Zuyan Jiang, Yan Tong, Ren Zhou

CSE 427 Final Project

Professor Neumann

May 4 2017

Final Project Report

1 Introduction

1.1 Background

As one of three possible topics for the final project, this collaborative filtering problem aims to analyze the Netflix data using MapReduce and/or Pig and to develop a feasible collaborative filtering approach, based on which one can predict how much a user would enjoy a new movie based on his previous movie preference (i.e. previous ratings without any additional information of movie or user).

Adopting from Netflix Prize, an open competition held by the Netflix company in 2009, the project uses Netflix data for analysis and prediction. For training and validation purpose, raw dataset is divided into two subsets, TrainingRatings.txt an TestingRatings.txt, where the training set is used to train the collaborative filtering model, and testing set is to validate how good the prediction is. In order to evaluate the performance, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are calculated by comparing the predicted rating to true rating for every movie-user pair in the testing set. In this project, the team implemented the problem in three steps and executed in both pseudo-cluster and Amazon EMR cloud.

The rest part of report contains detailed documentation of collaborative filtering approach, implementation and results, performance evaluation, and conclusion.

1.2 Motivation

Nowadays, recommendation system is commonly used in many areas. This recommendation system can efficiently make people life convenient and increase profit from item selling. Not only used in the movie recommendation, it can recommend songs, news, books, and paper. It is also used in online store such as Amazon. Some Social Apps like Facebook would use it to recommend new friends. Some search page could use collaborative filtering to rank the result. With usage of collaborative filtering, the relation between two items could be analyzed so as to predict other items with high similarity and filter other less important information. The result of this project can be used as criteria of recommendations system, and thus can be applied to real-world in many domains.

2 Methodology

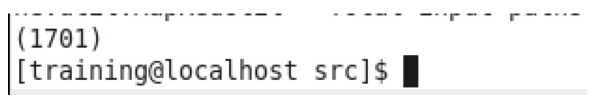
2.1 Netflix Data Analysis

In order to give feasible and efficient solution, the team first analyzed Netflix data and generated statistics of number of distinct movies and users in both training and testing datasets.

A Pig script was used for data analyzing. Figure 2-1 to Figure 2-4 describe the statistics for both datasets.

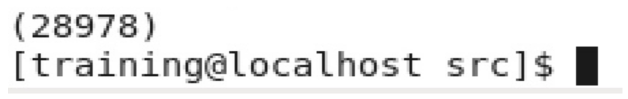
```
(27555)
```

Figure 2-1 Distinct User Count for TestingRatings.txt



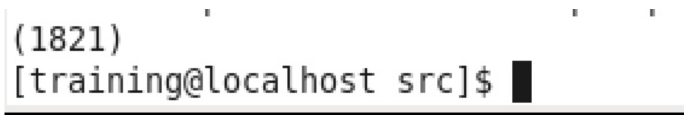
```
(1701)
[training@localhost src]$
```

Figure 2-2 Distinct movie count for TestingRatings.txt



```
(28978)
[training@localhost src]$
```

Figure 2-3 Distinct user count for TrainingRatings.txt



```
(1821)
[training@localhost src]$
```

Figure 2-4 Distinct movie count for TrainingRatings.txt

There are 27555 distinct users and 1701 distinct movies in the training set, while testing set contains 28978 distinct users and 1821 distinct movies. As what indicated in data description, the training set contains 3,255,352 rating records and testing set contains 100,478 records.

The average overlapping of items for users is defined as result of total rating records divided by distinct items, which is roughly 1914 per user for the training dataset and 55 for testing dataset. Similarly, the average overlapping of users for items is calculated as total records divided by distinct user count. Roughly speaking, each item has been rated by 118 users on average for training set, and 3 users for testing set.

Therefore, we concluded the average overlap of items for users is significantly higher than that of users for items, in either dataset.

2.2 Choice of Parameters

After discussing several possible approaches, the team decided to use the following parameter setups for collaborative filtering:

2.2.1 Model: Item-Item Model

Netflix data suggests that the average overlap of items for users is significantly higher than that of users for items in either set, so it would be more convenient to use item-item pairs as the key for implementation, compared with user-user pairs.

Moreover, since a user can have many inactive accounts that provides limited movie ratings yet is treated equally as active accounts, it would be hard to distinguish which is which and furthermore filter out active accounts for modeling. On the other hand, even though there may be some items that has never been rated or rated only once, the probability of getting once-or-no-rated items is getting smaller as more data are used. Hence, it would be efficient and practical in general to use item-item model.

Based on previous two reasons, the team concluded that using item-item model is the most suitable model yet reasonable and practical in general.

2.2.2 Similarity measurement: Pearson correlation

We decided to use Pearson similarity since it can remove bias caused by extreme users, and furthermore improve the performance.

The formula for Pearson correlation is shown as Figure 2-1, where x and y are items of interests, and r is similarity score between item x and y.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

2.2.3 Threshold the ratings: No threshold

Using a threshold is not necessary as a Pearson correlation is used.

2.2.4 Normalize the ratings or not: Yes

As lecture indicated, normalization can remove bias caused by enthusiastic or over-critical users, improves similarity measurement and give better performance. Since we have chosen Pearson correlation as the similarity measurement, ratings are indeed normalized.

2.2.5 Number of similar items k: 10

The team agreed to set up initial value of k as 10, which means that the program would search for 10 most similar items towards item of interest. The final value of K will be determined by a series of experiments and the related performance of model.

2.2.6 Prediction method: Weighted average

Referring to lecture notes, a weighted average would be a better choice for predicting rating scores. The equation for weighted average score is shown as Figure 2-2.

$$\textit{predicted score} = \frac{\sum_{x \in S} r(\textit{target-item}, x) * \textit{RATING}(x)}{\sum_{x \in S} r(\textit{target-item}, x)}$$

where, S = list of k most similar items for target-item

r = Pearson correlation score

RATING(x) = rating of given by target-user, 0 if no rating found

3 Implementation

3.1 Sample Data

The team chose a part of data from the trainingRatings.txt to be the sample data, and used it to train the model and test the model locally. The command used to generate sample data is as following:

```
[training@localhost src]$ sort -R TrainingRatings.txt | head -n 300000 > sampleT
raining.txt
```

