A Recommender System Using Netflix Prize Dataset

Vineeth Reddy Cherukupally (U24689350), cherukupally@mail.usf.edu

Approach:
We considered the collaborative filtering technique to build a recommender system. So, we explore different types of collaborative filtering techniques like the user based, item based, model based and analysed the prediction of each approach using the RMSE value.

Goal:
To implement and find the better recommender system that predicts the better set of movies to be recommended. Here we evaluate the performance of each system based on the correlation methods/models used in it, by finding the mean RMSE of each approach and getting to know about the best among them and when to use the approaches based on their performance and accuracy.

Dataset:
The dataset used in our project is a subset of the Netflix Competition dataset. The features of our dataset are movie_id, user_id, rating, movie title. The dataset consists of 1821 movies being rated by 28978 users in the scale of 1.0 - 5.0. The training set consists of 3.25 million instances and the testing set consists of 100,000 instances.

Processing the data
Data loading:
In order to make the data easy to access, we brought all the instances from the CSV file into the data frame in the program.

Data viewing:
This helps to have a better understanding of the distribution of the movies in terms of ratings.

Data cleaning:
We grouped all the instances in terms of the user_id and we sorted them for making the operations easy.

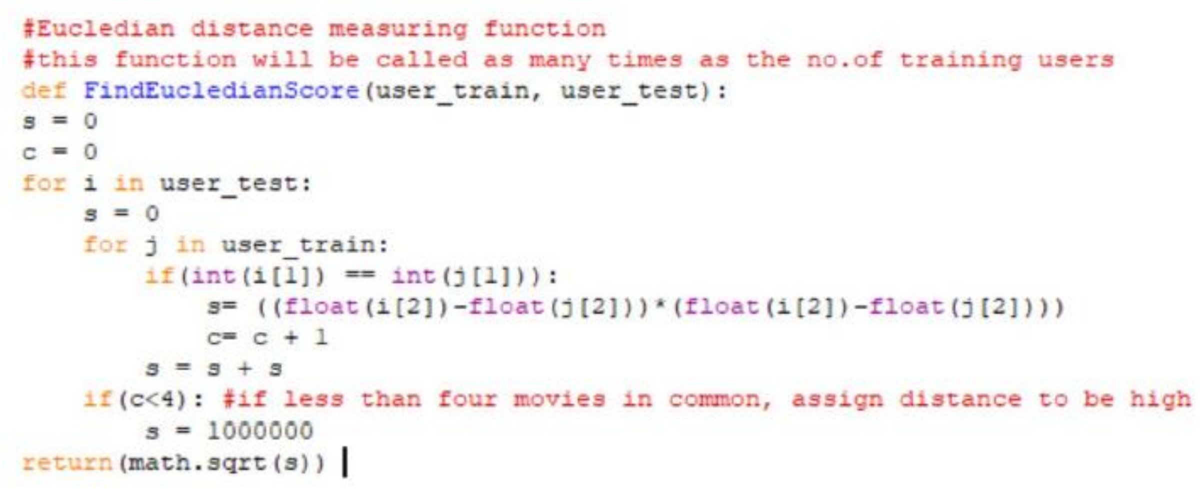
Data slicing:
We excluded all the instances of users who have rated very less movies and those instance for movies with very less ratings.

Data mapping:
We mapped the corresponding movie title with the movie_id to do the recommendations with movie title.

Building a recommender system
After processing the data, we found the correlation between all the users in the training

set and based on the calculated similarity values, we predict the ratings for the unseen movies of the test user and printed(recommended) the top 10 movies along with their predicted rating.

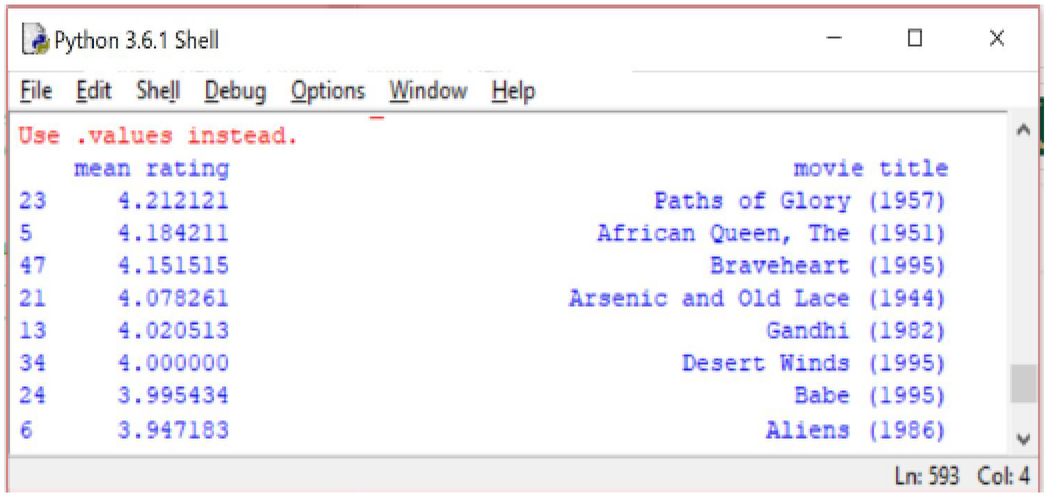
Euclidean distance metric (EDM):
The code snippet in the Fig 1 shows the implementation of euclidean distance measurement in our code. Here for the test user we found the Euclidean distance with all the user in the training set. From the calculated value we sort out the first 10 users who has the closer relationship with the test user for movie recommendation.



```
#Eucledian distance measuring function
#this function will be called as many times as the no.of training users
def FindEucledianScore(user_train, user_test):
s = 0
c = 0
for i in user_test:
     s = 0
     for j in user_train:
          if(int(i[1]) == int(j[1])):
               s= ((float(i[2])-float(j[2]))*(float(i[2])-float(j[2])))
               c= c + 1
          s = s + s
     if(c<4): #if less than four movies in common, assign distance to be high
          s = 1000000
return(math.sqrt(s))
```

