### **Music Recommendation System Based on Co-occurrence Matrix Multiplication and Pearson Correlation**

Shenghan Gao 8062002

Kun Gao 7555519

Dongqiao Ma 7733884

**Introduction**

Nowadays, the recommendation system is widely used in Web Application. One example for this is the Movie Online store can predict a movie the user might be interested and recommend it to user. The working principle for this recommendation system is based comparing the user history data with other users’ history data, and do the prediction based on the common preference among the users. However, how to choose appropriate items for users from a large item set is extremely challenging problem because the number of items is huge and to find an effective algorithm for recommendation is not easy. Most of the algorithms will check either millions of users or items to find good recommendations for a specific user, and this process is extremely time consuming. Such system usually requires computation for all pairs similarity (O(N^2)), which makes it impossible to build and maintain a recommendation system in a sequential program. In this project, we explore two ways to build the recommendation system. One is based on co-occurrence matrix multiplication and the other is based on Pearson correlation.

**Dataset**

The dataset used in the project is Yahoo! Music User Ratings of Musical Artists, version 1.0. The dataset contains 110,557,943 ratings of 98,211 artists by 1,948,882 anonymous users. The format of this dataset is very straight forward: UserID ArtistID Rating, which makes it very easy to parse. In this dataset, some users can rate thousands of songs while some might rate less than ten songs. Also some artists can be rated by thousands of users while some might be just rate a few times. These characteristics require the recommendation system to be able to handle the part-is-sparse and part-is-dense matrix in a fast fashion.

**Co-occurrence Matrix Multiplication**

**Reference**

Niemann, Katja, and Martin Wolpers. "A new collaborative filtering approach for increasing the aggregate diversity of recommender systems."Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013.  
**Strength**: This paper makes use of co-occurrence Item-Item matrix as well as cosine similarity to increase the accuracy of recommendation.

**Weakness**: As illustrated below, the method in this paper goes through many steps, it increases the total complexity of the whole system.

In our implementation, we add normalization of user’s rating in the co-occurrence matrix to express user’s personal rating level. And we also use eager filter and other methods to speed up the computation.

**Build Artist-Artist Similarity Matrix**

There are two steps to build Artist-Artist similarity matrix. The first step is to build the User-Artist (AU) matrix with normalization. And the second step is to transform the User-Artist (UA) matrix into Artist-Artist matrix.

**Build User-Artist Matrix**

Firstly, the input data is parsed and transformed into a User-Artist matrix directly using groupByKey() function directly. The UA matrix for the above input data is as the left table below.

Secondly, each rating in the matrix is normalized based on the user’s average rating: if the rating is above his average rating, put 1 otherwise put 0. As shown in the right table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Artist1 | Artist2 | Artist3 | Artist4 | Artist5 |
| User1 | 70 |  | 40 | 90 | 70 |
| User2 |  | 50 | 60 |  | 80 |
| User3 | 40 |  | 50 | 80 | 80 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Artist1 | Artist2 | Artist3 | Artist4 | Artist5 |
| User1 | 1 |  | 0 | 1 | 1 |
| User2 |  | 0 | 0 |  | 1 |
| User3 | 0 |  | 0 | 1 | 1 |

In order to build the Artist-Artist matrix, rating pairs are generated for each user. For every two item, if the rating is the same, put 1 otherwise put 0.

User1:  
 [((Artist1, Artist1), 1), ((Artist1, Artist4), 1), ((Artist1,Artist5),1),

((Artist3, Artists3), 1),

((Artist4, Artist1), 1), ((Artist4, Artist4), 1), ((Artist4, Artist5),1),  
 ((Artist5,Artist1),1), ((Artist5,Artist4),1), ((Artist5, Artist5),1)]

User2:

[((Artist2, Artist2), 1), ((Artist2, Artist3), 1),

((Artist3, Artist2), 1), ((Artist3, Artist3), 1),

((Artist5, Artist5), 1)]

User3:

[((Artist1, Artist1), 1), ((Artist1, Artist3), 1)),

((Artist3, Artist1), 1)), ((Artist3, Artist3), 1),

((Artist4, Artist4), 1), ((Artist4, Artist5), 1),

((Artist5, Artist4), 1), ((Artist5, Artist5), 1)]