3.2 Workflow and Data Flow

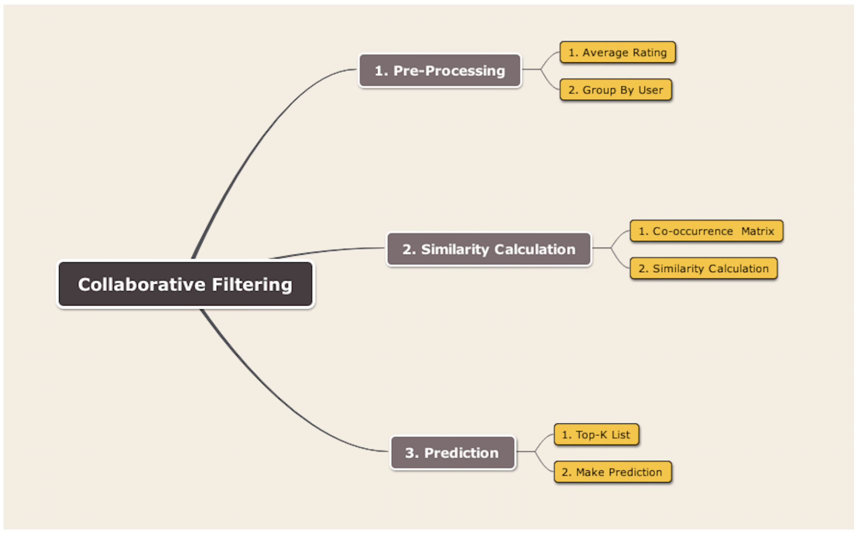


Figure 3-1 Job Structure

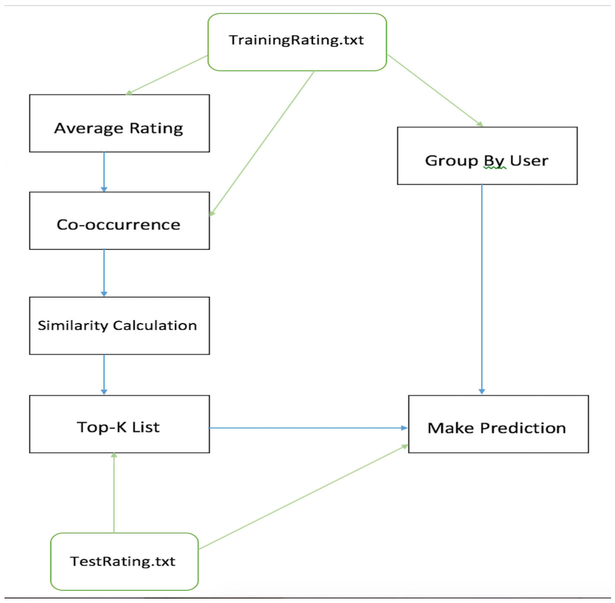


Figure 3-2 Data Flow

3.3 Detailed Job Implementation

3.3.1 Step 1: Preprocessing

3.3.1.1 Job 1: Average Rating Calculation

This job was designed to calculate the average rating of each item since the team will use it when create the similarity matrix. In the Mapper phase, the mapper read the trainingRating.txt and put out the key-value pair as <itemId,rating>. And in the Reducer phase, every reducer could calculate the average rating of the certain item.

The input and output of Mapper and Reducer are listed as below:

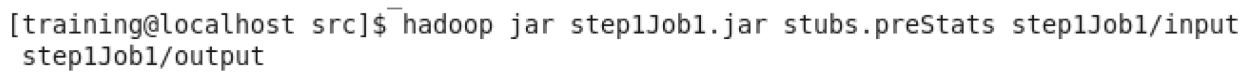
Mapper Input: (itemid, userid, rating)

Mapper Output: (itemid, rating)

Reducer Input: (itemid, [rating1, rating2, …])

Reducer Output: (itemid, avgRating)

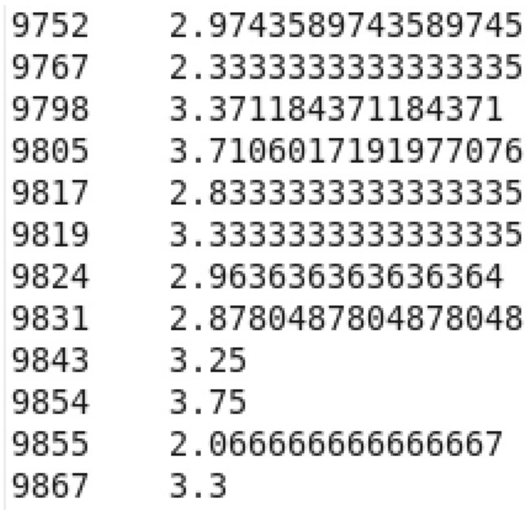
Command used for compiling is:



```
[training@localhost src]$ hadoop jar step1Job1.jar stubs.preStats step1Job1/input
 step1Job1/output
```

Figure 3-3 Execution Command Used in Pseudo-Cluster

Part of the output file of this job is shown as Figure 3-3. For every line of the result file, the key of output is the itemId. The value is the average rating of this item, which is separated by "\t" from key.



```
9752    2.9743589743589745
9767    2.3333333333333335
9798    3.371184371184371
9805    3.7106017191977076
9817    2.8333333333333335
9819    3.3333333333333335
9824    2.963636363636364
9831    2.8780487804878048
9843    3.25
9854    3.75
9855    2.066666666666667
9867    3.3
```

Figure 3-4 Result for Step 1 Job 1

3.3.1.2 Job 2: Group Data by User

This job is designed to prepare data for rating prediction. Taking the training records as input, the job breaks a line of record into item ID, user ID and rating. User ID is emitted as key, whereas item ID and rating are grouped as values.