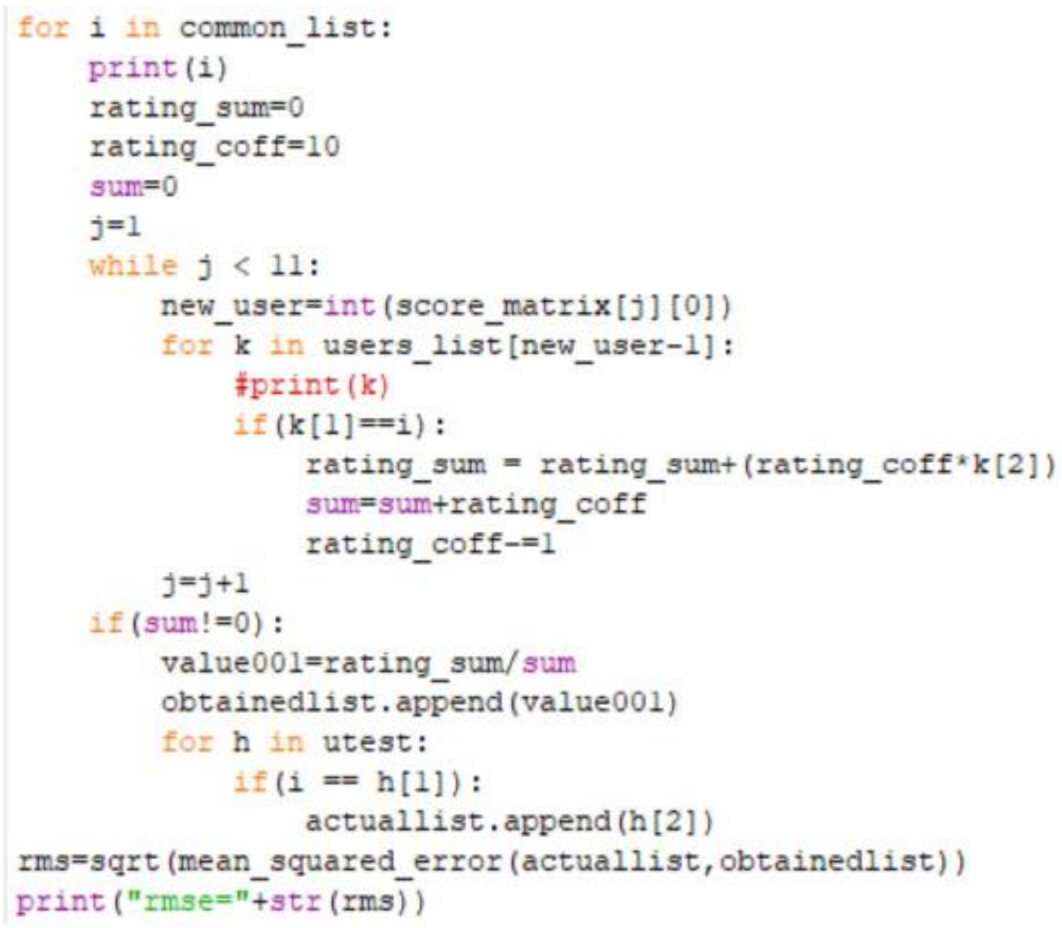
Fig 1: Euclidean distance calculation

Movies recommended for a user in the test set is shown in Fig 2. First, we took the movies that are not rated(seen) by the test user and at the same time it has to be rated(seen) by the users who have high correlation with the test user. For the obtained movies we found the predicted ratings by taking the weighted mean with respect to the euclidean distances. The movies which have the high predicted ratings are recommended to the users. Fig 3 shows how we calculated RMSE for the EDM recommender system. We found the predicted ratings for movies which are in common with the test user and his nearest neighbours. From the obtained ratings and the actual ratings for the movies in the list, we calculated the RMSE. We did this process for n (n=50) users and found the average RMSE value to be 0.9772 for our model.



```
Use .values instead.
    mean rating                               movie title
23     4.212121                      Paths of Glory (1957)
5      4.184211                 African Queen, The (1951)
47     4.151515                          Braveheart (1995)
21     4.078261                Arsenic and Old Lace (1944)
13     4.020513                              Gandhi (1982)
34     4.000000                        Desert Winds (1995)
24     3.995434                                Babe (1995)
6      3.947183                              Aliens (1986)
```

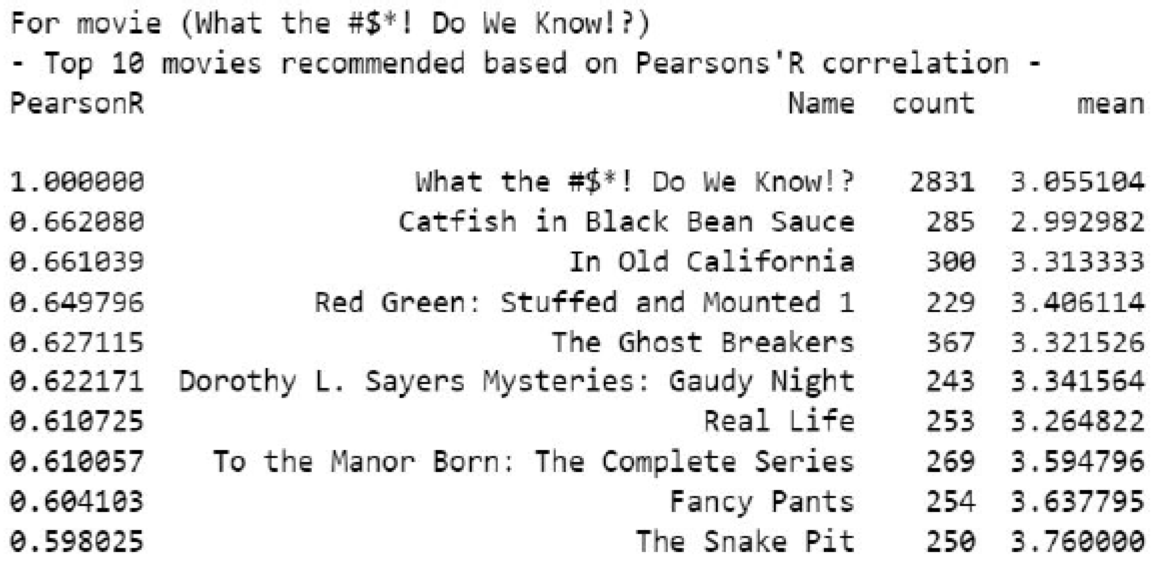
Fig 2: Movies recommended based on EDM



```
for i in common_list:
    print(i)
    rating_sum=0
    rating_coff=10
    sum=0
    j=1
    while j < 11:
        new_user=int(score_matrix[j][0])
        for k in users_list[new_user-1]:
            #print(k)
            if(k[1]==i):
                rating_sum = rating_sum+(rating_coff*k[2])
                sum=sum+rating_coff
                rating_coff-=1
        j=j+1
    if(sum!=0):
        value001=rating_sum/sum
        obtainedlist.append(value001)
        for h in utest:
            if(i == h[1]):
                actuallist.append(h[2])
rms=sqrt(mean_squared_error(actuallist,obtainedlist))
print("rmse="+str(rms))
```

