, 1.0000000000000004), (2305305, 0.755928946018457
3), (761430, 0), (836945, 0.866025403784439), (1213587, 0), (1610263, 0.7385489458759964), (434567, 0.4999999999999982), (37
7808, 0.9449111825230686), (1849621, 0.981980506061966), (2339191, 0.9449111825230686), (2482819, 1.0), (1919244, 1.0), (160
7539, 0.18898223650461402), (1100552, -0.7559289460184573), (964825, -0.5000000000000007), (918562, -0.9999999999999987), (8
37636, -0.49999999999999933), (1998719, 0.3746343246326776), (2114400, -0.9449111825230636), (599480, 0.13245323570650439),
(1684416, 0.899228803025897), (1452396, 0.48420012470625223), (168902, 1.0000000000000004), (1044232, 0), (191869, 0.16666666
666666666), (1990345, 0), (1973956, 0), (861862, 0.13245323570650439), (2320450, 0), (713980, 0), (1902140, -0.6622661785325
219), (110938, 0.4969039949999533), (561364, 1.0), (2120968, 1.0), (1439577, 0.6488856845230502), (328283, 0), (1983441, 0),
(839232, 0.9449111825230686), (2056554, 0.47140452079103173), (460258, 0), (2354764, 0.15309310892394898), (2232581, 0.18898
223650461435), (1221563, 0.6882472016116852), (755549, -0.14285714285714243), (1300124, 0.3244428422615251), (1559165, 0.612
3724356957964), (732375, 0.6546536707079773), (1042614, -0.22941573387056174), (1408341, 0.3726779962499645), (1888899, -0.7
559289460184548), (2134425, 0.9271726499455306), (2271702, 0.41522739926870117), (2165353, 0.3825460278380029), (834611, 0),
(525597, 0.6666666666666676), (1069088, 0.3333333333333333), (2153439, 0), (423167, 0.20412414523193154), (1675105, -0.13245
323570650439), (841091, 0.7492686492653552), (1005202, 0.6488856845230502), (1704248, 0.6546536707079773), (2262372, -0.3273
2683535398865), (214848, 0), (381625, 0), (649264, 0.5000000000000008), (2453706, 0.49999999999999933), (2125852, 0.13245323
570650439), (2520933, 1.0), (2331938, 0.4082482904638631), (2472440, 0.8660254037844387), (411705, 0.0)])
```

- ❖ Taking the weighted average of the ratings of the movies using the Pearson Correlation as the weight.
- ❖ Calculate the similarity index by multiply the movie rating by its weight, then sum up the new ratings and divide it by the sum of the weights. It shows the idea of all similar users to candidate movies for the input user.
- ❖ Next, we have to normalize the weighted rating values. This is done by dividing the sum of weighted ratings by the sum of the similarity index for users.

Prediction values:

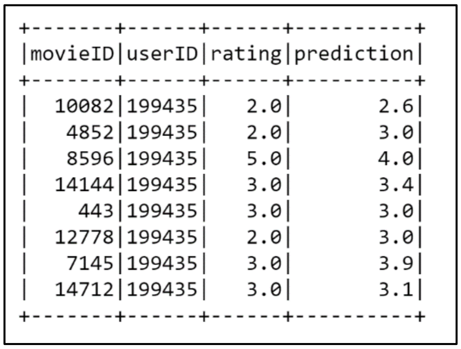
- ❖ The predicted ratings for the user are the final weighted average column.
- ❖ Let us see the actual rating and predicted rating of these movies by the user from the test data (table 1). We can also compare the values with the previous results which was done without Correlation Coefficient (table2). The new results are better.

Table 1



```
+-------+------+------+------------------+
|movieID|userID|rating|  weighted_average|
+-------+------+------+------------------+
|   8596|199435|   5.0|4.9723684565295745|
|    443|199435|   3.0| 3.445977395710858|
|   7145|199435|   3.0| 3.265089714698657|
|  14144|199435|   3.0| 3.567913145856496|
|  14712|199435|   3.0|2.9270655895802444|
|   4852|199435|   2.0|2.8204201472004296|
|  10082|199435|   2.0|2.7374908945750356|
|  12778|199435|   2.0|3.1878984149384726|
+-------+------+------+------------------+
```

Table 2