The input and output of Mapper and Reducer are listed as below:

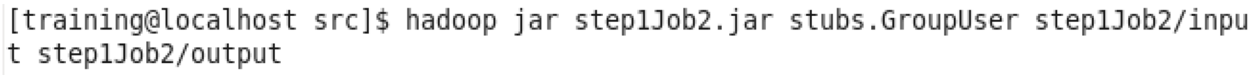
Mapper Input: (itemid, userid, rating)

Mapper Output: (userid, (itemid, rating))

Reducer Input: (userid, [itemid1, rating1), (itemid2, rating2), …])

Reducer Output: (user ID, [itemid1, rating1), (itemid2, rating2), …]);

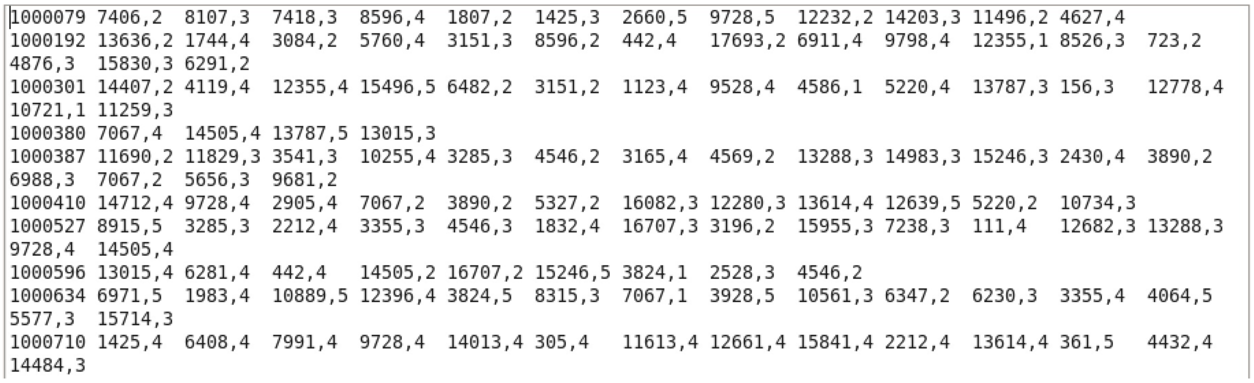
Command used for compiling is:



```
[training@localhost src]$ hadoop jar step1Job2.jar stubs.GroupUser step1Job2/inpu
t step1Job2/output
```

Figure 3-5 Execution Command Used in Pseudo-Cluster

The result of this job is as Figure 3-6. For each line of text, key is at the beginning, followed by values delimited by '\t', within which item ID and rating are separated by comma.



```
1000079 7406,2  8107,3  7418,3  8596,4  1807,2  1425,3  2660,5  9728,5  12232,2 14203,3 11496,2 4627,4
1000192 13636,2 1744,4  3084,2  5760,4  3151,3  8596,2  442,4   17693,2 6911,4  9798,4  12355,1 8526,3  723,2
4876,3  15830,3 6291,2
1000301 14407,2 4119,4  12355,4 15496,5 6482,2  3151,2  1123,4  9528,4  4586,1  5220,4  13787,3 156,3   12778,4
10721,1 11259,3
1000380 7067,4  14505,4 13787,5 13015,3
1000387 11690,2 11829,3 3541,3  10255,4 3285,3  4546,2  3165,4  4569,2  13288,3 14983,3 15246,3 2430,4  3890,2
6988,3  7067,2  5656,3  9681,2
1000410 14712,4 9728,4  2905,4  7067,2  3890,2  5327,2  16082,3 12280,3 13614,4 12639,5 5220,2  10734,3
1000527 8915,5  3285,3  2212,4  3355,3  4546,3  1832,4  16707,3 3196,2  15955,3 7238,3  111,4   12682,3 13288,3
9728,4  14505,4
1000596 13015,4 6281,4  442,4   14505,2 16707,2 15246,5 3824,1  2528,3  4546,2
1000634 6971,5  1983,4  10889,5 12396,4 3824,5  8315,3  7067,1  3928,5  10561,3 6347,2  6230,3  3355,4  4064,5
5577,3  15714,3
1000710 1425,4  6408,4  7991,4  9728,4  14013,4 305,4   11613,4 12661,4 15841,4 2212,4  13614,4 361,5   4432,4
14484,3
```

Figure 3-6 Result for Step 1 Job 2

3.3.2 Step 2: Find Similar Items

3.3.2.1 Job 1: Co-occurrence matrix

This job generates a co-occurrence matrix for each (item, item) pair with corresponding stats.

Program input comes from original training dataset, with a distributed cache coming from output of Step1Job1 (i.e item-avgRating pairs for each item in the training set).

The Mapper receives data from input file and distributed cache, and emits (userid, itemid, stats) where stats contains specific rating for that itemid given by user as well as average rating given by all users. The Reducer loops through the list of all (itemid, stats) pairs, performs a cross-product operation for each item pair using natural join operation method, and emits ((item1,item2),(stats1,stats2)).

Input and output for Mapper and Reducer are:

Mapper Input : (userid, itemid, rating)

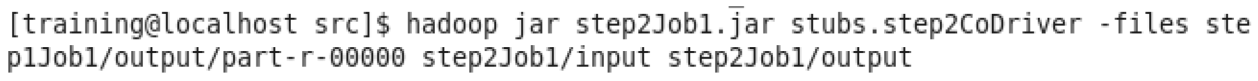
Distributed cache: (itemid, avgRating) from step1Job1/part-r-00000

Mapper Output : (userid, (itemid,rating, avgRating))

Reducer Input: (userid, [(itemid1, stats1), (itemid2, stats2), …])

Reducer Output: ((itemid1,itemid2), (rating1, avgRating1, rating2, avgRating2))

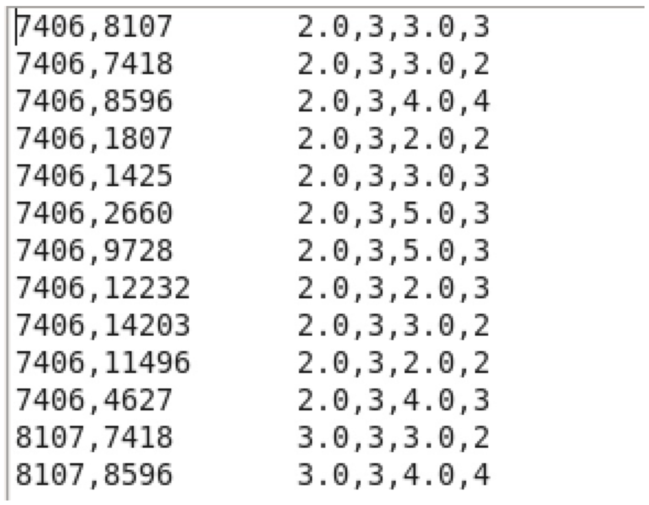
Command used:



```
[training@localhost src]$ hadoop jar step2Job1.jar stubs.step2CoDriver -files ste
p1Job1/output/part-r-00000 step2Job1/input step2Job1/output
```

