FeiFan Liu, Kunpeng Xie, Na Chen, Qianzhi Li nac 467573,465958 , 467510, 462597

CSE427S Final Project Report

Collaborative Filtering using the Netflix Data

FeiFan Liu, Kunpeng Xie, Na Chen, Qianzhi Li

December 14, 2018

1.Implementation

Submitted as code.

2.Introduction

Recently, recommendation systems have been used in diverse applications, such as social network suggesting friends, online market recommending items or services to users and so on. For a company, it is very important to have the ability of gaining profit and attracting customers. For a customer, it is very important to find the item or service that is truly liked by them and enhancing their satisfaction. In this case, a good recommendation becomes more and more important for both company and customer. In real life, with the development of technology, it is getting more and more easy for companies to provide better recommendation through gathering more data about user preferences and processing data on the cloud.

In this project, we are going to using collaborative filtering, which is doing the recommendations based on similar user preferences, to predict 100,000 movie ratings for users in a subset of the original NETFLIX data. To be more specific, we are going to use the random combination between pearson similarity, jaccard similarity, user-user model and item-item model, which is 4 models in total, to test which model can doing the best prediction. Through doing this project, we are able to get familiar with the implementation about recommendations using collaborative filtering.

3. Documentation of approach

3.1 the first combination: item-item model and paerson similarity

First of all, we need to do the item co-occurrence and item similarity. In this part, we load the data from training dataset, and then map the input to be (user-id, (movie-id, ratings)) and group the result by the key, which is user-id, so we can get (user-id, list of (movie-id, ratings)). Next, for any values in the list, if any movie-id is larger than another movie-id, we gather the result to be ((movie-id(the larger one), ratings), (movie-id(the smaller one), ratings)), and then map this result to be (movie-id, (ratings 1, ratings 1 squared, ratings 2, ratings 2 squared, ratings 1*ratings 2)). Then, we group the result by key, which is user-id, so we can get (movie-id, list of (ratings 1, ratings 1 squared, ratings 2, ratings 2 squared, ratings 1*ratings 2)). After that, for any value in the list, we apply the pearson function, which is

$$P(\mathbf{x},\mathbf{y}) = \frac{\sum_{i\in O_{\mathbf{xy}}}(r_{\mathbf{x}i}-\overline{r_{\mathbf{x}}})(r_{\mathbf{y}i}-\overline{r_{\mathbf{y}}})}{\sqrt{\sum_{i\in O_{\mathbf{xy}}}(r_{\mathbf{x}i}-\overline{r_{\mathbf{x}}})^2}\sqrt{\sum_{i\in O_{\mathbf{xy}}}(r_{\mathbf{y}i}-\overline{r_{\mathbf{y}}})^2}}$$

Then, we map the result, so we can get (similarity-score, ((movie-id1, movie-id2), (movie-id2, movie-id1))). Next, we map the result second time to change the order, and we can get (movie-id1, (movie-id2, similarity-score)). Then, we map the value, to find the top N movie that has the most similar ratings. After that, we do the map again and get the result (movie-id, similarity-item).

Secondly, we do the prediction. First of all, we only pick the (movie-id, user-id) from the test dataset. After that, we join it with the result we get from the first step, and get the result (movie-id, (similarity-movie, user-id)). Then, we join it with the processed training dataset, which gives the result ((movie-id, user-id), rating). Next, we map again to get the result ((movie-id, user-id), rating). Lastly, group the result by key and get the average for the ratings.

3.2 the second combination: user-user model and paerson similarity

Mostly, the implementation is the same as the first combination. The difference is that right now, we choose to do the user-user co-occurrence and user similarity, instead of item-item co-occurrence and item similarity. Therefore, the key for the map and reduce process would be user-id, instead of movie-id.

3.3 the third combination: item-item model and Jaccard similarity

First of all, we need to do the item co-occurrence and item similarity. In this part, we load the data from training dataset, and then map the input to be (movie-id, user-id) and group the result by the key, which is movie-id, so we can get (movie-id, list of user-id). Next, using cartesian to do the pair for each line. Then, if any user-id is not the same as another

$$J(\mathbf{x}, \mathbf{y}) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

user-id, we calculate the similarity using Jaccard function, which is

Then, we map the value, to find the top N movie that has the most similar ratings. After that, we do the map again and get the result (movie-id, similarity-item).

In this implementation, the prediction part the same as the first implementation.

3.4 the forth combination: user-user model and Jaccard similarity

Mostly, the implementation is the same as the third combination. The difference is that right now, we choose to do the user-user co-occurrence and user similarity, instead of item-item co-occurrence and item similarity. Therefore, the key for the map and reduce process would be user-id, instead of movie-id.

