COLLABORATIVE FILTERING USING THE NETFLIX DATA

Student : Venu Korada
Professor: Dr .Vahid Behzadan
Course: DSCI 6007

Motivation for the Collabrative filtering approach:

What is "**collaborative filtering**."

Some firms using recommendation engines would suggest products to you based on the common buying patterns. Say you bought/watched an item X They group you with several other users who have bought/watched X , and then they suggests you buy other things that they've bought.

Using the recommender system will enhance the customers satisfaction if they have thoroughly enjoyed the content . Now coming to Netflix you watch a movie and Netflix suggest you different movies based on several concepts. There can be 2 types of similarities in liking the recommended content i.e 2 people having similar mindset and like towards similar genre of movies. We can either suggest a person the next movie to watch based on comparing the Person1 previous interests matching with a group of peoples previous interests or comparing the aspects of a movie and suggesting based on movie similarity.

The above mentioned examples are of user-user model or item-item model .

I wanted to see how we can modify the similarity prediction and how it works so that is the motivation for Netflix project.

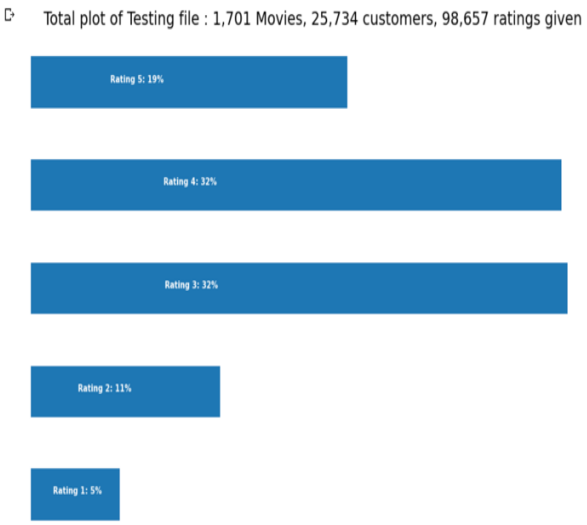
Motivation for the setup used:

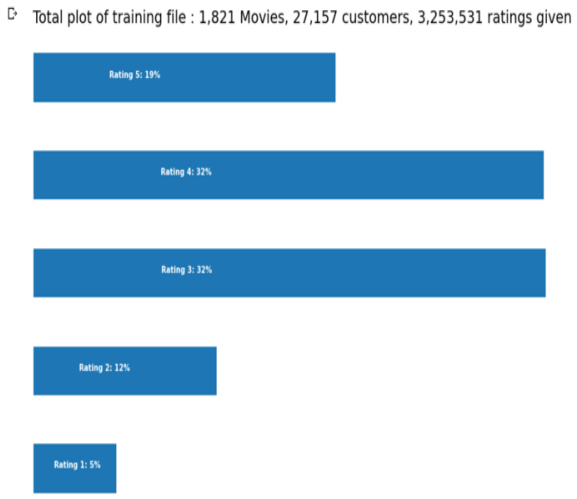
The main important point of this project is to highlight the importance of AWS EMR cluster the pyspark application and how easily we can process the data without any charges incurred at higher amounts than of buying a potential software cluster setup. So for such highluy confoigured cluster the project was choosen as the Netflix data.

Lets speak about the data it has about several entries and lets see them detailly.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. The dataset contains 1821 movies and 28978 users in all. Ratings are integers from 1 to 5. The training set has 3.25 million ratings, and the test set has 100,000.

What EDA have I performed . If you observe below the dataset has been segregated based on rating (on a scale of 1-5) and at what percentage does the ratings stand. It is done for both training as well as testing file.

We have plotted on the total values of movies and customers But the count on top of it depicts the unique number of values for movies and customers in the dataset .