Figure 3-7 Execution Command in Pseudo-cluster

Result from Reducer contains (itemid, itemid) and (stats1,stats2) separated by tab. For the first line in Figure 3-7, it suggests that item 13887 and 7067 are two items rated by same user X. Item 13387 is rated as 2.0 by user X and has an average rating of 3 for all users. Yet item 7067 is rated as 2.0 by user X and has an average rating of 2 for all users.



```
7406,8107       2.0,3,3.0,3
7406,7418       2.0,3,3.0,2
7406,8596       2.0,3,4.0,4
7406,1807       2.0,3,2.0,2
7406,1425       2.0,3,3.0,3
7406,2660       2.0,3,5.0,3
7406,9728       2.0,3,5.0,3
7406,12232      2.0,3,2.0,3
7406,14203      2.0,3,3.0,2
7406,11496      2.0,3,2.0,2
7406,4627       2.0,3,4.0,3
8107,7418       3.0,3,3.0,2
8107,8596       3.0,3,4.0,4
```

Figure 3-8 Result for Step 2 Job 1

3.3.2.2 Job 2: Similarity Calculation

After getting the co-occurrence matrix, the team needed to create the items' similarity matrix. As mentioned above, the team evaluated the similarity between pairs of items using Pearson Correlation and here is the formula:

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

So the purpose of this job was to implement the Pearson Correlation. The team got the result file from co-occurrence job as the input of this job. And put out the similarities of items.

The only purpose of Mapper was to prepare for the Reducer, so the team used an identity mapper. And in the Reducer phase, every reducer received certain pair of items and the aggregated statistics data. And then it could calculate the similarity between these two items based on Pearson Correlation.

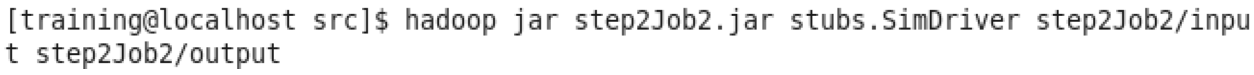
The input and output format is as following:

Mapper: Identity Mapper

Reducer Input: ((itemid,itemid), [(rating1-avgRating1, rating2 – avgRating2), …])

Reducer Output: ((item,item),similarity)

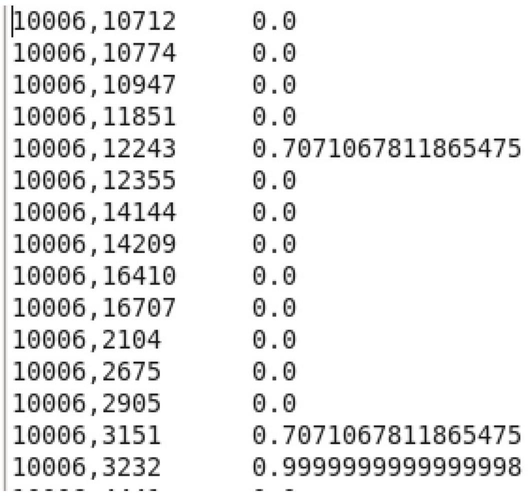
Command used for compiling:



```
[training@localhost src]$ hadoop jar step2Job2.jar stubs.SimDriver step2Job2/inpu
t step2Job2/output
```

Figure 3-9 Execution Command in Pseudo-cluster

Part of the output file of this job is shown as Figure 3-9. For every line of the output file, the key is comma separated item ID and item ID, value is the similarity of these two items.



```
10006,10712     0.0
10006,10774     0.0
10006,10947     0.0
10006,11851     0.0
10006,12243     0.7071067811865475
10006,12355     0.0
10006,14144     0.0
10006,14209     0.0
10006,16410     0.0
10006,16707     0.0
10006,2104      0.0
10006,2675      0.0
10006,2905      0.0
10006,3151      0.7071067811865475
10006,3232      0.9999999999999998
```

Figure 3-10 Result for Step 2 Job 2

To improve the performance of the model, there are three value in the output should be dealt with: NaN, 1 and -1. NaN indicated the denominator whenever a Pearson Correlation result is zero, and the team replaced these NaN by 0. For 1 and -1, which meant the related item pair only co-occurrence for once, is not such useful for the model. So the team simply got rid of them.

3.3.3 Step 3: Prediction

3.3.3.1 Job1: Top K Similar Items

As an essential step, this job generated the K most similar items and the corresponding similarities for every item involved in Testing Rating pair.

The input of this job, i.e. the similarity matrix, was obtained by step2-job2. Also, 'TestingRating.txt' would used as a lookup table. For every entry of the similarity matrix, if any of the two items was involved in 'TestingRating.txt', the job found all items that are similar to this item and picks K of them with highest similarities. And K was designed as a parameter passed when submitting the job. In this particular case, considering that the average overlap of items was approximately 1700, K number was set to 1500.

The input and output format were as following:

Mapper Input: ((itemid, itemid), similarity)

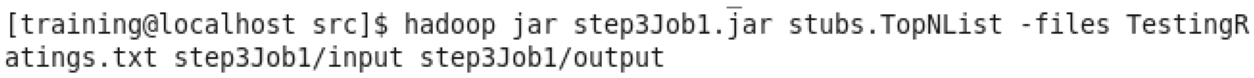
Distributed Cache: TestingRatings.txt

Mapper Output: (itemid, (itemid, similarity))

Reducer Input: (itemid, [(itemid1, similarity1),...,(itemidK, similarityK)])

Reducer output: item ID, list of (item ID, similarity) (size K);

Command used:

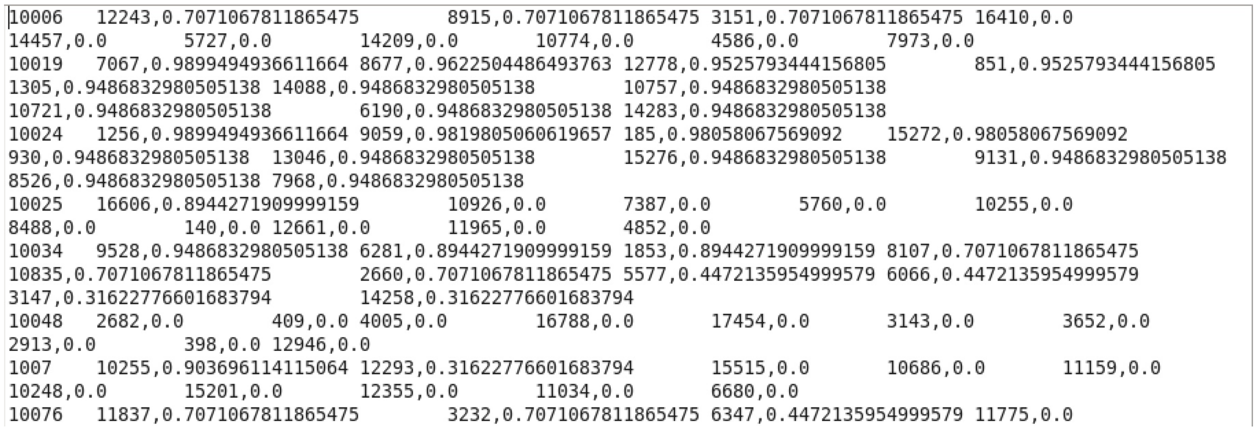


```
[training@localhost src]$ hadoop jar step3Job1.jar stubs.TopNList -files TestingR
atings.txt step3Job1/input step3Job1/output
```

