Project Report
on
Movie Recommendation system

Prem Kumar Kamasani, Manjusha Gadupudi

December 18, 2017

Introduction:

The data for this Project has been taken from grouplens.org.
The link for accessing the data is:
https://grouplens.org/datasets/movielens/100k/
This dataset consists of a 100,000 ratings on a scale of 1-5 from 943 users on 1682 movies. Each user has rated atleast 20 movies, a simple demographic information of the users such as age, gender, occupation and zip has been given along with movies information such as title, genre,ratings, timestamp.

Recommender systems are used to help users find new items or services based on information about the user.

The systems also play an important role in decision making helping users to maximize profits or minimize risks.

Today recommender systems are used in many information based companies such as Google, Twitter, Linkedin and Netflix

We tried to execute the code on our local machine and then used Stampede2 for greater efficiency and precision. We installed Anaconda python distribution onto stampede for the code execution.

Background:

This is a project that a many researchers are currently working on, though it is already in implementation, many algorithms are generated everyday to increase the efficiency of this system which is essential for the development of a model.

Many applications use a recommendation system or a variation of the basic idea has been implemented. Amazon, Netflix, Google, Facebook are just a few organizations that have implemented this system. A recommendation system provides the users with a direction in their search which leads them to be completely engaged in the application. It is a strategy that many organizations are implementing to increase their sales and the time spent on the website.

Algorithms available to solve this problem are listed below:

Case1:
Recommend most popular items- The most popular items are recommended for all the users. The major disadvantage of this type is that the recommendations are not personalized.

Case2:
Classifier for Recommendations- In this type of recommendation, the users are given various parameters which are classified to produce the outcome in 0s or 1s. If the outcome is a 1, the movie would be recommended to the user and if the outcome is a 0, it would not be recommended. An advantage here is it incorporates personalization and works even when the user history is short or unavailable, however the disadvantage here is that the features might not be available or even if they are they might not be sufficient to make a good classifier.As the number of users and items grow making a good classifier may become exponentially difficult.

Case3:
Collaborative Filtering Algorithm- In this type of filtering, if person A watches movie 1,2,3,4 and person B watches 3,4,5,6 person A will be recommended with movie 5,6 and person B will be recommended with 1,2. This is entirely based on past behaviour and not on the context.There are two types of collaborative filtering

User-User collaboration- This takes information about the users past history and finds look-alike customers. However implementing this is a bit challenging

as it requires a lot of information about each user, which takes a lot of time.
Item-Item collaboration- This is similar to the user-user collaboration, however instead of using user information it takes the item information.This takes item look-alike and recommends it to users. Implementing this is easier as all the item information is already stored in the database.
We have used Collaborative Filtering Algorithms in this project, as the disadvantages of this algorithms are very limited and the recommendations are personalized.
Flow of the Algorithm:
We first divided the data in 75:25 as test and train data.
We created a user-item Matrix on the entire dataset, and since we have train and test data, we will achieve 2 matrices of dimensions $943 * 1682$
We then found the similarity matrix where distance Matrix used in this system is cosine similarity. The similarity is calculated as angle between ratings vector.
Cosine similarity for user1 and 2 can be calculated using the formula below:

$$S_u^{cos}(U_k.U_1) = \frac{U_k.U_1}{||U_k|| \quad ||U_1||} = \frac{\sum X_{k,2}X_{a,m}}{\sqrt{\sum X_{a,2}^2 \quad \sum X_{1,2}^2}}$$

To calculate similarity between items m and b we used following formula:

$$S_u^{cos}(i_m.i_b) = \frac{i_m.i_b}{||i_m|| \quad ||i_b||} = \frac{\sum X_{a,m}X_{a,b}}{\sqrt{\sum X_{a,m}^2 \quad \sum X_{a,b}^2}}$$

Next step is to make predictions. Following formula is used to make predictions:

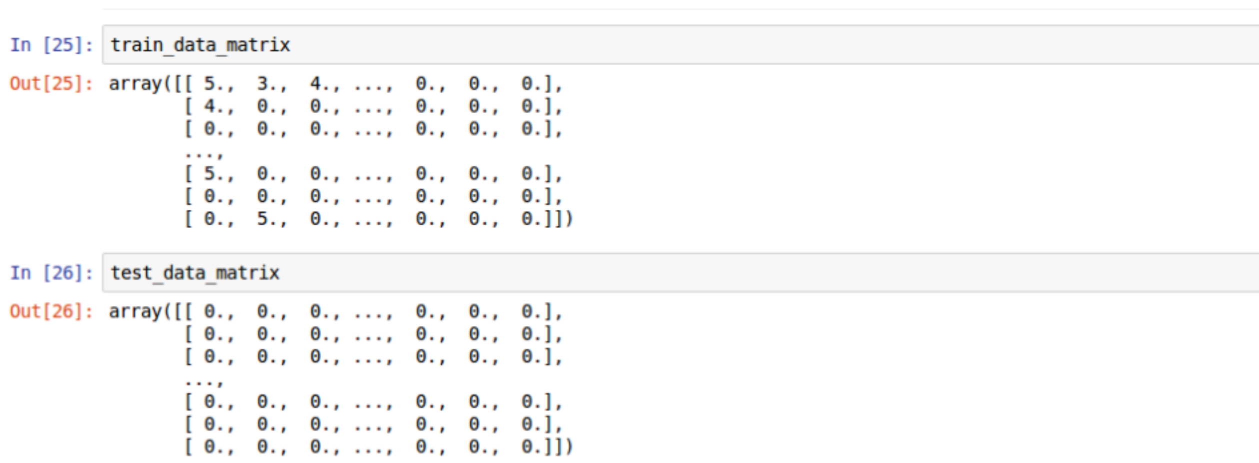
$$\hat{x}_{k,m} = \bar{x}_k + \frac{\sum sim_a(U_k, U_a)(x_{a,m} - \bar{x_{ua}})}{\sum |sim_u(u_k, u_a)|}$$

Since prediction is based on both item-item collaborative model and user-user based collaborative model we need to evaluate both the models which is done using Root Mean Squared Error(RMSE). Following is the formula to find evaluation metric:

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

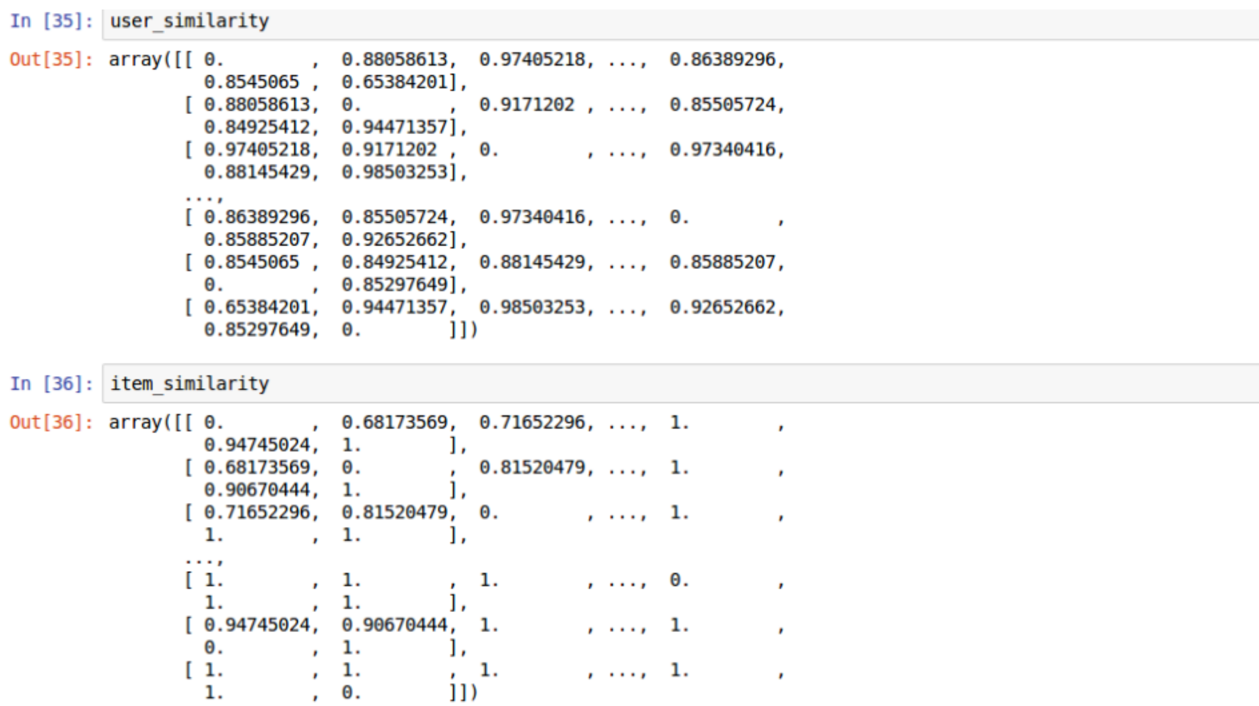
Results:

The Data matrix is shown below:



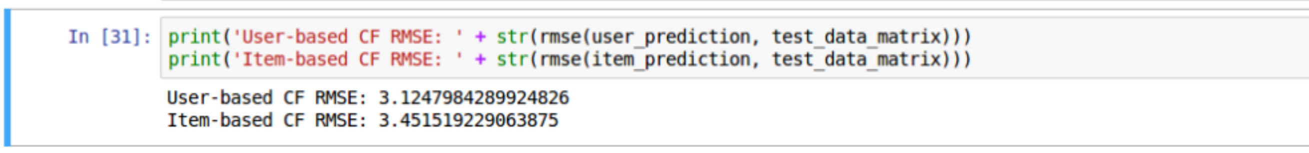
```
In [25]: train_data_matrix

Out[25]: array([[ 5.,  3.,  4., ...,  0.,  0.,  0.],
                [ 4.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                ...,
                [ 5.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  5.,  0., ...,  0.,  0.,  0.]])

In [26]: test_data_matrix

Out[26]: array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                ...,
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

The Similarity matrix is given here:



```
In [35]: user_similarity

Out[35]: array([[ 0.        ,  0.88058613,  0.97405218, ...,  0.86389296,
                  0.8545065 ,  0.65384201],
                [ 0.88058613,  0.        ,  0.9171202 , ...,  0.85505724,
                  0.84925412,  0.94471357],
                [ 0.97405218,  0.9171202 ,  0.        , ...,  0.97340416,
                  0.88145429,  0.98503253],
                ...,
                [ 0.86389296,  0.85505724,  0.97340416, ...,  0.        ,
                  0.85885207,  0.92652662],
                [ 0.8545065 ,  0.84925412,  0.88145429, ...,  0.85885207,
                  0.        ,  0.85297649],
                [ 0.65384201,  0.94471357,  0.98503253, ...,  0.92652662,
                  0.85297649,  0.        ]])

In [36]: item_similarity

Out[36]: array([[ 0.        ,  0.68173569,  0.71652296, ...,  1.        ,
                  0.94745024,  1.        ],
                [ 0.68173569,  0.        ,  0.81520479, ...,  1.        ,
                  0.90670444,  1.        ],
                [ 0.71652296,  0.81520479,  0.        , ...,  1.        ,
                  1.        ,  1.        ],
                ...,
                [ 1.        ,  1.        ,  1.        , ...,  0.        ,
                  1.        ,  1.        ],
                [ 0.94745024,  0.90670444,  1.        , ...,  1.        ,
                  0.        ,  1.        ],
                [ 1.        ,  1.        ,  1.        , ...,  1.        ,
                  1.        ,  0.        ]])
```

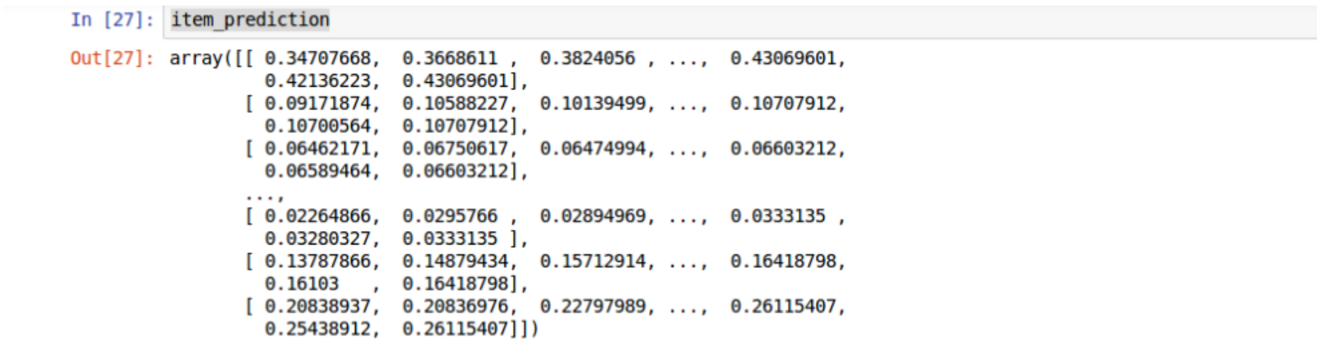
The matrix for Evaluation is given here:



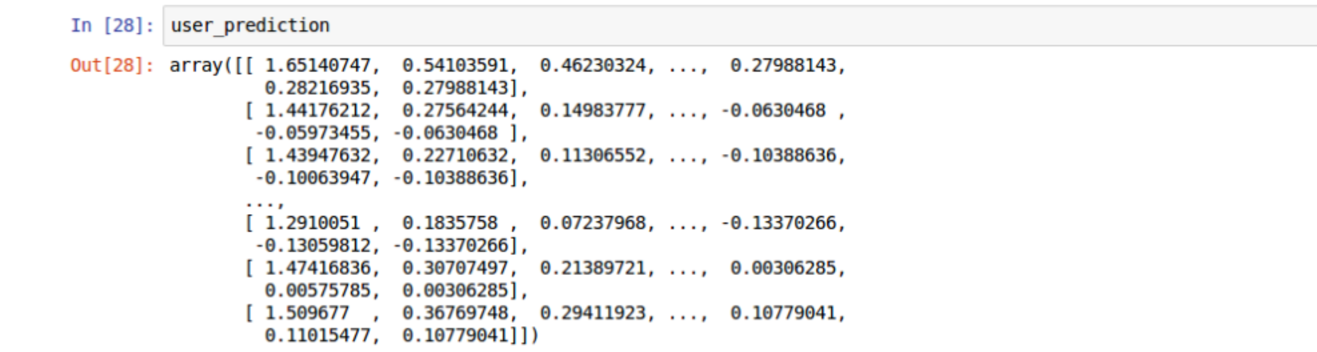
```
In [31]: print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
         print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))