Fig 3: RMSE calculation for EDM

Pearson’s correlation metric:
We tried the Pearson’s correlation metric in the hope of getting a better RMSE value compared to the EDM. Unlike finding the correlation between the users, here we did find correlation between movies. Correlation between two movies can be found by measuring the mean ratings given by all the users for these movies and finding its deviation from the mean. Two movies said to have high correlation if both the movies have the same deviation from the mean. Fig 4 shows the recommendation of movies for a given movie using the pearson’s correlation. In this approach we got the average RMSE value to be 0.9572 which is better than EDM. We found that these approaches (Pearson’s, EDM) to have high performance.

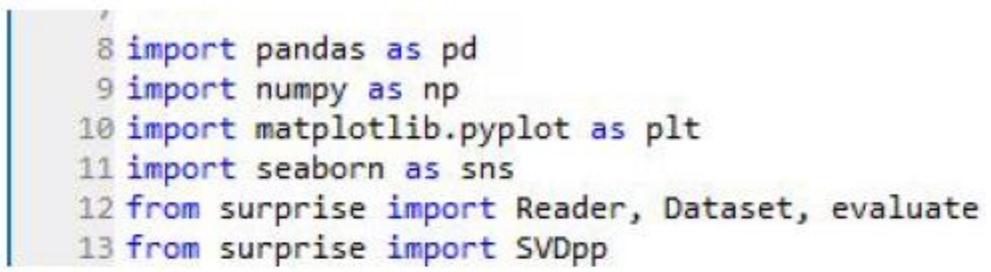


```
For movie (What the #$*! Do We Know!?)
- Top 10 movies recommended based on Pearsons'R correlation -
PearsonR                                    Name  count      mean

1.000000                 What the #$*! Do We Know!?   2831  3.055104
0.662080                 Catfish in Black Bean Sauce    285  2.992982
0.661039                         In Old California    300  3.313333
0.649796           Red Green: Stuffed and Mounted 1    229  3.406114
0.627115                          The Ghost Breakers    367  3.321526
0.622171  Dorothy L. Sayers Mysteries: Gaudy Night    243  3.341564
0.610725                                   Real Life    253  3.264822
0.610057    To the Manor Born: The Complete Series    269  3.594796
0.604103                                 Fancy Pants    254  3.637795
0.598025                               The Snake Pit    250  3.760000
```

Fig 4: Recommendations obtained from pearson’s correlation

Analysis of different model-based algorithms for collaborative filtering:

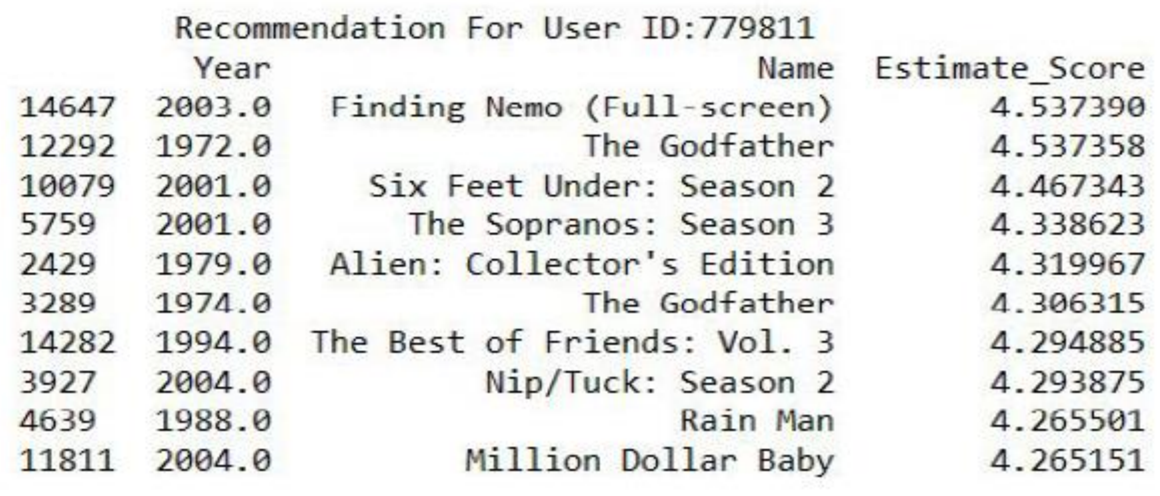
We have done an analysis of the following algorithms:
➢ Non-Negative Matrix Factorization (NMF)
➢ Singular Value Decomposition (SVD)
➢ Probabilistic Matrix Factorization (PMF)
➢ Singular Value Decomposition ++ (SVD++)
For implementing the above algorithms we have used the package named **“surprise”** which is a python scikit for building recommender systems.



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from surprise import Reader, Dataset, evaluate
from surprise import SVDpp
```

Fig 5: Packages used for the model-based algorithms

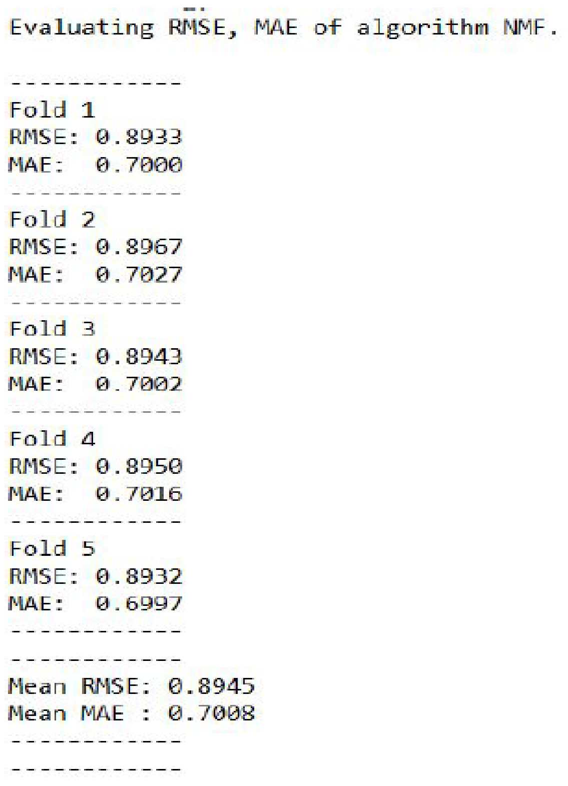
1. Non-Negative Matrix Factorization (NMF):
For the purpose of comparative analysis of the different algorithms used in our project we shown one user from the testing set as an example. We can also demonstrate the recommendations for a group of users. For example, we predict the user ratings for the user with user ID = 779811 using the NMF method.