Figure 3-11 Execution Command in Pseudo-cluster

Result

The figure below revealed part of one line of output by this job, where the K was 1500. One line of the output consisted key item and '\t' delimitated (similar item, similarity) pair.



```
10006   12243,0.7071067811865475        8915,0.7071067811865475 3151,0.7071067811865475 16410,0.0
14457,0.0       5727,0.0        14209,0.0       10774,0.0       4586,0.0        7973,0.0
10019   7067,0.9899494936611664 8677,0.9622504486493763 12778,0.9525793444156805        851,0.9525793444156805
1305,0.9486832980505138 14088,0.9486832980505138        10757,0.9486832980505138
10721,0.9486832980505138        6190,0.9486832980505138 14283,0.9486832980505138
10024   1256,0.9899494936611664 9059,0.9819805060619657 185,0.98058067569092    15272,0.98058067569092
930,0.9486832980505138  13046,0.9486832980505138        15276,0.9486832980505138        9131,0.9486832980505138
8526,0.9486832980505138 7968,0.9486832980505138
10025   16606,0.8944271909999159        10926,0.0       7387,0.0        5760,0.0        10255,0.0
8488,0.0        140,0.0 12661,0.0       11965,0.0       4852,0.0
10034   9528,0.9486832980505138 6281,0.8944271909999159 1853,0.8944271909999159 8107,0.7071067811865475
10835,0.7071067811865475        2660,0.7071067811865475 5577,0.4472135954999579 6066,0.4472135954999579
3147,0.31622776601683794        14258,0.31622776601683794
10048   2682,0.0        409,0.0 4005,0.0        16788,0.0       17454,0.0       3143,0.0        3652,0.0
2913,0.0        398,0.0 12946,0.0
1007    10255,0.903696114115064 12293,0.31622776601683794       15515,0.0       10686,0.0       11159,0.0
10248,0.0       15201,0.0       12355,0.0       11034,0.0       6680,0.0
10076   11837,0.7071067811865475        3232,0.7071067811865475 6347,0.4472135954999579 11775,0.0
```

Figure 3-12 Result for Step 3 Job 1

3.3.3.2 Step3, Job2-Prediction

As the final step of collaborative filter, prediction could be performed in several manners. Weighted average, a commonly used manner, was implemented in this project. As to certain (user ID, item ID) pair, aim of this job was to calculate the weighted average over all history ratings by this user on all items that were similar to the item from the pair. It was equivalent to joining the two lists, respectively the Top-K list of this item and the list of history ratings of this user, by item ID, and applying the formula below.

$$\frac{\sum(Similarity \times Rating)}{\sum Similarity}$$

As to the detail data reading, this job took grouped training file from step1-job2 as job input, and the smaller two files, the Top-K list yielded by step3-job1 and 'TestingRatings.txt', as distributed cache files. Though not very intuitive, this approach reduced the size of lookup tables, and thus, improved memory performance.

Input/output format

Mapper input: user ID, list of (item ID, rating) (records from step1-job2);

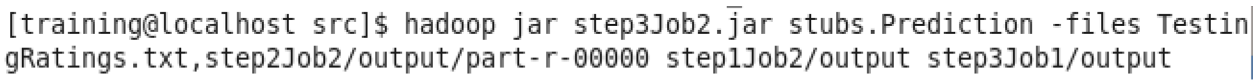
Distributed cache: TestingRatings.txt, Top-K list from step3-job1;

Mapper output: key: (item ID, user ID) (from 'TestingRatings.txt'), value: (similarity, rating) (Tope-K similar item ID omitted);

Reducer input: (item ID, user ID), list of (similarity, rating);

Reducer output: (item ID, user ID), weighted average (prediction);

Command used:

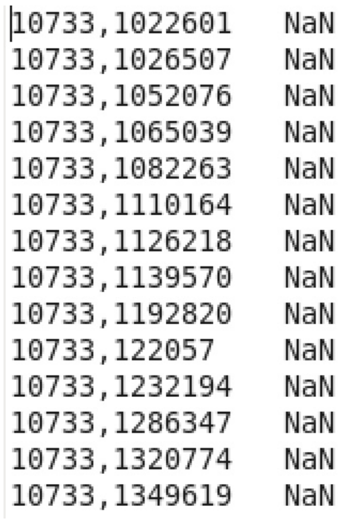


```
[training@localhost src]$ hadoop jar step3Job2.jar stubs.Prediction -files Testin
gRatings.txt,step2Job2/output/part-r-00000 step1Job2/output step3Job1/output
```

Figure 3-13 Execution Command in Pseudo-cluster

Result

Part of the output file of this job is shown below. In each line, the key is comma separated item ID and user ID, value is the prediction of the rating.



```
10733,1022601   NaN
10733,1026507   NaN
10733,1052076   NaN
10733,1065039   NaN
10733,1082263   NaN
10733,1110164   NaN
10733,1126218   NaN
10733,1139570   NaN
10733,1192820   NaN
10733,122057    NaN
10733,1232194   NaN
10733,1286347   NaN
10733,1320774   NaN
10733,1349619   NaN
```

Figure 3-14 Result for Step 3 Job 2

3.3.4 Result and Discussion

According to the result file of Step3 Job2, the team found that the most part of prediction result is NaN, which means that the model could not predict the missing rating of certain item by certain user. Meanwhile, there were too many 0 in step2 job2's output file since the team replaced all NaN by 0. So in term of the unreasonable output, there might be three possibilities: 1. The K value is too small. 2. The model's performance is poor. 3. The data set is not big enough. Then the team selected a part of data from the original sample data in which the similarities of item pairs are manifest and re-predicted the missing rating. The output was better based on the selected data set. So the team concluded that small data set, which result in the sparse similarity matrix, might be the main reason for the irrational output. And it is necessary to train the model using the whole data set.