```
+-------+------+------+----------+
|movieID|userID|rating|prediction|
+-------+------+------+----------+
|  10082|199435|   2.0|       2.6|
|   4852|199435|   2.0|       3.0|
|   8596|199435|   5.0|       4.0|
|  14144|199435|   3.0|       3.4|
|    443|199435|   3.0|       3.0|
|  12778|199435|   2.0|       3.0|
|   7145|199435|   3.0|       3.9|
|  14712|199435|   3.0|       3.1|
+-------+------+------+----------+
```

The predicted and actual values are close values. So, our recommendation worked quite well for the amount of data that we have given.

4.2 Collaborative Filtering Implementation:

Now, the whole data can be implemented using the combination of the approaches that were tried so far and then evaluated using evaluation methods:

- Similarity measure CORRELATION
- Prediction method weighted
- User-user model
- Evaluation- Mean Absolute Error and Root Mean Squared Error

In this project, the approach is applied for all users from the test dataset. The implementation is shown in jupyter notebook and explained below:

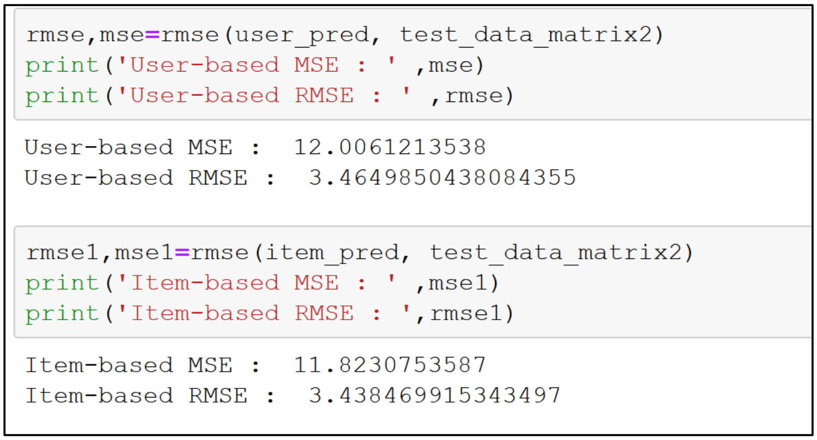
The implementation is shown in the jupyter notebook

Running the user-item pairs in Testing Ratings:

- All the users of the test dataset were chosen to run the approach. These users's movies are filtered for the users in train dataset.
- Correlation matrix was run for the entire data set

Evaluation:

The mean absolute error and root mean square error were calculated for the test data and below are the values for user-user and item-item predictions



```
rmse,mse=rmse(user_pred, test_data_matrix2)
print('User-based MSE : ' ,mse)
print('User-based RMSE : ' ,rmse)
```

```
User-based MSE :  12.0061213538
User-based RMSE :  3.4649850438084355
```

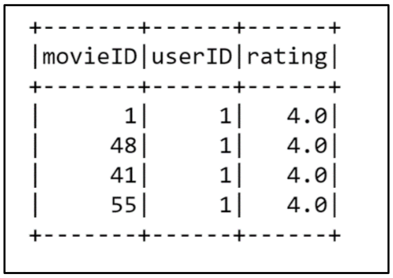
```
rmse1,mse1=rmse(item_pred, test_data_matrix2)
print('Item-based MSE : ' ,mse1)
print('Item-based RMSE : ',rmse1)
```

```
Item-based MSE :  11.8230753587
Item-based RMSE :  3.438469915343497
```

Trying approach work for my own preferences:

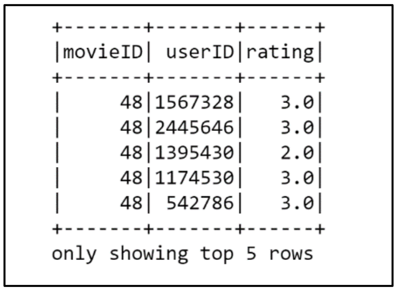
I choose four movies and gave useid =1 and choose to give 4 rating for all:

1) Justice League whose movieID = 48
2) Dinosaur Planet whose movieID =1
3) Horror Vision whose movieID = 41
4) Jade whose movieID = 55



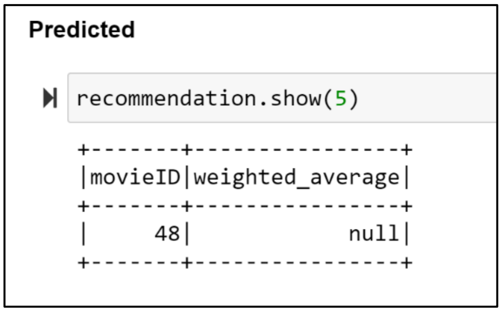
```
+-------+------+------+
|movieID|userID|rating|
+-------+------+------+
|      1|     1|   4.0|
|     48|     1|   4.0|
|     41|     1|   4.0|
|     55|     1|   4.0|
+-------+------+------+
```

Getting all the users who watched same movies as me. Looks like training data has only movieid =48 among the movies I watched.