```
        Recommendation For User ID:779811
         Year                          Name  Estimate_Score
14647  2003.0  Finding Nemo (Full-screen)        4.537390
12292  1972.0                 The Godfather        4.537358
10079  2001.0     Six Feet Under: Season 2        4.467343
5759   2001.0       The Sopranos: Season 3        4.338623
2429   1979.0   Alien: Collector's Edition        4.319967
3289   1974.0                 The Godfather        4.306315
14282  1994.0  The Best of Friends: Vol. 3        4.294885
3927   2004.0          Nip/Tuck: Season 2        4.293875
4639   1988.0                      Rain Man        4.265501
11811  2004.0           Million Dollar Baby        4.265151
```

Fig 6: Predictions using Non-negative matrix factorization

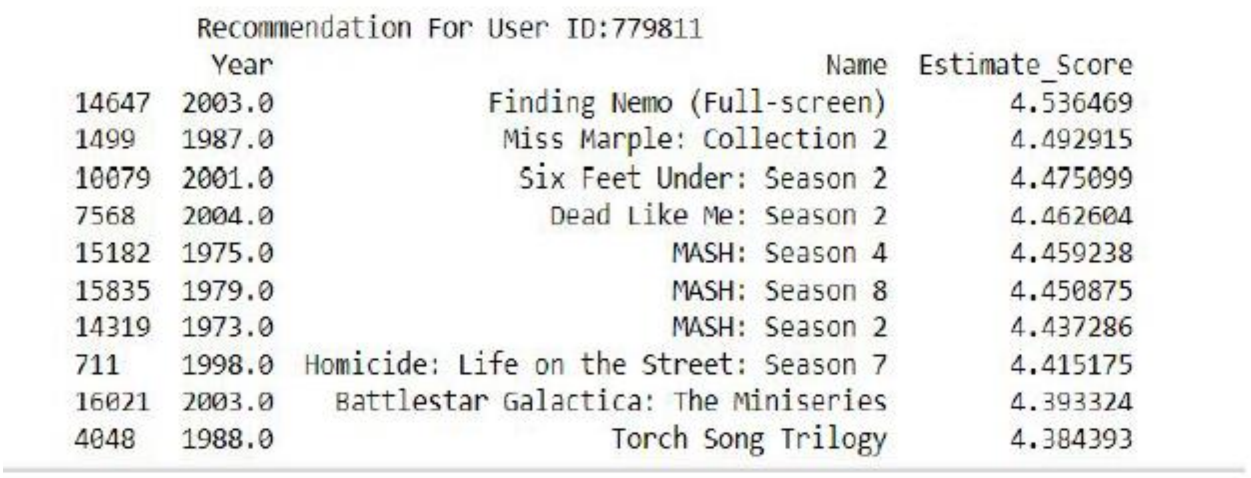
The RMSE values when calculated for the NMF algorithm for a group of test users are obtained as follows:



```
Evaluating RMSE, MAE of algorithm NMF.

------------
Fold 1
RMSE: 0.8933
MAE:  0.7000
------------
Fold 2
RMSE: 0.8967
MAE:  0.7027
------------
Fold 3
RMSE: 0.8943
MAE:  0.7002
------------
Fold 4
RMSE: 0.8950
MAE:  0.7016
------------
Fold 5
RMSE: 0.8932
MAE:  0.6997
------------
------------
Mean RMSE: 0.8945
Mean MAE : 0.7008
------------
------------
```

Fig 7: RMSE values for the Non-Negative Matrix Factorization Algorithm

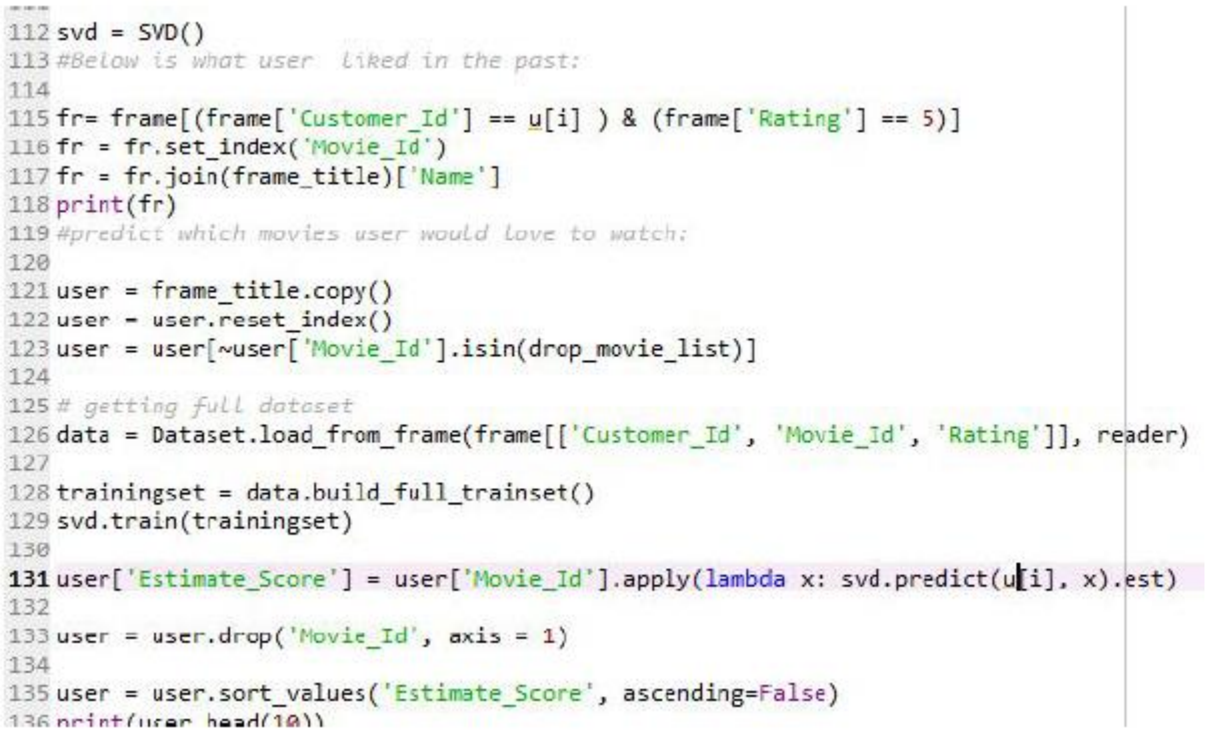
2. Singular Value Decomposition (SVD):
Here, we are predicting the user ratings for the same user with user ID = 779811 using the SVD method.