4 Cloud Execution in EMR

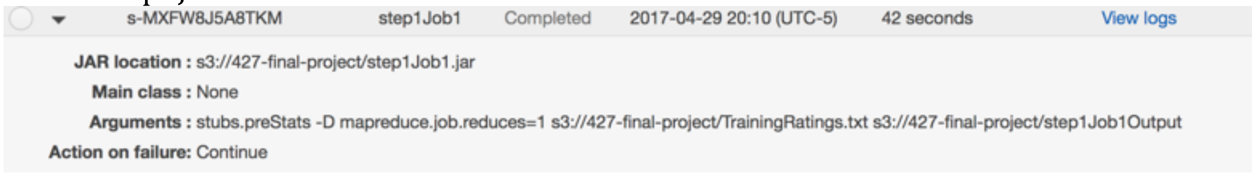
4.1 Challenge of Big Data

Our team faced a problem that the training data is too big to run in pseudo-cluster. In order to test correctness of code, we used part of the data set to get the testing result. Making sure that the train model worked, we ran the whole data set in EMR, and successfully got the real result from train model.

4.2 Implement

With the training model we ran with sample data set, our team used the same code to run in the EMR. Here is screenshot that train model ran in EMR with TrainingRating.txt

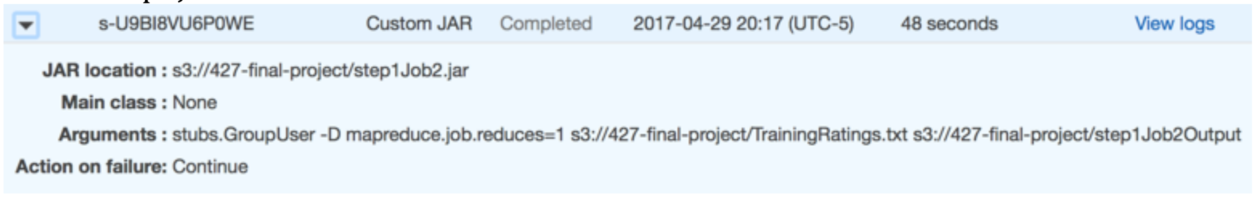
Step1Job1



s-MXFW8J5A8TKM step1Job1 Completed 2017-04-29 20:10 (UTC-5) 42 seconds View logs

JAR location : s3://427-final-project/step1Job1.jar
Main class : None
Arguments : stubs.preStats -D mapreduce.job.reduces=1 s3://427-final-project/TrainingRatings.txt s3://427-final-project/step1Job1Output
Action on failure: Continue

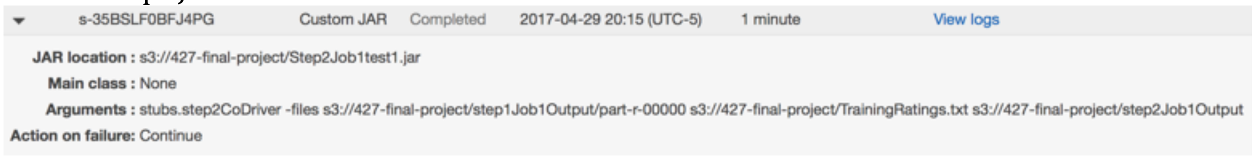
Step1Job2



s-U9BI8VU6P0WE Custom JAR Completed 2017-04-29 20:17 (UTC-5) 48 seconds View logs

JAR location : s3://427-final-project/step1Job2.jar
Main class : None
Arguments : stubs.GroupUser -D mapreduce.job.reduces=1 s3://427-final-project/TrainingRatings.txt s3://427-final-project/step1Job2Output
Action on failure: Continue

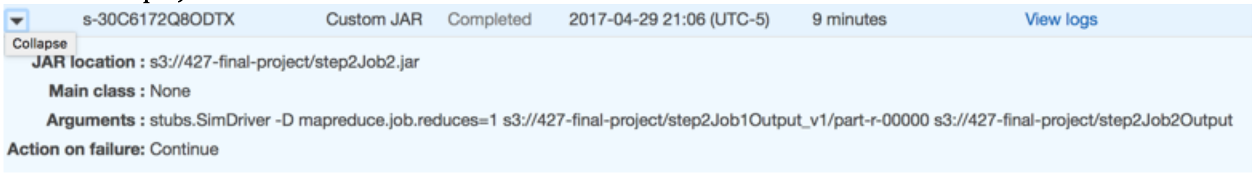
Step2Job1



s-35BSLF0BFJ4PG Custom JAR Completed 2017-04-29 20:15 (UTC-5) 1 minute View logs

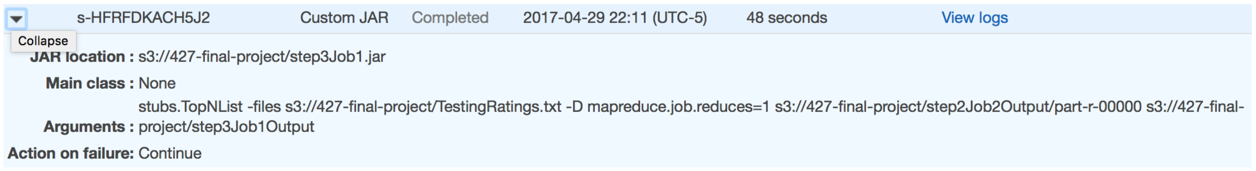
JAR location : s3://427-final-project/Step2Job1test1.jar
Main class : None
Arguments : stubs.step2CoDriver -files s3://427-final-project/step1Job1Output/part-r-00000 s3://427-final-project/TrainingRatings.txt s3://427-final-project/step2Job1Output
Action on failure: Continue

Step2Job2



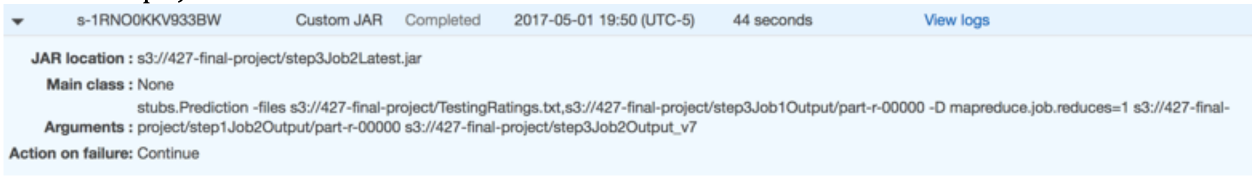
```
s-30C6172Q8ODTX    Custom JAR    Completed    2017-04-29 21:06 (UTC-5)    9 minutes    View logs
JAR location : s3://427-final-project/step2Job2.jar
Main class : None
Arguments : stubs.SimDriver -D mapreduce.job.reduces=1 s3://427-final-project/step2Job1Output_v1/part-r-00000 s3://427-final-project/step2Job2Output
Action on failure: Continue
```

