Research on Collaborative Filtering Algorithm Based on Slope One and User Similarity and Parallelization Implementation

For Big Data Theory and Applications

Luo Xinyu[a]

[a]*Northwestern Polytechnical University, Shaanxi, Xian*

May, 2022

Abstract

The rapid development of Internet technology has led to the era of big data, and the explosive growth of data volume also makes the problem of "information overload" increasingly serious, how to quickly and efficiently filter out useful information from the excessive information has become a problem that people are currently troubled, thus, the recommendation system was born. Traditional recommendation systems work on the principle of using past data to discover the interest and preference characteristics of different users, and rely on this function to make personalized recommendations for users, but when faced with large amounts of data, traditional recommendation algorithms take a lot of time and may not be able to meet user needs. In this situation, the implementation of distributed Spark technology can effectively solve this technical defect. In addition, traditional recommendation algorithms have exposed practical problems such as sparsity, cold start, and poor scalability in their applications.

In this paper, we analyze collaborative filtering recommendation algorithms and summarize the advantages and disadvantages of various algorithms. The Slope One algorithm uses a linear regression model to solve the data sparsity problem. The core of the algorithm is to combine the similarity between users and Slope One algorithm for rating prediction. First, the Slope One algorithm is used to predict the ratings of some users on items and populate the rating matrix, which solves the data sparsity problem and also makes the prediction result one of the final results. On the populated user-item rating matrix, the similarity of users is calculated, and the similarity ranking is used to obtain the neighborhood set s composed of the first n neighbors, followed by the recommendation ranking of all items in s to the target user to obtain the topN recommendation set.

In addition this paper also uses user similarity based analysis to optimize the data and quality of user ratings involved in prediction. The screening of k-nearest neighbors of currently active users is added to ensure that only users whose ratings are similar to the evaluation criteria of currently active users are involved in the recommendation. The ratings of the selected k-nearest neighbors are then used to calculate the deviation of the target item from other items, which means that fewer but higher quality ratings help the currently active users to predict their ratings of the target item.

To enhance the ability of the improved algorithm to handle massive data, this paper implements Slope One, similarity algorithm and improved collaborative filtering recommendation algorithm on Spark platform.

keywords: User-based Collaborative Filtering, Slope One, Modified Cosine Similarity, Spark

1. Introduction

Big data contains rich value and huge potential, which will bring transformative development to human society, but also brings serious "information overload" problem, how to quickly and effectively obtain valuable information from the complex data has become a key problem in the current development of big data. As an effective method to solve the problem of "information overload", recommendation system has become a hot spot in academia and industry, and has been widely used, resulting in many relevant research results. Based on the user's needs and interests, the recommendation system extracts the items of interest (such as information, services, goods, etc.) from the massive data through the recommendation algorithm, and recommends the results to the user in the form of personalized lists.

Although the recommendation algorithms used in personalized recommendation technology are different, the recommendation based on collaborative filtering algorithm has become the most widely used personalized recommendation [1] algorithm

due to its simplicity, accuracy and effectiveness. Data sparsity has always been a major problem for collaborative filtering algorithms, and scholars at home and abroad have conducted a lot of research on it. Leng [2] et al. argue that the data sparsity problem can adversely affect collaborative filtering in terms of inaccurate nearest neighbor search and too few nearest neighbor ratings. The problem of data sparsity and cold start in recommendation systems is addressed by Meng Xiangwu [3] et al. who provide a comprehensive review of social recommendation systems in terms of trust inference and key recommendation technologies. Vozalis [4] et al. integrate singular value decomposition and item-based methods into collaborative filtering to reduce the size of user item matrix and thus effectively alleviate the problem of data sparsity.

In this paper, a hybrid user-based and rating-based collaborative filtering recommendation algorithm is proposed to address the shortcomings of existing traditional algorithms.The Slope One algorithm uses a linear regression model to solve the data sparsity problem. The core of the algorithm is to combine the similarity between users and Slope One algorithm for rating prediction. First, the Slope One algorithm is used to predict the ratings of users by some users and populate the rating matrix, which solves the data sparsity problem and allows the prediction results to be used as one of the final results. Then, the original dataset and the predicted dataset are combined to form a new dataset, and the similarity between users and the average preference difference value matrix are calculated. Then, the similarity between users is used as the weights of the Slope One algorithm for rating prediction. Finally, the two parts of the results are combined in different proportions by parameter training to form the final predicted scores.

In addition this paper also uses user similarity based analysis to optimize the data and quality of user ratings involved in the prediction. The screening of k-nearest neighbors of currently active users is added to ensure that only users whose ratings are similar to the evaluation criteria of currently active users are involved in the recommendation. The ratings of the selected k-nearest neighbors are then used to calculate the deviation of the target item from other items, which means that fewer but higher quality ratings help the currently active users to predict their ratings of the target item.