**Build Artist-User Matrix**

The rating pairs generated from above step are reduced by summing the value up, generating the Artist-Artist co-occurrence matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Artist1 | Artist2 | Artist3 | Artist4 | Artist5 |
| Artist1 | 1 |  | 0.2 | 0.25 | 0.2 |
| Artist2 |  | 1 | 0.25 |  |  |
| Artist3 | 0.2 | 0.25 | 1 |  |  |
| Artist4 | 0.25 |  |  | 1 | 0.4 |
| Artist5 | 0.2 |  |  | 0.4 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Artist1 | Artist2 | Artist3 | Artist4 | Artist5 |
| Artist1 | 2 |  | 1 | 1 | 1 |
| Artist2 |  | 1 | 1 |  |  |
| Artist3 | 1 | 1 | 3 |  |  |
| Artist4 | 1 |  |  | 2 | 2 |
| Artist5 | 1 |  |  | 2 | 3 |

Table.1 Co-occurrence matrix Table.2 Jaccard Similarity

Each artist-artist co-occurrence in the matrix is then normalized using Jaccard Similarity function:

Then the matrix with cosine similarity: 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Artist1 | Artist2 | Artist3 | Artist4 | Artist5 |
| Artist1 | 1 | 0.045381352 | 0.35640402 | 0.49076712 | 0.42702301 |
| Artist2 | 0.045381352 | 1 | 0.46197262 |  |  |
| Artist3 | 0.35640402 | 0.4619726 | 1 | 0.043068155 | 0.034776035 |
| Artist4 | 0.49076712 |  | 0.043068155 | 1 | 0.70178466 |
| Artist5 | 0.42702301 |  | 0.034776035 | 0.70178466 | 1 |

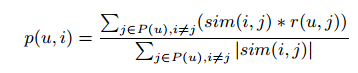
Notice that the similarity between Artist3 and Artist4, which is 0 in Jaccard Similarity, is a more accurate non-zero value in cosine similarity.

After the two major steps above, the Artist-Artist recommendation matrix is established.

**Recommendation**

Given a user history vector: Artist1 90; Artist2 40. The user vector is normalized based on its rating: Artist110; Artist2 2.

The user’s expected rating for item i is calculated as follows:



|  |  |
| --- | --- |
| Artist3 | 5.48401 |
| Artist4 | 10 |
| Artist5 | 10 |

In this way, the top N artists will be recommended to user.

**The work flow for this system:**

1. Filter out users with less than 50 ratings using filter program written in C and build User-artist matrix

2. Randomly choose a N rating records from filtered data, and use it as the training data to build the Item-Item Co-Occurrence Matrix. (Based on our observation, we find 1M rating records is already good enough to help us to do the prediction)

2. Based on average rating from each user, transfer rating score to 1 or 0. (1 means like, 0 means dislike)  
3. From each user record, pair to pair compare every two items, if this user has same attitude to those 2 items (either 1 or 0) return 1,   
4. Reduce result from step 3 from every user. -- after this step, Item to Item Co-Occurrence matrix will be generated  
5. Normalize the matrix by “Item(i)- Item(j)\_Co-Occurrence **/ (**Item(i) appease time + Item(j) appease time)” -- this step will lighten the effect from popular items

6. Multiply user purchase vector with generated Item-Item Co-Occurrence matrix, then we can produce recommendation vector.

7. (optional) In order to increase the accuracy, we can randomly choose 8 blocks of data with size of 1M rating records, then people can use it to generate a 8 Item-Item matrix. Then in order to do the recommendation, people can multiply a user rating history vector with all these 8 matrix and obtain 8 recommendation vectors respectively. At last, combine 8 recommendation vectors to build one final recommendation vector. (as matter of fact, based on our observation, all these 8 recommendation vectors which generated from 8 different matrices are almost same. This can indirectly prove the correctness of our model)