Step3Job1



```
s-HFRFDKACH5J2    Custom JAR    Completed    2017-04-29 22:11 (UTC-5)    48 seconds    View logs
JAR location : s3://427-final-project/step3Job1.jar
Main class : None
Arguments : stubs.TopNList -files s3://427-final-project/TestingRatings.txt -D mapreduce.job.reduces=1 s3://427-final-project/step2Job2Output/part-r-00000 s3://427-final-project/step3Job1Output
Action on failure: Continue
```

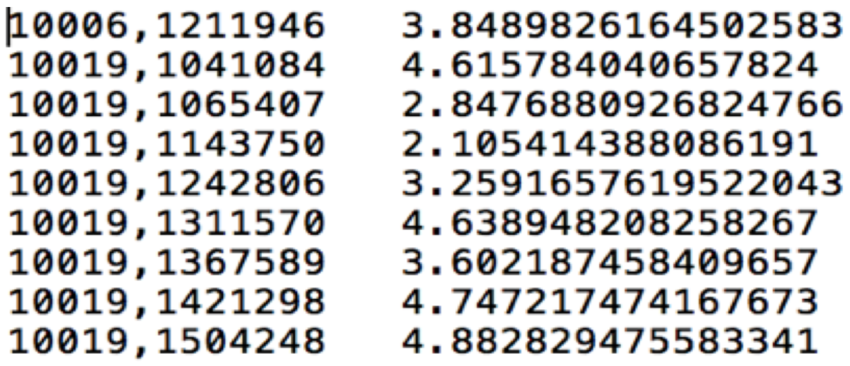
Step3Job2



```
s-1RNO0KKV933BW    Custom JAR    Completed    2017-05-01 19:50 (UTC-5)    44 seconds    View logs
JAR location : s3://427-final-project/step3Job2Latest.jar
Main class : None
Arguments : stubs.Prediction -files s3://427-final-project/TestingRatings.txt,s3://427-final-project/step3Job1Output/part-r-00000 -D mapreduce.job.reduces=1 s3://427-final-project/step1Job2Output/part-r-00000 s3://427-final-project/step3Job2Output_v7
Action on failure: Continue
```

4.3 Result

Part of the output file with big data input, TrainingRatings.txt, is shown below. In each line, the key is comma separated item ID and user ID, value is the prediction of the rating.



```
10006,1211946    3.848982616450258
10019,1041084    4.615784040657824
10019,1065407    2.847688092682476
10019,1143750    2.10541438808619l
10019,1242806    3.259165761952204
10019,1311570    4.63894820825826
10019,1367589    3.60218745840965
10019,1421298    4.74721747416767
10019,1504248    4.88282947558334
```

4.4 Discussion

With a data set which is big enough, we could get a reasonable weight rating of each item according to item-item similarity. In the following evaluation of MAE and RMSE, it proves that an acceptable prediction with low error could be gotten when the whole data set of training rating run in the EMR.

5 Result Evaluation

5.1 MAE and RMSE calculation

A Pig job was executed to evaluate how good the prediction is. The job reads in output from Step3Job2 (i.e. predictive result for each item-user pair in TestingRatings.txt) along with TestingRating.txt, and generates the (predictiveRating, trueRating) pair for every item-user pair to be predicted.

Equation used for calculation are:

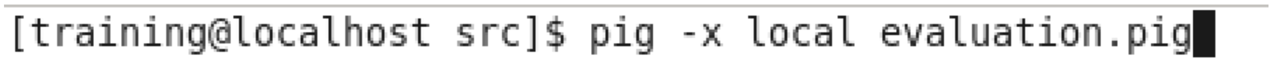
$$MAE = \frac{\sum_{i=1}^{n} |predictRating - trueRating|}{n}$$

where n is the total number of item-user pairs being predicted.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (predictRating - trueRating)^2}{n}}$$

where n is total number of item-user pairs being predicted.

Command used:



```
[training@localhost src]$ pig -x local evaluation.pig
```

Figure 5-1 Command used in pseudo-cluster

The team tried to evaluate the model for several different k values, and constructed a graph showing how MAE and RMSE changes as the value of k changes (shown in Figure 5-2).