```
       Recommendation For User ID:779811
        Year                                 Name  Estimate_Score
14647  2003.0            Finding Nemo (Full-screen)        4.536469
1499   1987.0             Miss Marple: Collection 2        4.492915
10079  2001.0              Six Feet Under: Season 2        4.475099
7568   2004.0                Dead Like Me: Season 2        4.462604
15182  1975.0                        MASH: Season 4        4.459238
15835  1979.0                        MASH: Season 8        4.450875
14319  1973.0                        MASH: Season 2        4.437286
711    1998.0  Homicide: Life on the Street: Season 7        4.415175
16021  2003.0     Battlestar Galactica: The Miniseries        4.393324
4048   1988.0                    Torch Song Trilogy        4.384393
```

Fig 8: Predictions using Singular Value Decomposition

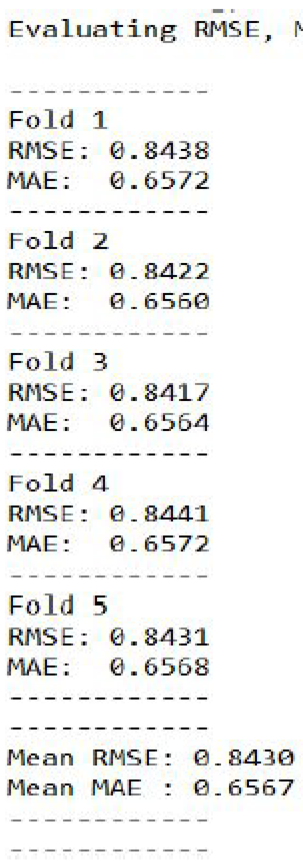
The SVD algorithm can be implemented using the following procedure, similarly the other algorithms can be implemented. The code for the PMF algorithm is implemented by using the **'svd()'** method which is available from the python package. To predict the rating for the used, **svd.predict()** is used.



```
svd = SVD()
#Below is what user  liked in the past:

fr= frame[(frame['Customer_Id'] == u[i] ) & (frame['Rating'] == 5)]
fr = fr.set_index('Movie_Id')
fr = fr.join(frame_title)['Name']
print(fr)
#predict which movies user would love to watch:

user = frame_title.copy()
user = user.reset_index()
user = user[~user['Movie_Id'].isin(drop_movie_list)]

# getting full dataset
data = Dataset.load_from_frame(frame[['Customer_Id', 'Movie_Id', 'Rating']], reader)

trainingset = data.build_full_trainset()
svd.train(trainingset)

user['Estimate_Score'] = user['Movie_Id'].apply(lambda x: svd.predict(u[i], x).est)

user = user.drop('Movie_Id', axis = 1)

user = user.sort_values('Estimate_Score', ascending=False)
print(user.head(10))
```

Fig 9: Code Snippet for implementing the SVD algorithm

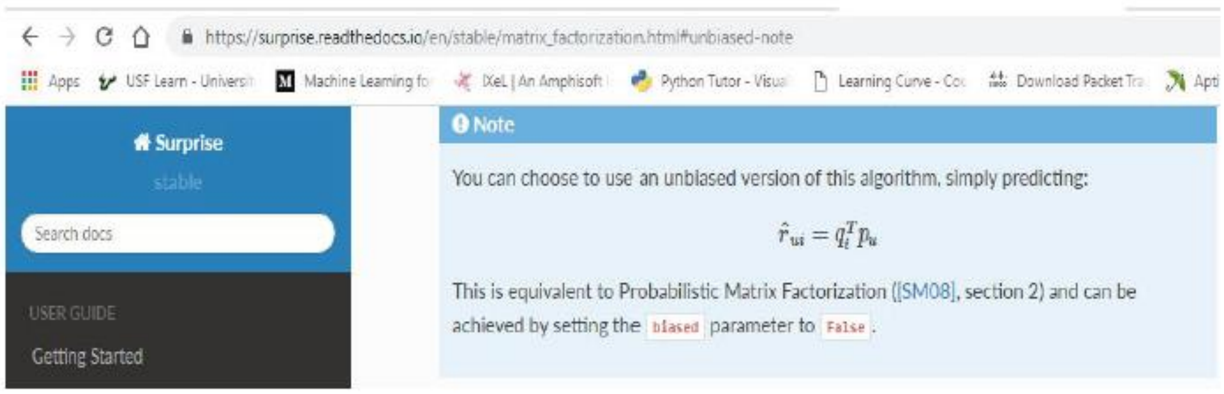
We iterated through the test users stored in the array of 'u [ ]' and printed the recommendations for them, using the 'svd.predict' method. The RMSE values when calculated for the SVD algorithm for a group of test users are obtained as follows:



```
Evaluating RMSE, MAE of algorithm SVD.