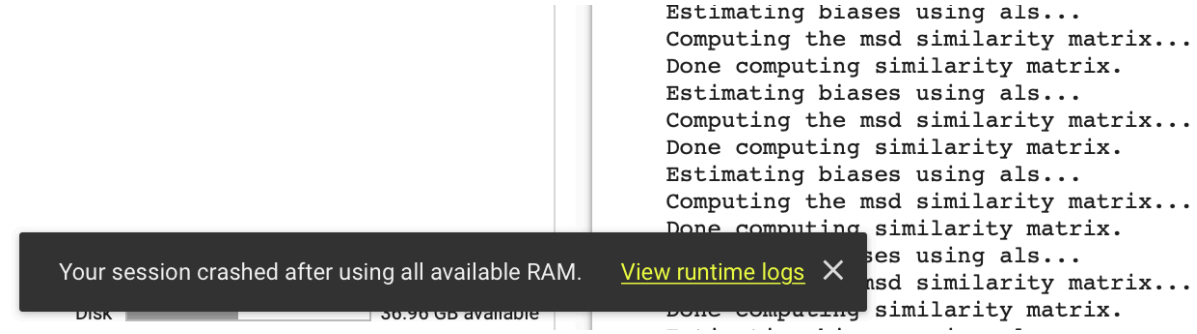


No I will show you a screenshot of what happened when I just tried to run the code on google colab with the GPU and extended RAM. The total training file couldnot be trained for SVD nor the KNNBaseline model. So I have just randomly segregated the data of 100000 inputs and tried creating the models and training it . it ran successfully for SVD but for Kmeans (KABOOM!!!!) the system crashed because of lack of RAM.

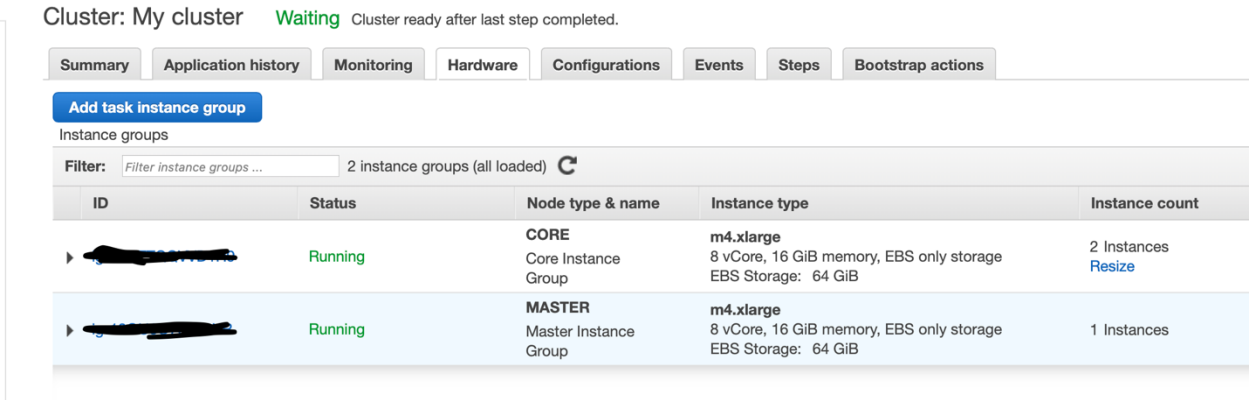
```
[ ] from surprise import KNNBaseline
    from surprise import Dataset
    from surprise.model_selection import GridSearchCV


    sim_options = {
        "name": ["msd", "cosine"],
        "min_support": [3, 4, 5],
        "user_based": [False, True],
    }

    param_grid = {"sim_options": sim_options}

    gs = GridSearchCV(KNNBaseline, param_grid, measures=["rmse", "mae"], cv=3)
    gs.fit(data)

    print(gs.best_score["rmse"])
    print(gs.best_params["rmse"])
```

```
Estimating biases using als...
Computing the msd similarity matrix...
```

The system crashed just trying to compute the MSD similarity matrix where instead it has several computation left behind like the below ones. Of course it is still computing.



So the main Motivation for this project is to handle large data and process it on certain highend technologies like AWS, EMR cluster, Spark App.



System Configuration – how did you setup and configure the EMR cluster

With the help of the AWS educate account provided as a part of the coursework we had the opportunity to work on the project on AWS. I have Created a EMR cluster with the below specified applications.

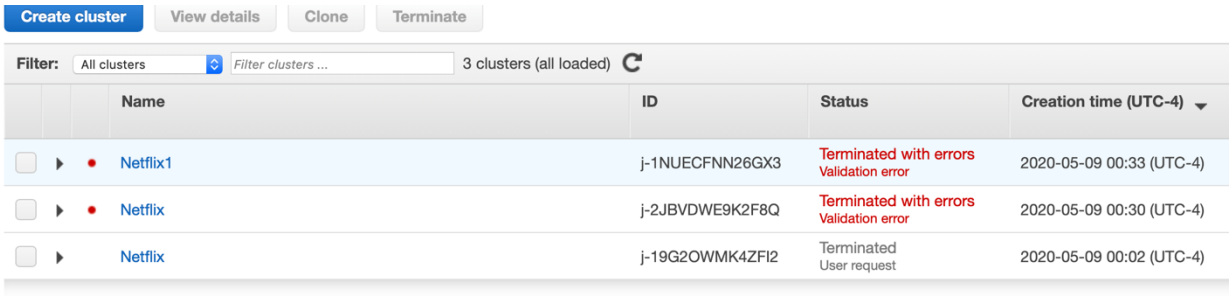
Ganglia 3.7.2, Spark 2.4.4, Zeppelin 0.8.2

Then gave the Master node and 2 core nodes the m4.xlarge and still felt that's bit less based on the datasize.