**Time Consuming, Challenge && Effect of Parallelism**

The most computation intensive is building the Artist-Artist matrix including building the co-occurrence matrix and calculating the cosine similarity for the whole matrix, since it is an all pair algorithm. This part is also I/O intensive as it loads the whole input data and generates a large amount of user-item pairs and item-item pairs. We reduce the total amount of work to speed up the computation using eager filter. Before using the entire dataset, we run a filter program to filter out users whose rating records are less than 50. And in our implementation, we do early filtering on co-occurrence matrix to filter out those item pairs whose value is less than 5. And to decrease the total amount of item pairs, we only use half of the matrix until rebuilding the matrix using cosine similarity. We also noticed that when an item is popular, it is actually unrelated to other items. if one artist is rated 100 by all users, this artist would be recommended to the user no matter what the user’s profile is. This artist, of course, can be recommended as popular item, but it can mask other recommended items based on user’s profile. Different from the paper’s expensive EPC method, in our implementation, we solve this issue by counting the occurrence of every item and use Jaccard Similarity to punish popular items. So the recommended result is more related to the user’s profile. Besides, instead of using association measures, we use a greedy filter to filter out items that are not likely to recommend to reduce the computation. The performance improvement using one core shows as below.

|  |  |  |  |
| --- | --- | --- | --- |
| Data size | Original | Eager Filter | Improvement |
| 1000 | 120s | 20s | 83% |
| 50,000 | 1 hour | 15min | 75% |

And for the recommendation, we multiply the Artist-Artist matrix with user vector only on the row associated with Artist in user vector instead of the whole matrix for fast recommendation.

Building the Co-Occurrence matrix in the method1 is really time consuming, because every items need to do pair to pair comparison. In addition, in order to build the item to item Co-Occurrence, the program need to count the times of co-similar-rating between every 2 items. This step cannot finish in one step calculation. Instead, it need to accumulate every user who have a same attitude toward these 2 items. So theoretically, the complexity for this algorithm is ItemNumber\*ItemNumber\*UserNumber. (in practical, the real computations should be much smaller than it, since most users buy much less items than 90000 items)

In order to speed up the process for computing matrix, we use parallelization(mapreduce paradigm) to improve speed. The effect from parallelization is obvious.

Improvement from parallelism (Method1: Build Co-Occurrence matrix):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data size | 1 core | 2 cores | 2 cores speedup | 4 cores | 4 cores speedup |
| 500000 | 960s | 611s | 1.57 | 412s | 2.334 |
| 1000000 | 3560s | 1701s | 2.09 | 980s | 3.632 |

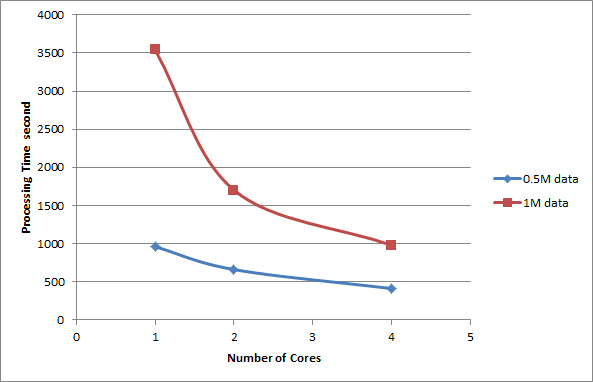


Figure1: processing time vs Number of Cores

From the table and graph above, we can see the improvement from the parallelism. The more processing data we have, the better the parallelization we can perform. Because more cores not only mean more computation units, but also means more cache resource and more IO resource. So, if the dataset is relatively large, the advantage for using parallelization will be relatively obvious.