```
+-------+-------+------+
|movieID| userID|rating|
+-------+-------+------+
|     48|1567328|   3.0|
|     48|2445646|   3.0|
|     48|1395430|   2.0|
|     48|1174530|   3.0|
|     48| 542786|   3.0|
+-------+-------+------+
only showing top 5 rows
```

As per the prediction, users in the training dataset has no match as my taste and that's why the value is null.



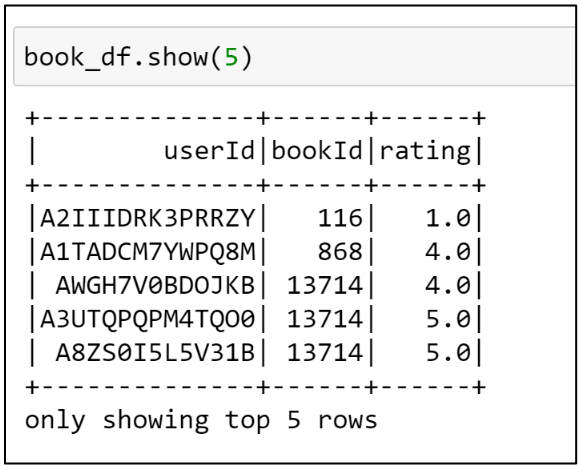
Predicted

```
recommendation.show(5)
```

```
+-------+----------------+
|movieID|weighted_average|
+-------+----------------+
|     48|            null|
+-------+----------------+
```

Implement the approach on Amazon Product Data:

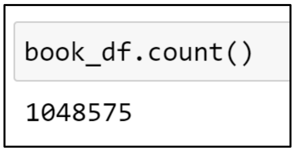
A dataset called booksRating is stored in the S3 bucket of AWS. The booksRating has three columns, userid, bookid and rating as shown below:



```
book_df.show(5)
```

```
+-------------+------+------+
|       userId|bookId|rating|
+-------------+------+------+
|A2IIIDRK3PRRZY|   116|   1.0|
|A1TADCM7YWPQ8M|   868|   4.0|
| AWGH7V0BDOJKB| 13714|   4.0|
|A3UTQPQPM4TQO0| 13714|   5.0|
| A8ZS0I5L5V31B| 13714|   5.0|
+-------------+------+------+
only showing top 5 rows
```

There are 1048575 rows, and this dataset is split into test and train of 80-20%



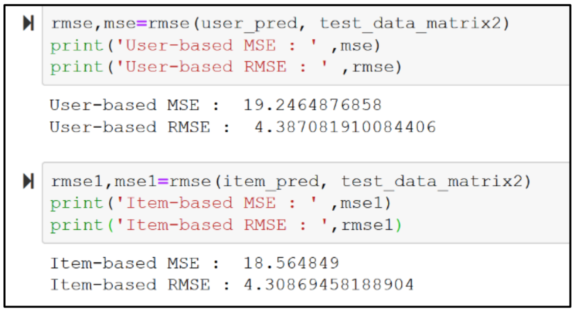
```
book_df.count()
```

```
1048575
```

All the approaches below were implemented:

- Similarity measure CORRELATION
- Prediction method weighted
- User-user model
- Evaluation- Mean Absolute Error and Root Mean Squared Error

The MAE and RMSE of the amazon dataset are :



```
rmse,mse=rmse(user_pred, test_data_matrix2)
print('User-based MSE : ' ,mse)
print('User-based RMSE : ' ,rmse)
```

```
User-based MSE :  19.2464876858
User-based RMSE :  4.387081910084406
```

```
rmse1,mse1=rmse(item_pred, test_data_matrix2)
print('Item-based MSE : ' ,mse1)
print('Item-based RMSE : ',rmse1)
```

```
Item-based MSE :  18.564849
Item-based RMSE : 4.30869458188904
```

Chapter Five

CONCLUSION

Collaborative filtering approach was used to predict the ratings of the Netflix users. Correlation similarity measure was used measure the strength of a linear association between input users and the users. Weighted average of the ratings of the movies was computed using the Pearson Correlation as the weight. User-user model for completed dataset was implanted and the approach was evaluated using methods like Mean Absolute Error and Root Mean Squared Error.

Overall, we can say that collaborative filtering takes other user’s ratings into consideration, it does not need to study or extract information from the recommended item, and it adapts to the user’s interests which might change over time.

REFERENCE

- https://spark.apache.org/docs/latest/ml-collaborative-filtering.html
- https://spark.apache.org/docs/latest/ml-guide.html
- https://spark.apache.org/docs/latest/api/python/pyspark.ml.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.euclidean_distances.html