------------
Fold 1
RMSE: 0.8438
MAE:  0.6572
------------
Fold 2
RMSE: 0.8422
MAE:  0.6560
------------
Fold 3
RMSE: 0.8417
MAE:  0.6564
------------
Fold 4
RMSE: 0.8441
MAE:  0.6572
------------
Fold 5
RMSE: 0.8431
MAE:  0.6568
------------
------------
Mean RMSE: 0.8430
Mean MAE : 0.6567
------------
------------
```

Fig 10: RMSE values for the Singular Value Decomposition Algorithm

3. Probabilistic Matrix Factorization (PMF):
Probabilistic Matrix Factorization algorithm is implemented by putting the parameter, **biased ='false'** in the SVD( ) method of the scikit-package. The reference along with the link for this is shown below.



Note

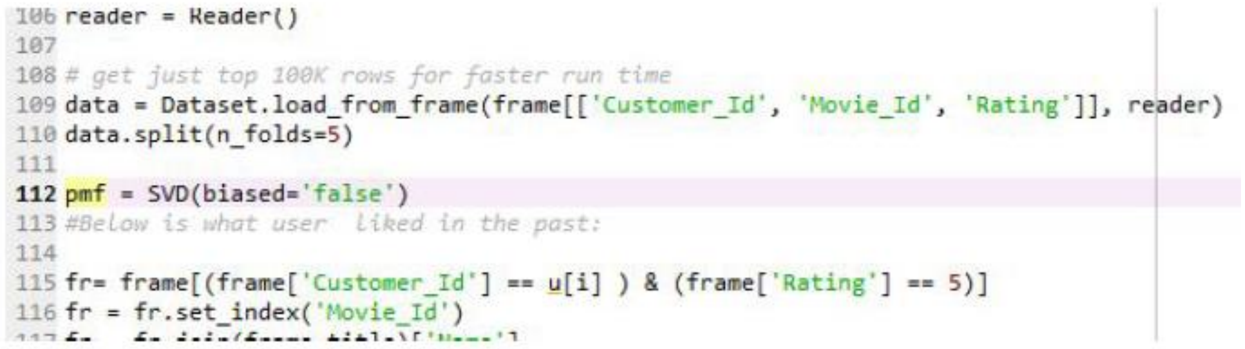
You can choose to use an unbiased version of this algorithm, simply predicting:

$$\hat{r}_{ui} = q_i^T p_u$$

This is equivalent to Probabilistic Matrix Factorization ([SM08], section 2) and can be achieved by setting the `biased` parameter to `False`.

Fig 11: Procedure to implement PMF algorithm

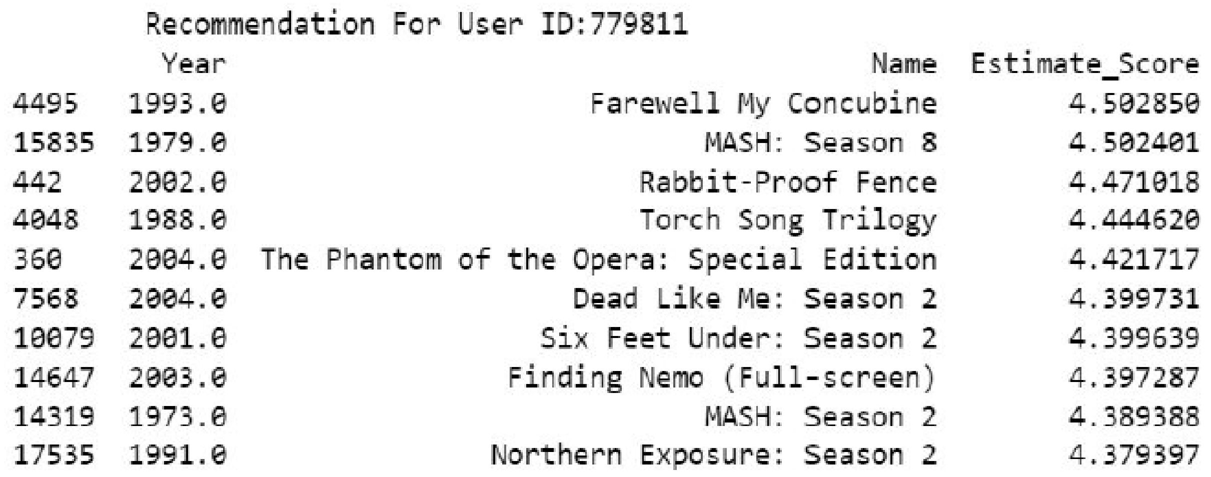
Implementing PMF:



```
reader = Reader()

# get just top 100K rows for faster run time
data = Dataset.load_from_frame(frame[['Customer_Id', 'Movie_Id', 'Rating']], reader)
data.split(n_folds=5)

pmf = SVD(biased='false')
#Below is what user  liked in the past:

fr= frame[(frame['Customer_Id'] == u[i] ) & (frame['Rating'] == 5)]
fr = fr.set_index('Movie_Id')
```

Fig 12: Code Snippet for implementing the PMF algorithm

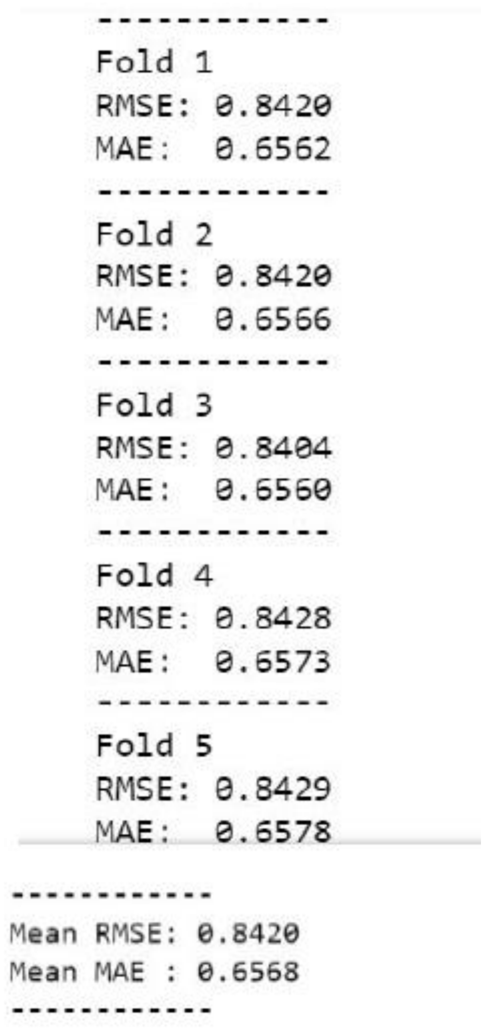
The code for the PMF algorithm is implemented by using the **'svd(biased='false')'** method which is available from the python package. To predict the rating for the used, **svd.predict()** is used.
The recommendation for the user with ID:779811 is shown below:



```
       Recommendation For User ID:779811
        Year                                    Name  Estimate_Score
4495   1993.0                   Farewell My Concubine        4.502850
15835  1979.0                          MASH: Season 8        4.502401
442    2002.0                      Rabbit-Proof Fence        4.471018
4048   1988.0                      Torch Song Trilogy        4.444620
360    2004.0  The Phantom of the Opera: Special Edition     4.421717
7568   2004.0                    Dead Like Me: Season 2      4.399731
10079  2001.0                  Six Feet Under: Season 2      4.399639
14647  2003.0                Finding Nemo (Full-screen)      4.397287
14319  1973.0                          MASH: Season 2        4.389388
17535  1991.0              Northern Exposure: Season 2       4.379397
```

Fig 13: Predictions using the PMF Algorithm

As shown above, since the RMSE values are almost similar for both SVD and PMF algorithms there are some common recommendations for the same user in the both SVD and PMF outputs.
The RMSE values for the PMF algorithm are obtained as shown below:

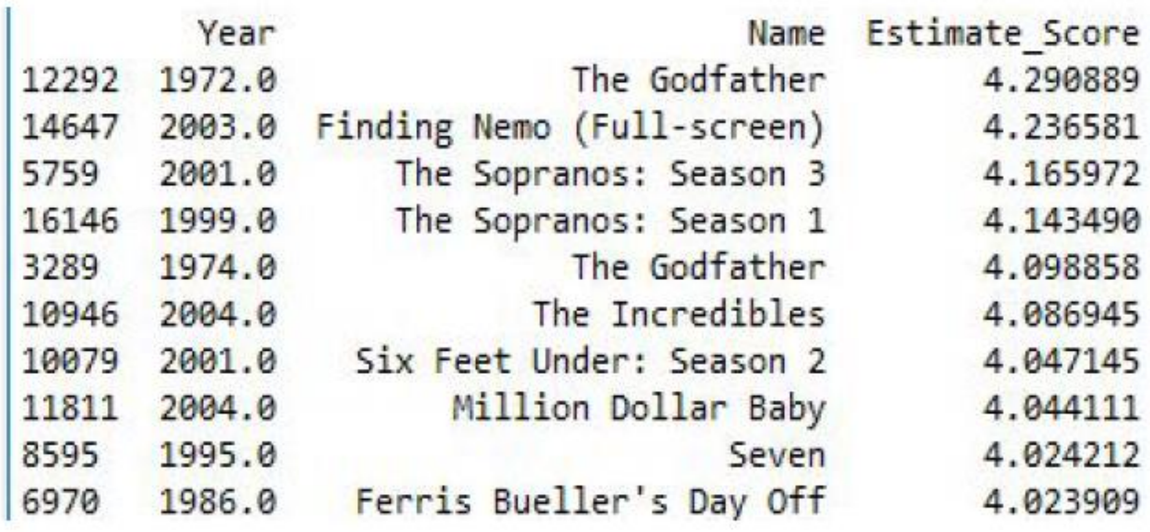


```
------------
Fold 1
RMSE: 0.8420
MAE:  0.6562
------------
Fold 2
RMSE: 0.8420
MAE:  0.6566
------------
Fold 3
RMSE: 0.8404
MAE:  0.6560
------------
Fold 4
RMSE: 0.8428
MAE:  0.6573
------------
Fold 5
RMSE: 0.8429
MAE:  0.6578
```

```
-----------
Mean RMSE: 0.8420
Mean MAE : 0.6568
-----------
```

Fig 14: RMSE values for the PMF Algorithm

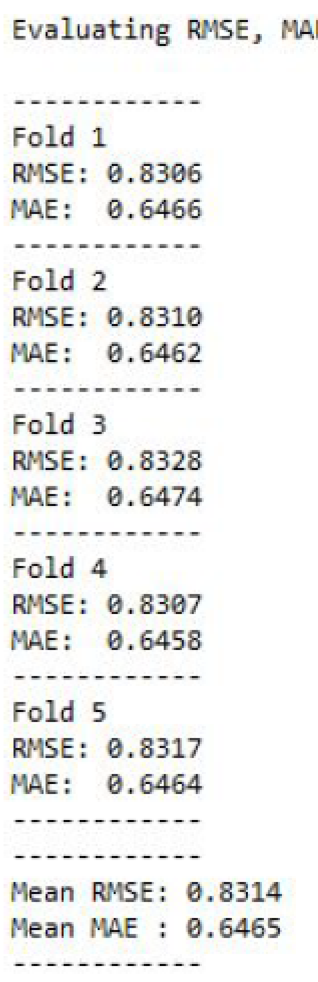
4. Singular Value Decomposition ++ (SVDpp):
The *SVD++* algorithm, is an extension of *SVD* will also take implicit ratings into account. The code for the SVD++ algorithm is implemented by using the svdpp() method which is available from the python package. To predict the rating for the used, svdpp.predict() is used.
The recommendation for the same user with ID:779811 is obtained as follows:



```
        Year                       Name  Estimate_Score
12292  1972.0              The Godfather        4.290889
14647  2003.0  Finding Nemo (Full-screen)        4.236581
5759   2001.0     The Sopranos: Season 3        4.165972
16146  1999.0     The Sopranos: Season 1        4.143490
3289   1974.0              The Godfather        4.098858
10946  2004.0            The Incredibles        4.086945
10079  2001.0    Six Feet Under: Season 2        4.047145
11811  2004.0          Million Dollar Baby        4.044111
8595   1995.0                      Seven        4.024212
6970   1986.0    Ferris Bueller's Day Off        4.023909
```

Fig 15: Predictions using the Singular Value Decomposition ++ Algorithm

The RMSE values for a set of test users using a 5-fold cross validation on SVD++ is obtained as follows:



```
Evaluating RMSE, MAE of algorithm SVDpp.