**Pearson Correlation with Decoupling Normalization**

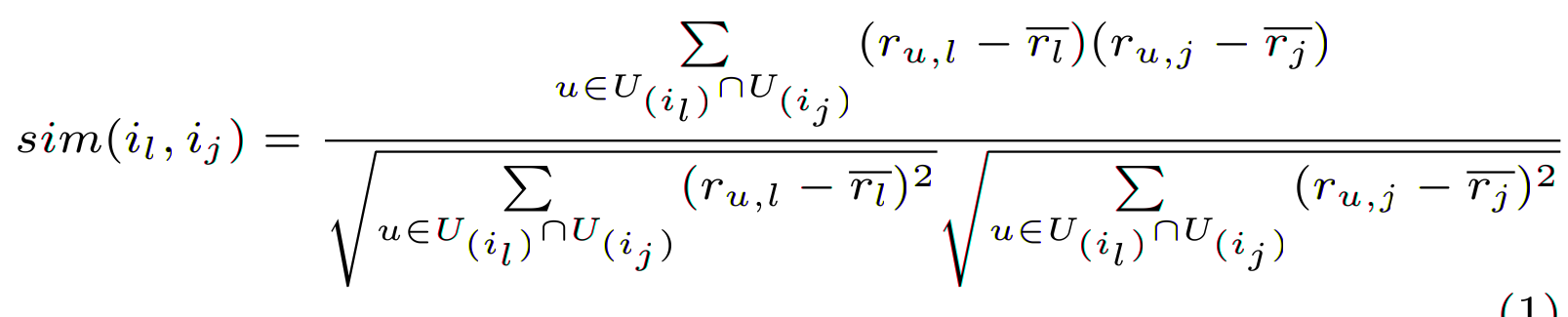
The user ratings data is normalized using decoupling normalization. In our project, we first convert ratings into rating categories and then apply the formula:

Pr(R is preferred) = Pr(Rating≤R) − Pr(Rating=R)/2

where R is a rating category.

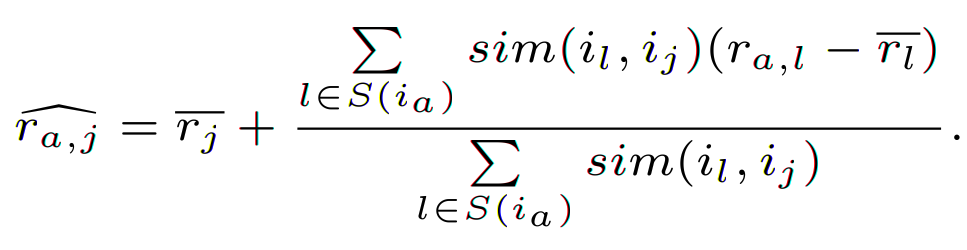
By doing this, we can change the ratings of different users with different rating standards into preferences for users and then users with similar preferences can be found out. This step ensures that the similarity calculations are not affected by different rating standards by users.

Traditional strategies are used in this method. In order to measure the similarity between two artists, Pearson correlation coefficients between every two of them are calculated and these values are used to determine the similarities between artists. Let the set of users who both rated i and j are denoted by U, Pearson correlation coefficient is calculated by the following formula:



where rl bar stands for the average rating of the artist l by all the users in U, rj bar stands for the average rating of the artist j by all the users in U. ru, l stands for the rating on artist l by user u. Notice that only the users who rate both artist l and artist j are taken into account when this similarity value is calculated. The similarity coefficient is calculated this way so that it reflects the correlation between artists.

After Pearson correlation matrix is calculated, we predict the ratings based on this matrix and the user rating vector. The formula used is:



where rj is the average rating by all the users who have rated artist j, sim(il, ij) is the Pearson correlation value between artist l and artist j, ra, l is the rating on artist l by user a.

This formula is used to predict the rating on artist j by user a. The average ratings of artists are taken into account because Pearson correlation value is not sensitive to absolute value of elements in the vector. This will be discussed further later.

Pearson correlation coefficient reflects the trend of the data, which is different from cosine similarity.

**Advantages of using Pearson correlation:**

Using Pearson correlation coefficient instead of cosine similarity takes grade inflation into account. That is, the different rating styles among users are considered. By doing this, when two users with similar preferences have different rating standards, we can still find them.

**Weaknesses of using Pearson correlation:**

When Pearson correlation is used, some problems have to be taken into account.

1. Pearson correlation value is not sensitive to absolute value of ratings.

For example, Pearson correlation between two vectors [1, 2, 3, 4, 5] and [95, 96, 97, 98, 99] is 1. If the entries of these two vectors are ratings, it cannot exactly reflect the relations between the two vectors. To solve this problem, we have to take the average value of the ratings of an artist into account when we calculate the predicted rating of an artist by a user.