         User-based CF RMSE: 3.1247984289924826
         Item-based CF RMSE: 3.451519229063875
```

From the calculated RMSE values we can observe that Item based collaborative filtering algorithm reccomends more accurate than user based collaborative filtering algorithm.

The Final Prediction matrix is given here:

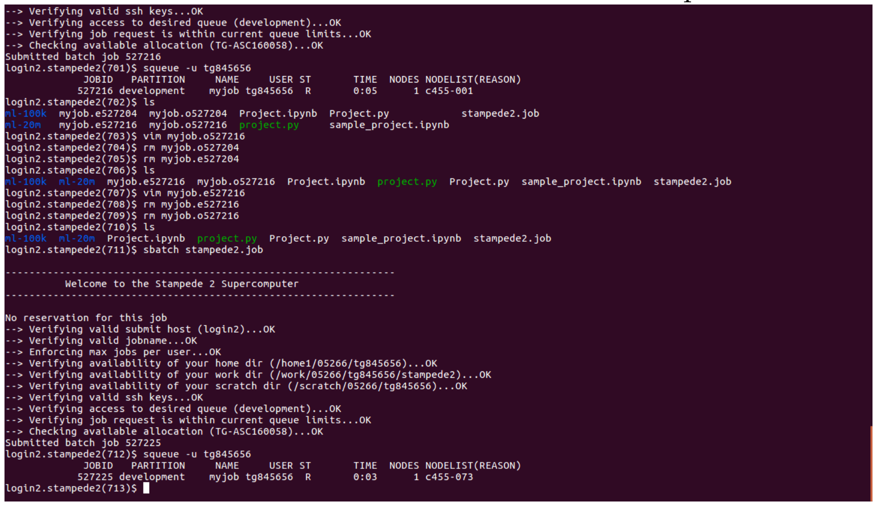


```
In [27]: item_prediction

Out[27]: array([[ 0.34707668,  0.3668611 ,  0.3824056 , ...,  0.43069601,
           0.42136223,  0.43069601],
         [ 0.09171874,  0.10588227,  0.10139499, ...,  0.10707912,
           0.10700564,  0.10707912],
         [ 0.06462171,  0.06750617,  0.06474994, ...,  0.06603212,
           0.06589464,  0.06603212],
         ...,
         [ 0.02264866,  0.0295766 ,  0.02894969, ...,  0.0333135 ,
           0.03280327,  0.0333135 ],
         [ 0.13787866,  0.14879434,  0.15712914, ...,  0.16418798,
           0.16103   ,  0.16418798],
         [ 0.20838937,  0.20836976,  0.22797989, ...,  0.26115407,
           0.25438912,  0.26115407]])
```



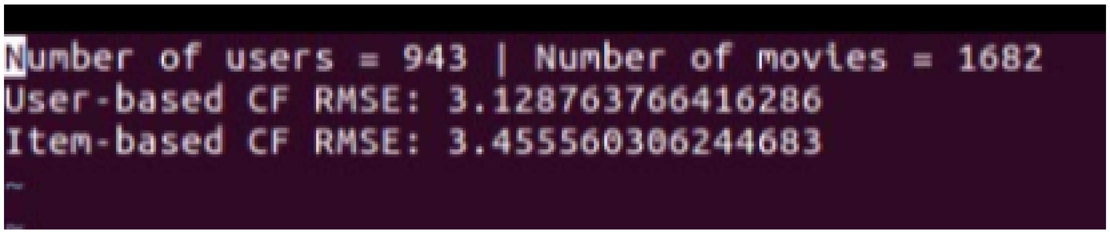
```
In [28]: user_prediction

