

Description:

Versions used:

Spark- 2.2.1

Python- 2.7

1. Commands to execute the program: (LSH)

```
bin\spark-submit Prashanth_Manja_task1_Jaccard.py <path_of_the_ratings.csv_file>
```

Example:

```
bin\spark-submit Prashanth_Manja_task1_Jaccard.py \data\ratings.csv
```

The above command will execute the program and will produce the output file "Prashanth_Manja_SimilarMovies_Jaccard.txt " .

Time taken : 28s

Precision and Recall of the implementation of LSH were calculated comparing with the "Ground_Truth_file" that has 384233 similar movies with jaccard similarity > 0.5.

Precision = $tp / (tp + fp) = 1.0$

Recall = $tp / (tp + fn) = 0.9236$

2. Commands to execute the program: (Task_2: Model-Based-CF):

1. Small Dataset:

```
bin\spark-submit Prashanth_Manja_task2_ModelBasedCF.py <path_of_the_ratings.csv_file>  
<path_of_testing_small.csv>
```

Example:

```
bin\spark-submit Prashanth_Manja_task2_ModelBasedCF.py \data\ratings.csv  
\data\testing_small.csv
```

Accuracy Output:

```
('>=0 and <1: ', 13731)  
( '>=1 and <2: ', 4997)  
( '>=2 and <3: ', 1214)  
( '>=3 and <4: ', 278)  
( '>=4: ', 36)  
( 'RMSE = ', 1.109115110800053)  
( 'Total_Execution_Time_Is-->', '21.8276219368')
```

2. Large Dataset

```
bin\spark-submit Prashanth_Manja_task2_ModelBasedCF.py <path_of_the_ratings.csv_file>  
<path_of_testing_20m.csv>
```

Example:

```
bin\spark-submit Prashanth_Manja_task2_ModelBasedCF.py \data\ratings.csv \data\  
testing_20m.csv
```

Accuracy Output:

```
('>=0 and <1: ', 3234566)  
('>=1 and <2: ', 717243)  
('>=2 and <3: ', 91023)  
('>=3 and <4: ', 10921)  
('>=4: ', 690)  
('RMSE = ', 0.8304312387014875)  
('Total_Execution_Time_Is-->', '2644.48605609')
```

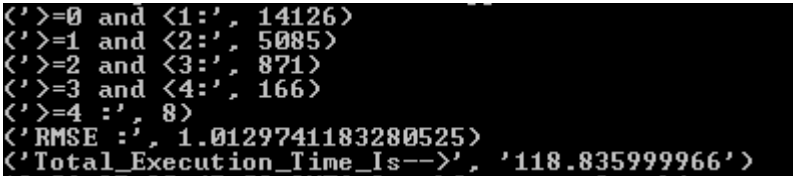
3. Commands to execute the program: (Task_2: User-Based-CF):

```
bin\spark-submit Prashanth_Manja_task2_UserBasedCF.py <path_of_the_ratings.csv_file>  
<path_of_testing_small.csv>
```

Example:

```
bin\spark-submit Prashanth_Manja_task2_UserBasedCF.py \data\ratings.csv \data\testing_small.csv
```

Accuracy Output:



```
('>=0 and <1: ', 14126)  
('>=1 and <2: ', 5085)  
('>=2 and <3: ', 871)  
('>=3 and <4: ', 166)  
('>=4 : ', 8)  
('RMSE : ', 1.0129741183280525)  
('Total_Execution_Time_Is-->', '118.835999966')
```

Improvements :To handle cold start problem (New User not rated any movie / movie not reviewed by other users) – by computing average ratings, we can optimize the RMSE by assigning weights to improve the ratings prediction for cases where no user has previously rated the movie similar to the current tested movie.

4. Commands to execute the program: (Bonus: Item-Based-CF):

```
bin\spark-submit Prashanth_Manja_task2_ItemBasedCF.py <path_of_the_ratings.csv_file>  
<path_of_testing_small.csv> <path_of_lsh_similarity_file>
```

Example:

```
bin\spark-submit Prashanth_Manja_task2_ItemBasedCF.py \data\ratings.csv  
\data\testing_small.csv \data\LSH_similarity.txt
```

Accuracy Output:

```
<'>=0 and <1:' , 14734>  
<'>=1 and <2:' , 4500>  
<'>=2 and <3:' , 909>  
<'>=3 and <4:' , 113>  
<'>=4 : , 0>  
<'RMSE : , 0.975031859977768>  
<'Total_Execution_Time_Is-->' , '175.769999981'>
```

LSH results in reduction of feature space when performing a similar items in the system.

Hash functions used in LSH to do row permutations and helps us we save a lot of disk space and time but we lose few data in the process.

Depending on the bands and rows chosen the output may contain false positives or negatives this will help in reducing time and space when running for huge datasets.

Both large and small systems can benefit from LSH with reductions in overall search times and disk space requirements making the recommendation efficient in finding similar items.

When output generated by LSH is used to calculate the prediction in recommendation system, we may observe a slight less accuracy but gives proper output and runs faster and efficiently.