2. If only one user rates the two artists between which Pearson correlation value is being calculated, Pearson correlation coefficient cannot be calculated. This is obvious if we look at the formula above. In our case, we simply say this value is 0.

3. If the ratings from all related users for an artist are identical, Pearson correlation cannot be calculated. This is obvious if we look at the formula above. When this happens, say, if all users give artist 1 ratings of 90 and artist 2 ratings of 91, Pearson correlation coefficient between these two artist cannot be calculated, in our case, we simply say this value is 0. Then it will lead to a phenomenon that the Pearson correlation value between two popular artists with similar high ratings is small. As a result, popular artists are not tended to be recommended when this strategy is used. However, this does not happen in the other system implemented.

**The workflow of the Pearson correlation-based recommendation system is:**

Offline:

1. Filter out users with less than 50 ratings using filter program written in C  
 -- User-artist matrix is generated  
2. Filter out artists with less than 50 ratings  
 -- User-artist matrix is generated  
3. Normalize ratings of the users  
 -- User-artist matrix with normalized ratings is generated   
4. Generate artist-user matrix and artist average rating vector using user-artist matrix  
5. Calculate Pearson correlation between every two artists

The average rating vector and the Pearson correlation matrix should be stored.  
Online:

The vector and the matrix is used when we want to do recommendations for a user.  
6. Normalize the user rating vector and calculate the predicted artist ratings of this user.  
Pick a number of artists (in our case, 20 of them are picked) with highest ratings.

**Computation and I/O Analysis**

The most time consuming part of this method is to build Pearson correlation matrix since we have to calculate this value between every two artists. To speed up this process, every calculation between two artists are encapsulated into a map task and these map tasks can be parallelized by using Spark. Every map task is designed in a way so that every mapper can calculate the result without communicating with other mappers in order to reduce overhead. I/O operations can be reduced by chaining RDD operations across different steps without writing or reading intermediate results to files. However, this is not beneficial if something unexpected happens and we have to start over. One optimization we have done on I/O is that when we want to store the Pearson correlation matrix, if the correlation value is 0 between two artists, we simply do not store it in the file so that we reduce the number of I/O operations.

**Parallelization Tests**

The program is written in Python and run on Spark. The following tests are done when spark.default.parallelism is configured to 8. If this parameter is configured to be too small, the performance of the program would be deteriorated. One of the common strategies used is to set spark.default.parallelism to the number of cores on the local machine when run a Spark program locally. The following table shows the results when the program runs locally.

|  |  |  |  |
| --- | --- | --- | --- |
| Data size | 1 core (s) | 2 cores (s) | Speedup |
| 1M | 544.417 | 354.356 | 1.536356094 |
| 2M | 1749.329 | 1165.215 | 1.501292894 |

From the data above, it is not hard to find that when the number of cores increase, the time spent to run the program decreases. Since the number of cores on the local machine is limited, add more threads will not benefit the performance at all.

**Reference**

Niemann, Katja, and Martin Wolpers. "A new collaborative filtering approach for increasing the aggregate diversity of recommender systems."Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013.

Wang, C. Q., et al. "Distributed collaborative filtering recommendation model based on two-phase similarity." *Future Communication, Information and Computer Science: Proceedings of the 2014 International Conference on Future Communication, Information and Computer Science (FCICS 2014), May 22-23, 2014, Beijing, China.*. CRC Press, 2015.

Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." Proceedings of the 10th international conference on World Wide Web. ACM, 2001.  
  
Linden, Greg, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering." Internet Computing, IEEE 7.1 (2003): 76-80.  
  
Jin, Rong, and Luo Si. "A study of methods for normalizing user ratings in collaborative filtering." Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2004.  
  
Jin, Rong, et al. "Collaborative filtering with decoupled models for preferences and ratings." Proceedings of the twelfth international conference on Information and knowledge management. ACM, 2003.  
  
Kim, Taek-Hun, et al. "An improved recommendation algorithm in collaborative filtering." E-Commerce and Web Technologies. Springer Berlin Heidelberg, 2002. 254-261.