Out[28]: array([[ 1.65140747,  0.54103591,  0.46230324, ...,  0.27988143,
           0.28216935,  0.27988143],
         [ 1.44176212,  0.27564244,  0.14983777, ..., -0.0630468 ,
          -0.05973455, -0.0630468 ],
         [ 1.43947632,  0.22710632,  0.11306552, ..., -0.10388636,
          -0.10063947, -0.10388636],
         ...,
         [ 1.2910051 ,  0.1835758 ,  0.07237968, ..., -0.13370266,
          -0.13059812, -0.13370266],
         [ 1.47416836,  0.30707497,  0.21389721, ...,  0.00306285,
           0.00575785,  0.00306285],
         [ 1.509677  ,  0.36769748,  0.29411923, ...,  0.10779041,
           0.11015477,  0.10779041]])
```

Screen shot of terminal when code is run on stampede2:



```
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (development)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (TG-ASC160058)...OK
Submitted batch job 527216
login2.stampede2(701)$ squeue -u tg845656
             JOBID   PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            527216 development    myjob tg845656  R       0:05      1 c455-001
login2.stampede2(702)$ ls
ml-100k  myjob.e527204  myjob.o527204  Project.ipynb  Project.py            stampede2.job
ml-20m   myjob.e527216  myjob.o527216  project.py     sample_project.ipynb
login2.stampede2(703)$ vim myjob.o527216
login2.stampede2(704)$ rm myjob.o527204
login2.stampede2(705)$ rm myjob.e527204
login2.stampede2(706)$ ls
ml-100k  ml-20m  myjob.e527216  myjob.o527216  Project.ipynb  project.py  Project.py  sample_project.ipynb  stampede2.job
login2.stampede2(707)$ vim myjob.e527216
login2.stampede2(708)$ rm myjob.e527216
login2.stampede2(709)$ rm myjob.o527216
login2.stampede2(710)$ ls
ml-100k  ml-20m  Project.ipynb  project.py  Project.py  sample_project.ipynb  stampede2.job
login2.stampede2(711)$ sbatch stampede2.job

-----------------------------------------------------------------
          Welcome to the Stampede 2 Supercomputer
-----------------------------------------------------------------

No reservation for this job
--> Verifying valid submit host (login2)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/05266/tg845656)...OK
--> Verifying availability of your work dir (/work/05266/tg845656/stampede2)...OK
--> Verifying availability of your scratch dir (/scratch/05266/tg845656)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (development)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (TG-ASC160058)...OK
Submitted batch job 527225
login2.stampede2(712)$ squeue -u tg845656
             JOBID   PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            527225 development    myjob tg845656  R       0:03      1 c455-073
login2.stampede2(713)$ 
```

Stampede Output:



```
Number of users = 943 | Number of movies = 1682
User-based CF RMSE: 3.128763766416286
Item-based CF RMSE: 3.455560306244683
~
~
```

Conclusion:

Finally, the project was implemented using Stampede2 for quicker prediction and efficiency. The data provided did not have much missing values, the prediction levels produced by the code are also reliable and hence generate closely similar data in the recommendation section using both Item-Item filtering and User-User collaborative filtering. Running the code on stampede2 required installations of Anaconda python distributions. This was done mainly because we used Jupyter Notebook which uses python3 for coding on the local machine, however Stampede was using python2 and hence required an installation. I wasn't able to run Jupyter Notebook on Stampede and hence did not use Jupyter.

References:

https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/

https://rpubs.com/jeknov/movieRec

https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

https://www.analyticsvidhya.com/blog/2015/12/started-graphlab-python/

https://pandas.pydata.org/pandas-docs/stable/merging.html