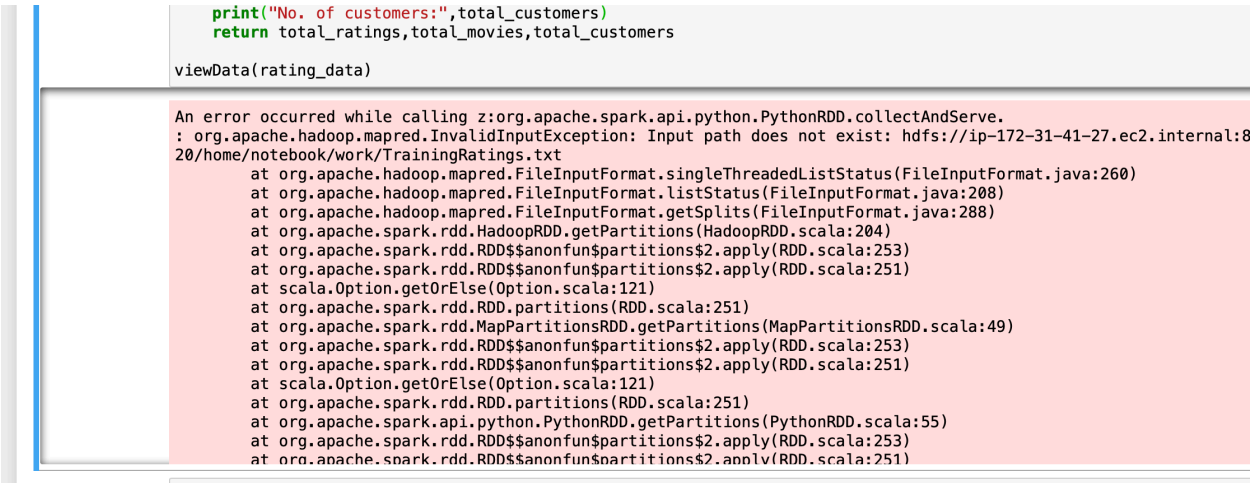
m4.xlarge
8 vCore, 16 GiB memory, EBS only storage
EBS Storage:64 GiB

That's it we create the cluster pretty simple. And we also need to specify a .pem file for mac and ppk file for windows. The permissions should be later changed to 0400 instead of any othere permissions like 777 we never should give full permisiions to the key file.

We need to wait until the server is totally up and running to add SSH and TCP port 8888 to it for ease of access. If you donot wait until the server is up along with the cluster setup. There is a huge risk of failure of the startup and implementation of EMR cluster Which was experienced several times and never ever clone your p[revious Cluster for which the permissions are already changed as it will directly terminate the startup process.



It is better you have a s3 or hdfs path specified so that you don't encounter these errors.



```
    print("No. of customers:",total_customers)
    return total_ratings,total_movies,total_customers

viewData(rating_data)
```

```
An error occurred while calling z:org.apache.spark.api.python.PythonRDD.collectAndServe.
: org.apache.hadoop.mapred.InvalidInputException: Input path does not exist: hdfs://ip-172-31-41-27.ec2.internal:8
20/home/notebook/work/TrainingRatings.txt
        at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:260)
        at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:208)
        at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:288)
        at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:204)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:251)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
        at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:49)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:251)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
        at org.apache.spark.api.python.PythonRDD.getPartitions(PythonRDD.scala:55)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:253)
        at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:251)
```

Although we have created the pyspark module itself in EMR cluster its better we recheck if the yarnID is created or not.

```
In [1]: import pyspark

Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
4	application_1589213110926_0005	pyspark	idle	Link	Link	✔

```
SparkSession available as 'spark'.

In [2]: # Loading the dataset
        from pyspark import SparkContext, SparkConf
        from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
        import sys

        sc = SparkContext()
        #Load the ratings file
        data = sc.textFile("/home/notebook/work/TestingRatings.txt.txt")
```

Do not run SparkContext during the execution of the runtime,e of a file as you will be observing this error.