4. Small data result and discussion

	IIP	IIJ	UUP	UUJ
MAE	1.05	0.97	1.03	0.94
RMSD	1.39	1.14	1.37	1.09

From the result, we can see that, Jaccard similarity is more efficient that the pearson similarity, since we get a smaller value for both the MAE and RMSD. And the item-item model is more efficient than the user-user model. However, we are using a small subset, which means the result we get above does not count, so we still want to try user-user model on big dataset to see if its result can be better than item-item model and how long it is going to take.

5. Big data application

In the big data part, we use the whole dataset from the the training dataset. However, when running the small data, we use only 1% of the training dataset.

One issue we noticed during cloud execution is that the number of predicted ratings is relatively small comparing to the data amount of the testing data. After scrutiny we found that this is related to the parameter N in top-N similar users. The default value of N equals 3, which won't cause significant variance when we are dealing with small dataset. Then we tried many different values of N on big dataset in order to get the largest amount of predictions while trying to achieve a better MAE and RMSD result.

6. Big data description

We implemented and executed 4 different collaborative filtering algorithms on Amazon EMR, including *Item-item model & Jaccard Similarity* **(IIJ)***, user-user model & Jaccard Similarity* **(UUJ)***, Item-item model & Pearson Correlation* **(IIP)** and *user-user model & Pearson Correlation* **(UUP)**. We changed the parameter **N,** which is the number of top-N similar users used for the prediction**,** several times to get the results under different conditions. To evaluate the accuracy of different models, we calculated the Mean Absolute Error (**MAE**) and the the Root Mean Squared Error (**RMSE**). The number of predicted ratings and execution time are recorded for each run.

7. Cloud execution

We executed the four algorithms(submitted as code) on cluster with the hardware environment remains unchanged for each turn in order to make the results comparable.

The execution environment: One master node and two core nodes, both nodes are "m4.large" configuration provided by Amazon EMR.

The hardware configuration of each instance: 4 vCore, 8 GiB memory, EBS Storage:32 GiB (EBS only).

` We use full data in the provided Netflix movie rating data subset for training and testing, and the same for each cloud execution.

After the result was successfully outputted, we then calculate the MAE and RMSD of each output by executing MAE and RMSD algorithms(submitted as code).