------------
Fold 1
RMSE: 0.8306
MAE:  0.6466
------------
Fold 2
RMSE: 0.8310
MAE:  0.6462
------------
Fold 3
RMSE: 0.8328
MAE:  0.6474
------------
Fold 4
RMSE: 0.8307
MAE:  0.6458
------------
Fold 5
RMSE: 0.8317
MAE:  0.6464
------------
------------
Mean RMSE: 0.8314
Mean MAE : 0.6465
------------
```

Fig 16: RMSE values for the SVD++ Algorithm

RMSE COMPARISON

We have used RMSE as evaluation metric for evaluating the different approach we implemented here for recommendation system. On comparing first two memory based techniques, Euclidean Distance and Pearson correlation, we can see that Pearson has better RMSE than Euclidean as shown in Fig 15 below.

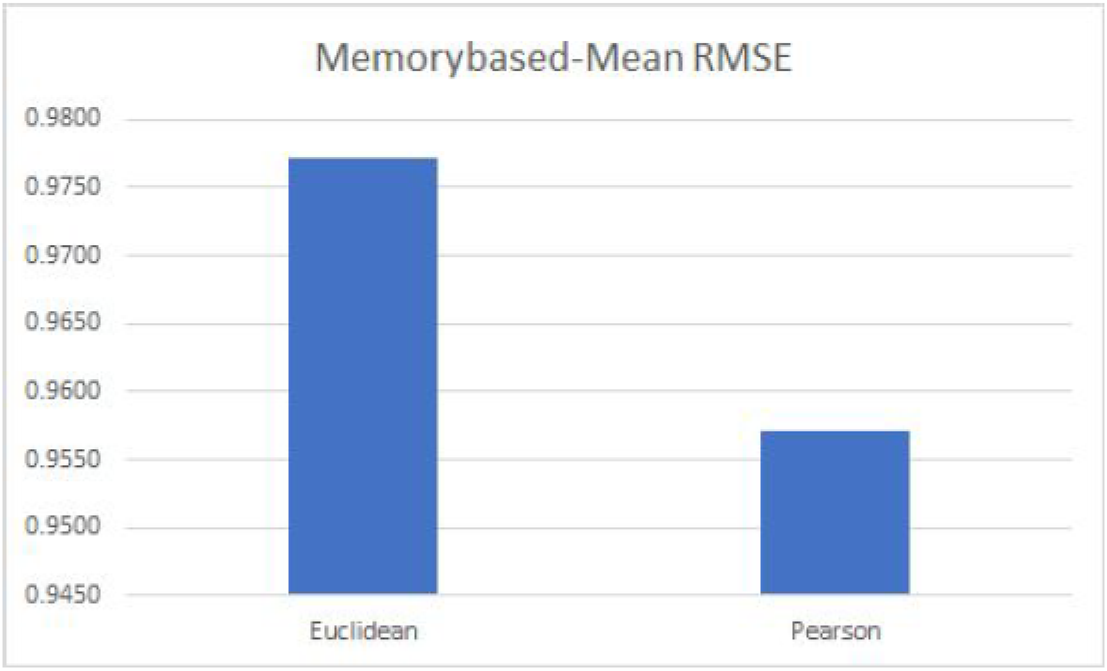


Fig 17: Memory based techniques Mean RMSE comparison

We have also implemented four Model based technique and evaluated their prediction using RMSE. While comparing those models' prediction using RMSE, we can find that SVD++
has better RMSE than the other models considered as shown in Fig 16 below, it is also better than both memory-based techniques implemented.

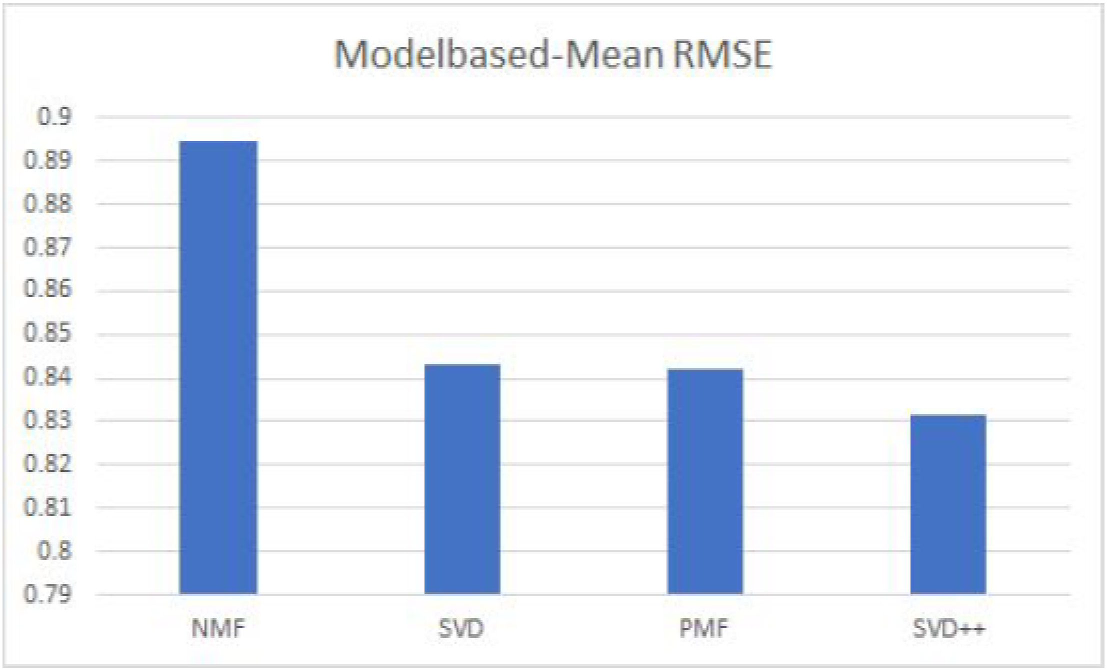


Fig 18: Model based techniques Mean RMSE comparison

Discussion:
From the obtained RMSE value for the approaches, we figured out that the user based recommender systems have got high RMSE values whereas the model-based technique has got better RMSE. Even though model-based techniques especially SVD++ has better prediction value they are slow in terms of performance where memory-based techniques are better, like pearson has considerably good RMSE with good performance. So, we suggest that if performance is primary concern then we could use user/ item-based approach (like Pearson's correlation technique), if prediction accuracy is primary concern then we could use user/item-based approach (like SVD++). For future work we can consider combination of two or more models where the prediction accuracy can be improved considerably without compromising on performance.

References

- https://www.kaggle.com/laowingkin/netflix-movie-recommendation/noteboo k
- https://github.com/JieniChen/Recommender-System
- https://surprise.readthedocs.io/en/latest/matrix_factorization.html
- https://surprise.readthedocs.io/en/stable/matrix_factorization.html#unbiased-note