BITS - Pilani, Hyderabad Campus

CS F469 IR Assignment – 3

Recommender Systems

Team Members:

1. Kushagra Gupta	2018A7PS0208H
2. Parveen	2018A7PS0623H
3. S. Sai Naga Shashank	2018AAPS0347H

- Dataset Used – MovieLens Dataset
  It contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000

- Language Used – Python
- Libraries Used – Numpy

- Dataset is divided into 60-40 training and testing, only for the collaborative filtering.
- Training and Test data is represented using a Matrix (Users * Movies)

Recommender System Technique	RMSE	Precision on top K	Spearman Rank Correlation	Time taken for prediction
Collaborative	1.1539	0.3457	0.9999	255 seconds
Collaborative along with Baseline approach	1.2013	0.3283	0.9999	255 seconds
SVD	0.7030	0.8439	0.9999	4.55 seconds
SVD with 90% retained energy	1.1351	0.6916	0.9999	3.18 seconds
CUR	3.7316	0.2104	0.9999	6.80 seconds
CUR with 90% retained energy	3.7399	0.2175	0.9999	5.95 seconds

Time taken is of all Test queries, not for a single query

Collaborative Filtering

- Item – Item collaborative filtering used
- Firstly, find top K items similar to the item which we are predicting for the user, which the user has also previously rated.
- K used = 5
- Similarity measure used between items: Pearson Similarity

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

- Strict and easy raters are handled properly, because the rating predicted for the user only depends on the ratings it gave to other items, not on the other users, which will be the case if User- User filtering would be used.

- Finally, using the top K ratings of the user for the similar items, prediction is made using the below formula.

$$\boxed{r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}}$$

s_{ij} - similarity of items ***i*** and ***j***
r_{xj} - rating of user ***x*** on item ***j***
N(i;x) - items rated by user ***x*** similar to item ***i***

Collaborative along with Baseline approach

- Since in above mentioned collaborative filtering, similarity is found only for those items which user has previously rated, this became a problem while predicting for a new user who doesn't have rated much.
- Hence, global baseline approach is used where, along with the prediction made earlier, some new parameters are also added as show below:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$\boldsymbol{b_{xi} = \mu + b_x + b_i}$$

- $\boldsymbol{\mu}$ = overall mean movie rating
- $\boldsymbol{b_x}$ = rating deviation of user ***x***
  = *(avg. rating of user* ***x****)* $-\boldsymbol{\mu}$
- $\boldsymbol{b_i}$ = rating deviation of movie ***i***

Pre-processing has been done on the training dataset to reduce the query time on test data:

- Pearson similarities has been calculated for every pair of movies.

- $\boldsymbol{b_{xi}}$ has been calculated in advance for each user-movie pair

Singular Value Decomposition

- SVD is calculated for any given M x N matrix by decomposing that matrix into three component matrices defined as follows:
    - Matrix U: An M x r column orthonormal matrix where r is the rank of the original matrix.
    - Matrix ∑: An r x r diagonal matrix containing the singular values of the original matrix in decreasing order.
    - Matrix V: An N x r column orthonormal matrix where r is the rank of the original matrix.

- But since, the rank of the above used dataset has been found out to be approx. equal to the number of movies itself, this doesn’t help in reducing the dimension of the original matrix.
- Hence, we chose r = MOVIES / 3

- Using, $U*\sum*V^T$ we can get back the original matrix.

- This way using a smaller r we can reduce the dimensions significantly and can have a very good approximation of our original matrix as well.

SVD with 90% energy retain: Here we retain enough singular values, represented by the matrix ∑, such that it holds 90% energy of the whole SVD.

It means that the sum of the squares of the retained singular values should represent at least 90% of the total squared sum.

CUR Decomposition

- Similar to SVD, here also we decompose our utility matrix into 3 parts:
    - C – An M x k orthogonal matrix, where k is chosen to be r.log(r) in practice. [r – value used during SVD]
    - U – An k x k diagonal matrix containing singular values
    - R – An k x N orthogonal matrix
    - Approximation of original matrix = C * U * R

- C is found by randomly selecting k columns from the original matrix.
- Probability of selection of each column is given by ratio of sum of its squared element to sum of squared element of whole matrix.
- R is calculated in a similar fashion but we select rows instead of the columns.

- In most cases, CUR with no duplicates gives better result than with duplicates. But in this dataset, both are giving the same result, and surprisingly with duplicates is slightly better than the former. Hence, we moved with that only.

- U is calculated as the pseudo inverse of the SVD decomposition of the intersection matrix of selected rows and columns.

SVD with 90% energy retain: Similar to SVD, here also same approach has been taken to retain 90% of the total energy.

<u>Note:</u> Since the ratings in the given dataset are only within the range (0, 5) and the number of ratings are in Millions, therefore using a smaller K for calculating Precision for top K would not give not accurate results.

Hence, we have used top 20% as K. It can also be seen as finding precision for the predicted ratings with ratings > 4, because top 20% of the total ratings are greater than 4.