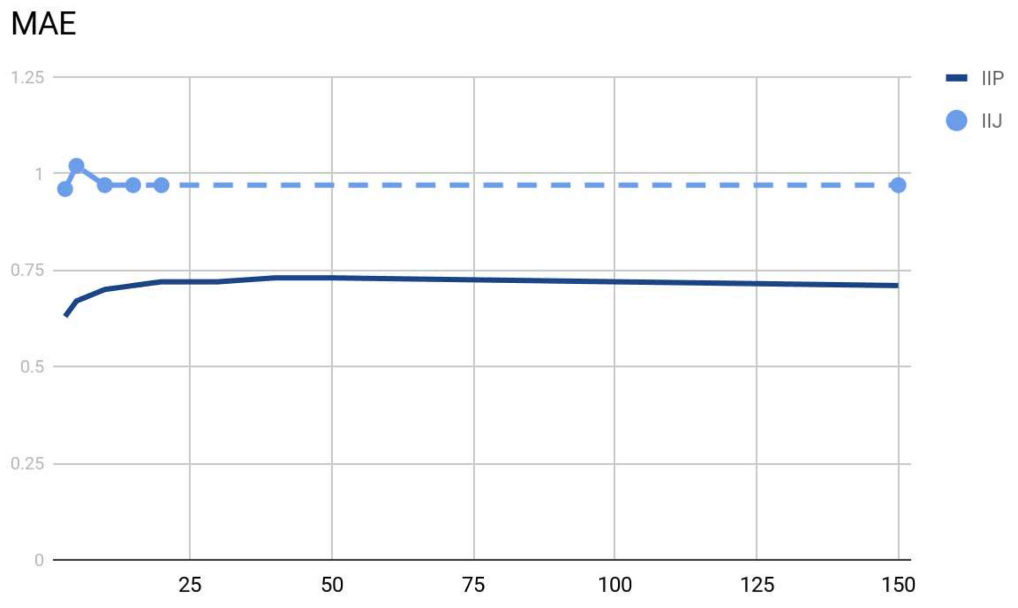
8. Big data result and discussion

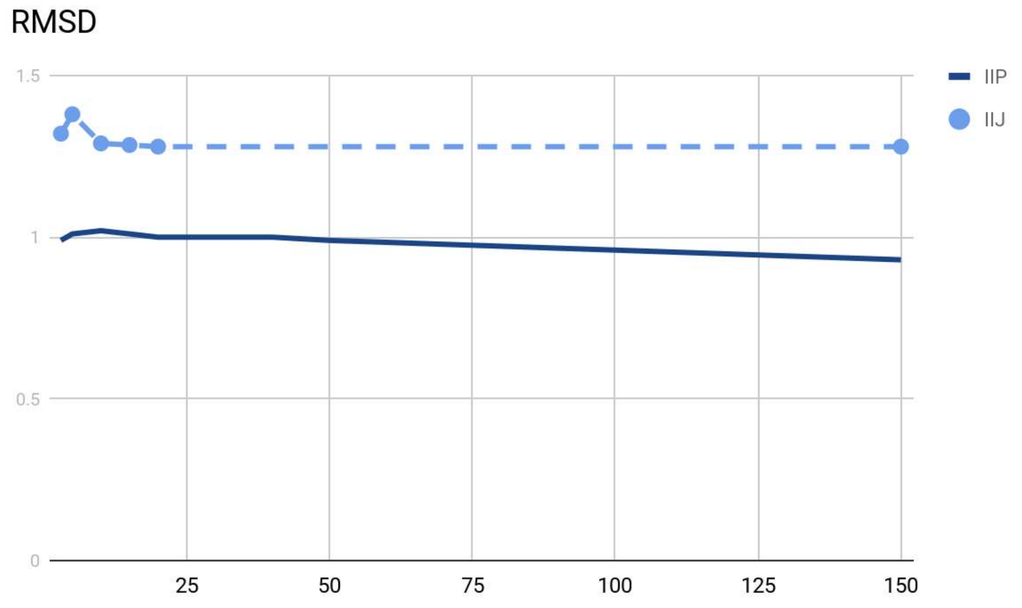
Result:

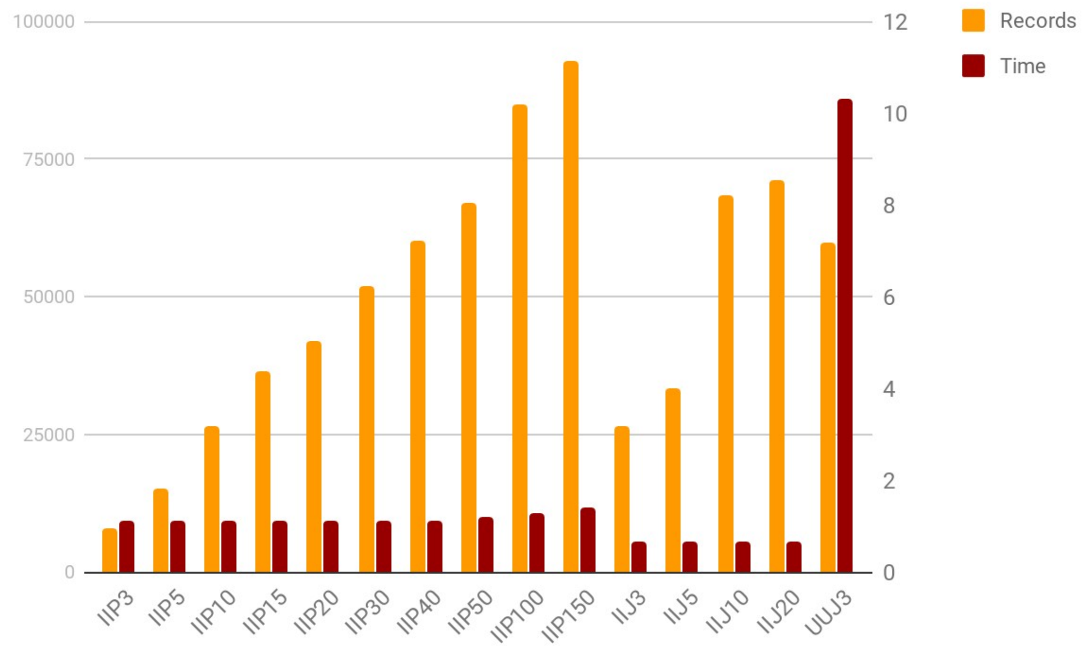
	MAE	RMSD	# of records	time
IIP3	0.63	0.99	7979	1.1h
IIP5	0.67	1.01	15205	1.1h
IIP10	0.7	1.02	26539	1.1h
IIP15	0.71	1.01	36545	1.1h
IIP20	0.72	1	41848	1.1h
IIP30	0.72	1	51829	1.1h
IIP40	0.73	1	60094	1.1h
IIP50	0.73	0.99	67011	1.2h
IIP100	0.72	0.96	84981	1.3h
IIP150	0.71	0.93	92782	1.4h
IIJ3	0.96	1.32	26413	40m
IIJ5	1.02	1.38	33420	39m
IIJ10	0.97	1.29	68254	39m
IIJ20	0.97	1.28	71294	41m
UUJ3	0.93	1.21	59847	10.3h