```
Cannot run multiple SparkContexts at once; existing SparkContext(app=livy-session-7, master=yarn) created by __init
__ at /tmp/1094634487368643381:595
Traceback (most recent call last):
  File "/usr/lib/spark/python/lib/pyspark.zip/pyspark/context.py", line 133, in __init__
    SparkContext._ensure_initialized(self, gateway=gateway, conf=conf)
  File "/usr/lib/spark/python/lib/pyspark.zip/pyspark/context.py", line 340, in _ensure_initialized
    callsite.function, callsite.file, callsite.linenum))
ValueError: Cannot run multiple SparkContexts at once; existing SparkContext(app=livy-session-7, master=yarn) creat
ed by __init__ at /tmp/1094634487368643381:595

In [ ]: import argparse
        from pyspark import SparkContext

        sc = SparkContext()
```

This is how I have setup the EMR cluster and the basic checks before coding it.
Also remember that if you are using python 2 the **pyspark** command will not work and you need to use **./bin/spark-submit** for committing the code and also it is better we install python3

Documentation of approach – *how* did I approach the problem

The data has been preprocessed and now we need to find the similarity between them . I choose to use the scikit-surprise for performing this step. Therer are several measure but would like to discuss these of the below ones.

cosine	Compute the cosine similarity between all pairs of users (or items).

msd	Compute the Mean Squared Difference similarity between all pairs of users (or items).
pearson	Compute the Pearson correlation coefficient between all pairs of users (or items).

surprise.similarities.cosine()

Compute the cosine similarity between all pairs of users (or items).

Only **common** users (or items) are taken into account. The cosine similarity is defined as:

$$\text{cosine\_sim}(u, v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{vi}^2}}$$

or

$$\text{cosine\_sim}(i, j) = \frac{\sum_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum_{u \in U_{ij}} r_{uj}^2}}$$

surprise.similarities.msd()

Compute the Mean Squared Difference similarity between all pairs of users (or items).

Only **common** users (or items) are taken into account. The Mean Squared Difference is defined as:

$$\text{msd}(u, v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2$$

or

$$\text{msd}(i, j) = \frac{1}{|U_{ij}|} \cdot \sum_{u \in U_{ij}} (r_{ui} - r_{uj})^2$$

surprise.similarities.pearson()

Compute the Pearson correlation coefficient between all pairs of users (or items).

Only **common** users (or items) are taken into account. The Pearson correlation coefficient can be seen as a mean-centered cosine similarity, and is defined as:

$$\text{pearson\_sim}(u, v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

or

$$\text{pearson\_sim}(i, j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i) \cdot (r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \mu_j)^2}}$$

So I have applied the Pearsons'R correlation to determine the 10 most similar items and below is the output.

```
[22] recommend("What the #$*! Do We Know!?", 0)
```

```
For movie (What the #$*! Do We Know!?)
- Top 10 movies recommended based on Pearsons'R correlation -
/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar)
/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py:2455: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
 PearsonR                                        Name  count      mean
 1.000000                What the #$*! Do We Know!?   2831  3.055104
 1.000000                                Hockey Mom      3  2.000000
 0.943900                                Korn: Deuce     45  3.222222
 0.940354                          A Change of Place     57  2.719298
 0.937972                   Mo'Nique: One Night Stand     67  3.223881
 0.929670        The Guess Who: Running Back Thru Canada     30  2.500000
 0.923381  The Collected Shorts of Jan Svankmajer: Vol. 2     16  3.125000
 0.918559                Superstars of '70s Soul Live     31  3.129032
 0.902599                                       Masti     47  2.531915
 0.899559                     The Beauty of Ireland     34  2.676471
```

I have cross checked myself by trying to recommend a movie which has the pearson coefficient of 1 i.e the highjest similar items. And below are the outputs for that of item item based ones using pearson.