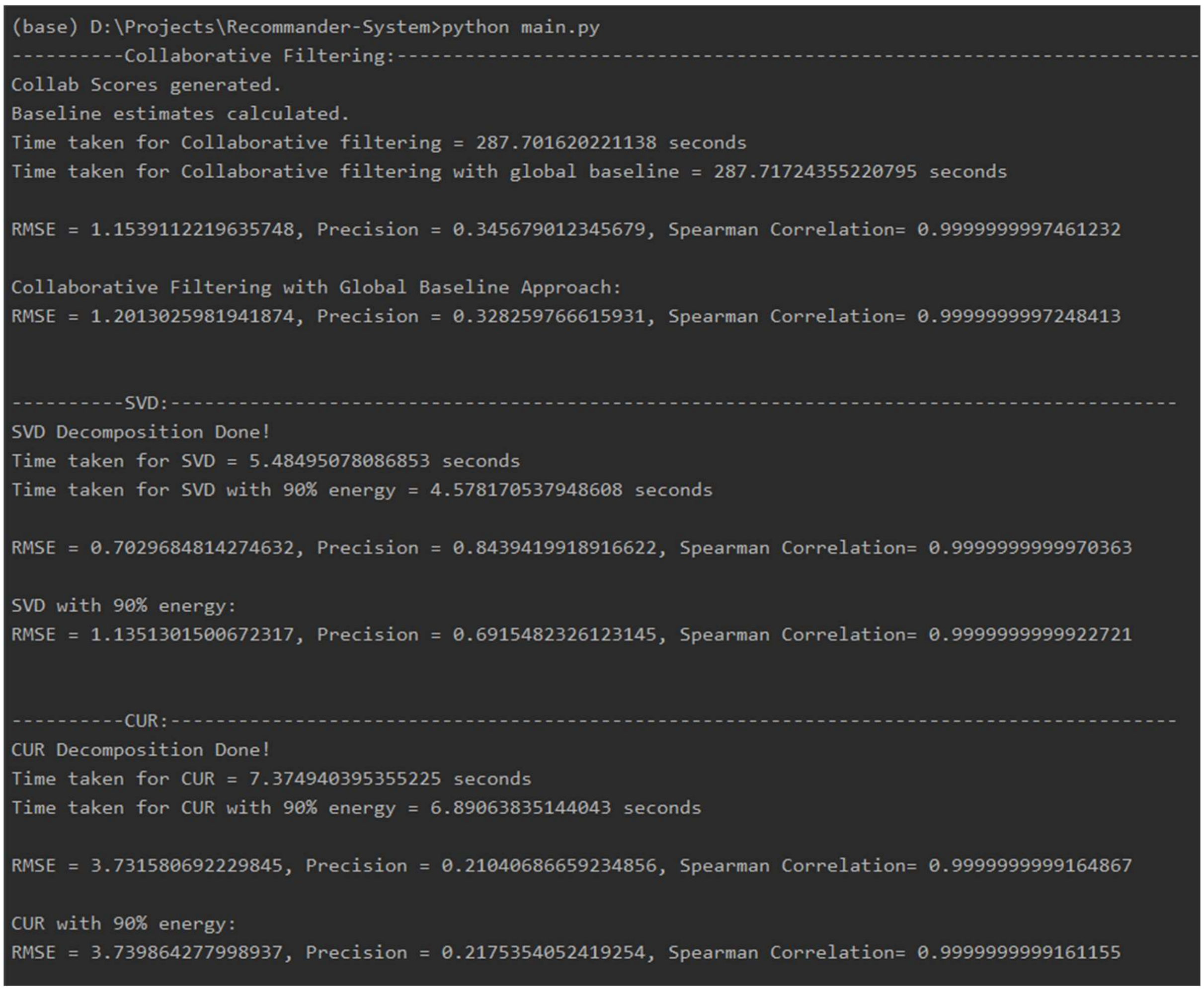
<u>Note</u>: In code part of SVD decomposition, we have directly used NumPy's SVD function. That's because of the fact that NumPy's inbuilt function is about 3 or 4 times faster than the method we have implemented, and it helped us in debugging the code faster.

If we have used our own implemented code, then it would take around half an hour for a single run, which was not practical for debugging purposes.

But to avoid any kind of ambiguity, we have not removed the code we have implemented, but instead commented that part, and it is fully working code.

Both the codes are giving the same result, but the difference being the efficiency only.

Results screenshot:



```
(base) D:\Projects\Recommander-System>python main.py
---------Collaborative Filtering:----------------------------------------------------------------------
Collab Scores generated.
Baseline estimates calculated.
Time taken for Collaborative filtering = 287.701620221138 seconds
Time taken for Collaborative filtering with global baseline = 287.71724355220795 seconds

RMSE = 1.1539112219635748, Precision = 0.345679012345679, Spearman Correlation= 0.9999999997461232

Collaborative Filtering with Global Baseline Approach:
RMSE = 1.2013025981941874, Precision = 0.328259766615931, Spearman Correlation= 0.9999999997248413


---------SVD:-----------------------------------------------------------------------------------------
SVD Decomposition Done!
Time taken for SVD = 5.48495078086853 seconds
Time taken for SVD with 90% energy = 4.578170537948608 seconds

RMSE = 0.7029684814274632, Precision = 0.8439419918916622, Spearman Correlation= 0.9999999999970363

SVD with 90% energy:
RMSE = 1.1351301500672317, Precision = 0.6915482326123145, Spearman Correlation= 0.9999999999922721


---------CUR:-----------------------------------------------------------------------------------------
CUR Decomposition Done!
Time taken for CUR = 7.374940395355225 seconds
Time taken for CUR with 90% energy = 6.89063835144043 seconds

RMSE = 3.731580692229845, Precision = 0.21040686659234856, Spearman Correlation= 0.9999999999164867

CUR with 90% energy:
RMSE = 3.739864277998937, Precision = 0.2175354052419254, Spearman Correlation= 0.9999999999161155
```