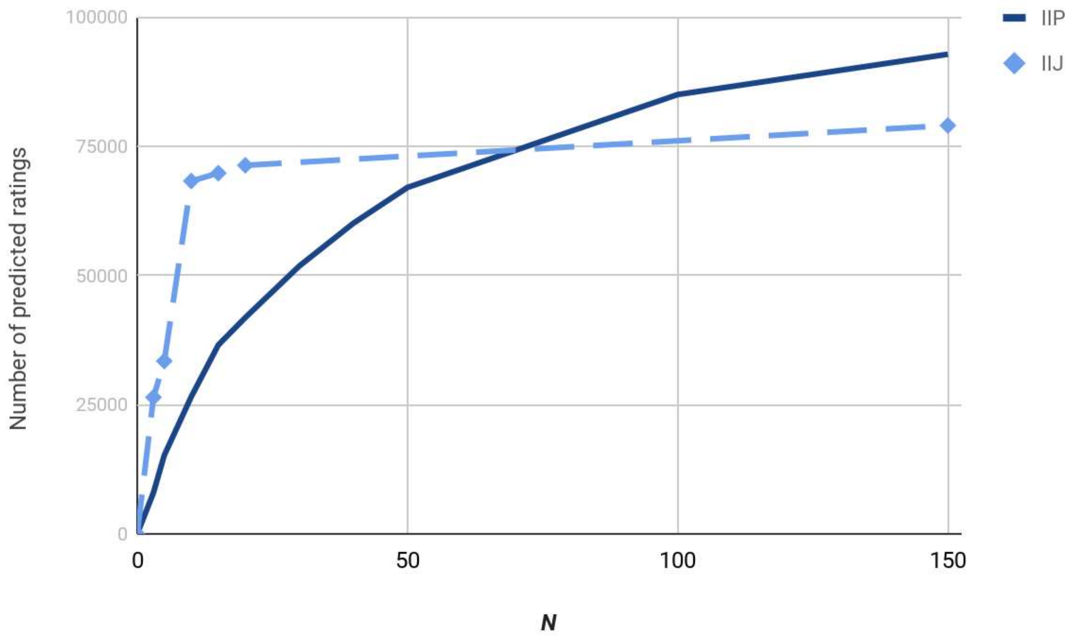
UUP	memory limited			

*The number following each model name denotes parameter N, which is the number of top-N similar users used for the prediction.






100000
75000
50000
25000
0
12
10
8
6
4
2
0
Records
Time
IIP3
IIP5
IIP10
IIP15
IIP20
IIP30
IIP40
IIP50
IIP100
IIP150
IIJ3
IIJ5
IIJ10
IIJ20
UUJ3


Number of predicted ratings
100000
75000
50000
25000
0
0
50
100
150
N
IIP
IIJ

Discussion:

IIJ(Item-item model & Jaccard Similarity)

In this case, Item-item model is more efficient than user-user model. Jaccard Similarity is more efficient and convenient than Pearson Correlation. Thus, IIJ is the most efficient approach. The process time it took proved its efficiency. Surprisingly, other than efficiency, IIJ has good performance in practice. With acceptable MAE and RMSD as well as large amount of predictions efficiently generated, IIJ may not be the best but is obviously a good approach to use.

UUJ(User-user model & Jaccard Similarity)

UUJ has similar MAE and RMSD compared to IIJ, but the process time it took is much more than IIJ, which is approximately 10 hours and 40 minutes respectively. So we threw away UUJ approach.

IIP(Item-item model & Pearson Correlation)

IIP achieves better accuracy comparing to the IIJ and the UUJ method, and it's quite effective. The number of predicting ratings seem to be small when N is small, whereas this problem can be solved by increasing N. This will not cost extra time on the run. In our experiments, when the value set for N exceeds 150, the algorithm can predict more than 90% ratings in the full testing data, and the percentage keeps increasing with the increasement of N. When N is 150, the number of records is close to 92782 with MAE 0.71, RMSD 0.93, which is the best result we have for now.

UUP(User-user model & Pearson Correlation)

Same as UUJ, UUP also has time issue. In addition, UUP has memory issue. It always failed when running on cluster due to exceeding memory limits.

9. conclusion

In this project, we have developed a Spark program to predict the missing rating value for a movie that has not been watched by a specific user using collaborative filtering. These rating values could be used by a movie recommendation system. All the data analysis and implementation is written in Spark, and the program is executed on Amazon EMR. The best result is when we use N equal to 150, we can get the result that MAE is 0.71, RMSD is 0.93, number of record is 92782 and the execution time on cloud is 1.4h. At the end, we find that Jaccard similarity is more efficient than Pearson similarity, and item-item model is more efficient than user-user model. In the future, we might try other combination of models to get a better prediction.