```
[23] recommend("A Change of Place", 0)
```

```
For movie (A Change of Place)
- Top 10 movies recommended based on Pearsons'R correlation -
/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar)
/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py:2455: RuntimeWarning: divide by zero encountered in true_
  c *= np.true_divide(1, fact)
 PearsonR                                        Name  count      mean
      1.0                                Cafe au Lait     33  2.757576
      1.0                             Heaven Help Us     151  3.198675
      1.0                                 Dallas 362     100  2.310000
      1.0                                  Ghost Rig      81  2.358025
      1.0                     Eric Idle's Personal Best      25  2.320000
      1.0  The Collected Shorts of Jan Svankmajer: Vol. 2      16  3.125000
      1.0          Eminem: The Slim Shady Show Uncut      49  2.244898
      1.0                           A Loving Father      48  2.812500
      1.0                                 Real Life     253  3.264822
      1.0                          Witness to a Kill      53  2.264151
```

Here we can observe that hockey mom and the 1st movies recommendations are similar

```
recommend("Hockey Mom", 0)
```

```
For movie (Hockey Mom)
- Top 10 movies recommended based on Pearsons'R correlation -
/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar)
/usr/local/lib/python3.6/dist-packages/numpy/lib/function_base.py:2455: RuntimeWarning: divide by zero encountered in true_d
  c *= np.true_divide(1, fact)
 PearsonR                                Name  count      mean
      1.0          What the #$*! Do We Know!?   2831  3.055104
      1.0                      Vegas Vacation   6150  3.219512
      1.0                           Dorm Daze   1146  2.526178
      1.0                           Boomerang   7631  3.024112
      1.0              The Sopranos: Season 3   9579  4.410690
      1.0                      One Hour Photo  18298  3.013062
      1.0                  Jeepers Creepers 2   6721  2.986163
      1.0               The Truth About Love    146  2.630137
      1.0              Incident at Loch Ness    356  2.660112
      1.0  Dora the Explorer: Catch the Stars    410  3.707317
```

It depends on whether the user has specific genre intrests than B is choosen .If there is no specific interst in genre then A is choosen based on user .

(A) average overlap of items rated by the users in the training set for users in the test set or

(B) average overlap of users that rated items in the training set for items appearing in the test set

Below are the movies that are rated by a user with overlap.

```
df_1205593 = df_1205593.set_index('MovieID')
df_1205593 = df_1205593.join(df_title)['Name']
print(df_1205593)
```

```
MovieID
8                                   What the #$*! Do We Know!?
443                                      Rabbit-Proof Fence
636                                              Stir Crazy
681                          Police Academy 3: Back in Training
850                          Stoked: The Rise and Fall of Gator
1123                            Dragon: The Bruce Lee Story
1901                           Cheech & Chong's Up in Smoke
2375                                          Fletch Lives
2512                                          Trees Lounge
2755                                     The Longest Yard
2905                                             Croupier
3151                                    Napoleon Dynamite
3274                                          Half Baked
3541                           History of the World: Part 1
3893                             The People vs. Larry Flynt
3943                                             Hitman
4432                                      The Italian Job
4761                                    The Cannonball Run
4849                                                Taps
6336                        The Crying Game: Collector's Edition
6912                          The Taking of Pelham One Two Three
8107                               Predator: Collector's Edition
8596                                               Seven
9481                                         Deliverance
10109                                        Major League II
10947                                     The Incredibles
11673                                           Drumline
12280                                          The Jerk
13015                        The Lost Boys: Special Edition
13328                                Death and the Maiden
14096                                     Step Into Liquid
14459                                          Uncle Buck
14643                          3: The Dale Earnhardt Story
16606                                               Fear
17358                                               Narc
17725    Jerry Seinfeld: I'm Telling You for the Last Time
```

I have run several Models for checking the RMSE and the k pair with the below specified models.

Algorithm	test_rmse	fit_time	test_time
SVD	0.998301	4.213025	0.241482
KNNBaseline	1.077418	0.201730	0.532709
KNNWithMeans	1.088175	0.040668	0.509448
KNNWithZScore	1.089639	0.073683	0.553372
KNNBasic	1.178040	0.028085	0.506293

Found out that SVD has the best rmse and then comes KNN . Trained a SVD model with grid search and then trained KNN with means along with the cosine and msd similarity. That is when my system and server crashed.

So couldn’t predict with the kmeans along with cosine and msd similarity will be putting the code in github if needed can be tried with larger instances.

Problems encountered:

There are several errors that I have encountered with the spark or either with the hdfs errors and also the RDD errors. Have tried discussing some and update the rest in giothub with screenshots.

Conclusion:

Couldn’t predict with kmeans but if it was computationally possible wou;ld have surely presented best results than in the table shown on the top.

Disclaimer:

All the above shown images of the errors are clearly executed by me .
All the outputs that are shown are run by me with help of defining functions from online and the scikit surprise library.
The Similarity definitions were taken from Surprise library.