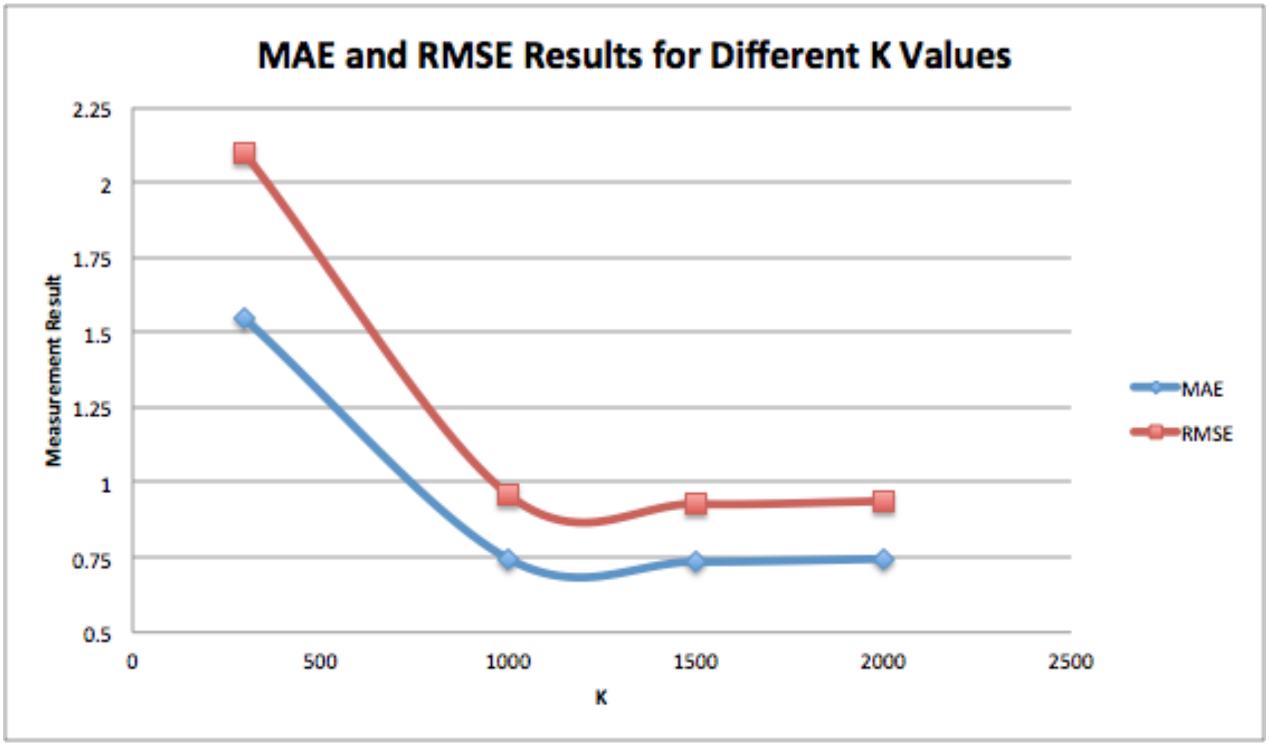


Figure 5-2 MAE and RMSE Results for Different K Values

Our team set k as the top k ratings of similarity rather than the overlap k ratings between similarity matrix and testing ratings matrix. According to the table above, when k equaled to 1500, it reached an optimal situation with lower MAE and RMSE.

Table 5-1 MAE and RSME under Different Values of K

Value of K	Mean Absolute Error	Root Square Mean Error
300	1.5437	2.1012
1000	0.7457	0.9598
1500	0.73512	0.9276
2000	0.74478	0.93709

6 New User Performance

6.1 Prediction Process

Step1. Our team created three new users, named as 18000000, 19930705, 92930, following with movie rating to different movie.

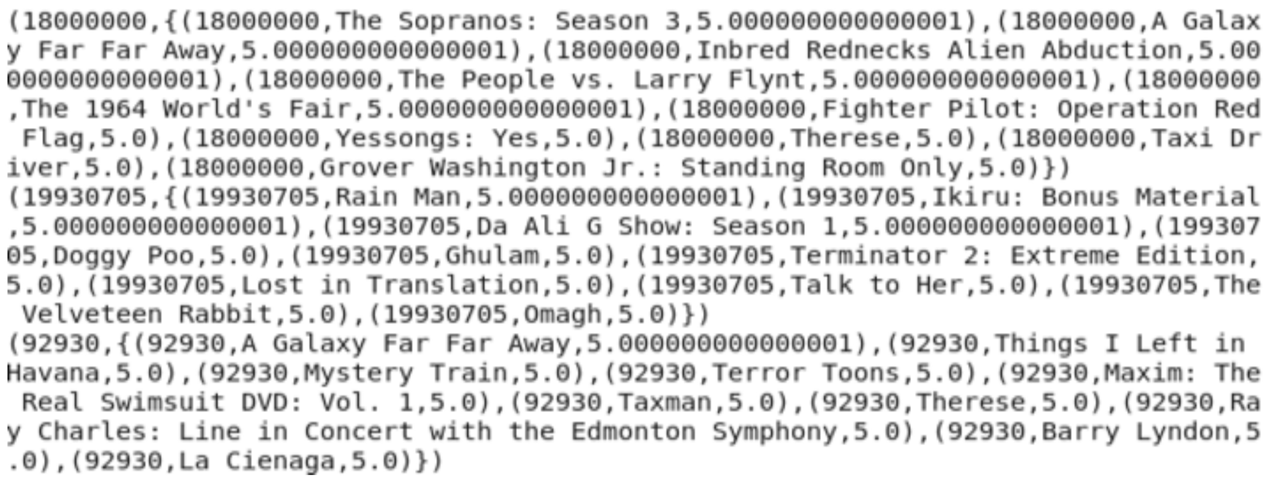
Step2. Replaced TestingRatings.txt with three new users rating. Run Step1, Step2, Step3 and got the weighted rating of each movie for three users.

Step3. Using Pig to select top 10 ratings item and search movie title according to item id. Predicted.

Step4. Let new users to determine whether recommend movies are interesting or not. After the inquiry, we get that the most of the movies made new users interested.

6.2 Result

In the following result, the format is (user id, {list top 10 of (user id, movie title, weighted ratings)}).



```
(18000000,{(18000000,The Sopranos: Season 3,5.000000000000001),(18000000,A Galax
y Far Far Away,5.000000000000001),(18000000,Inbred Rednecks Alien Abduction,5.00
0000000000001),(18000000,The People vs. Larry Flynt,5.000000000000001),(18000000
,The 1964 World's Fair,5.000000000000001),(18000000,Fighter Pilot: Operation Red
 Flag,5.0),(18000000,Yessongs: Yes,5.0),(18000000,Therese,5.0),(18000000,Taxi Dr
iver,5.0),(18000000,Grover Washington Jr.: Standing Room Only,5.0)})
(19930705,{(19930705,Rain Man,5.000000000000001),(19930705,Ikiru: Bonus Material
,5.000000000000001),(19930705,Da Ali G Show: Season 1,5.000000000000001),(199307
05,Doggy Poo,5.0),(19930705,Ghulam,5.0),(19930705,Terminator 2: Extreme Edition,
5.0),(19930705,Lost in Translation,5.0),(19930705,Talk to Her,5.0),(19930705,The
 Velveteen Rabbit,5.0),(19930705,Omagh,5.0)})
(92930,{(92930,A Galaxy Far Far Away,5.000000000000001),(92930,Things I Left in
Havana,5.0),(92930,Mystery Train,5.0),(92930,Terror Toons,5.0),(92930,Maxim: The
 Real Swimsuit DVD: Vol. 1,5.0),(92930,Taxman,5.0),(92930,Therese,5.0),(92930,Ra
y Charles: Line in Concert with the Edmonton Symphony,5.0),(92930,Barry Lyndon,5
.0),(92930,La Cienaga,5.0)})
```

Figure 6-1 Result of New User Recommendation

7 Conclusion

As the conclusion, the model can predict the rating for a certain movie by a certain user according to his/her historical rating, as well as generate the reasonable recommendation list for users of those movies they are likely interested in.

In further study, the team will refer to more materials on collaborative filtering algorithms and optimize the model in many ways such as tuning the K value, modifying the formula of calculating weighted rating and so on.