2. Algorithm Description and Model Design

In this section, the detailed ideas involved in the improvement of this algorithm and other basic algorithms involved in this algorithm will be presented.

2.1. Related Algorithm Introduction

2.1.1. Slope One Algorithm

The Slope One algorithm is a scoring-based [5] prediction algorithm. Unlike general item-based algorithms, the algorithm does not calculate the similarity between items, but uses a minimal linear regression model for prediction, which is easy to implement, fast to compute, scalable, and good for data sparsity [6].

The Slope One algorithm is used in many online recommendation systems, such as Hitflip DVD recommendation system, inDiscover MP3 recommendation system, Value Investing News stock news website, AllTheBests shopping recommendation engine, etc.

The Slope One algorithm is a classical algorithm based on rating prediction, which is based on linear regression. The predictor takes the form of $f(x) = x + b$, and the free parameter b is the average deviation of the user's rating of the two items.

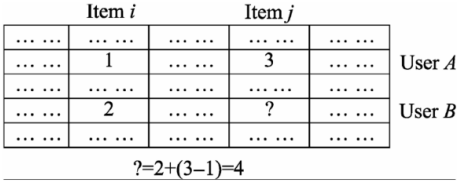

Figure 1: Slope One Interpretation

As shown in Figure 1, the predicted score of user B for item j is $r_{Bj} = r_{Bi} + (r_{Aj} - r_{Ai})$. In many instances, it is more accurate and faster than the usual linear regression method $f(x) = ax + b$, and the algorithm requires only half (or even less) the storage capacity.

2.1.2. KNN Algorithm

K-Nearest Neighbor (KNN), an algorithm commonly used for classification, is one of the simpler classical machine learning algorithms supported by a mature theory. The basic idea of this method is that if most of the k most similar (i.e., k-nearest neighbors in feature space) samples of a sample to be classified belong to a certain category, the sample also belongs to this category, i.e., the one who is close to the vermilion is red and the one who is close to the ink is black. Obviously, the classification of the current sample to be classified requires the support of a large number of samples known to be classified, so KNN is a supervised learning algorithm [7].

k The selection of nearest neighbors usually relies on similarity measures. The main similarity

measures include cosine similarity, correlation similarity, and modified cosine similarity. Since the cosine similarity measure does not consider the scoring scale of different users, the correlation similarity is more often used to measure the similarity between items. In this paper, **modified cosine similarity** is used to calculate the similarity between users.

2.1.3. User-based Collaborative Filtering Algorithm

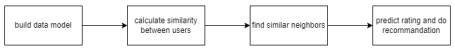

Figure 2: User-based Collaborative Filtering Algorithm Process

User-based collaborative filtering recommendation looks for similarity among users, and it is based on the assumption that users who rate the same items similarly may have the same preferences. Its basic idea is to use the rating matrix to find highly similar user groups and then make recommendations for target users based on the preferences of similar users. Its working process is shown in Fig 2.

The following is a simple example to illustrate the basic principle of the UserCF algorithm.

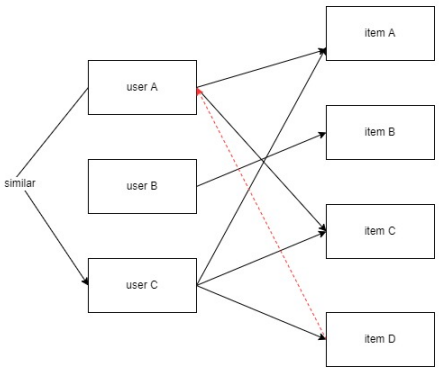

Figure 3: User-based Collaborative Filtering Algorithm Example

As shown in Figure 3, user A prefers items A and c, user b prefers item B, and user C prefers items A, C, and D. According to the user-based collaborative filtering algorithm, user A and user C jointly prefer items A and C. It is presumed that user A and user C are similar, i.e., the tastes of the two users may be the same, so item D, which is preferred by user C, is recommended to user A.

If the rating data set is relatively small, the user-based collaborative filtering recommendation algorithm will encounter the problem of data sparsity. Also, because analysis and comparison can become very complex, the user-based collaborative filtering recommendation algorithm is not applicable to a large number of users and items.

The user-based collaborative filtering algorithm pseudo-code is shown in Algorithm 1.

Algorithm 1 User-based Collaborative Filtering Algorithm

Assuming that the target user is u, the pseudo-code of the user-based collaborative filtering recommendation algorithm is shown below
for Every other user w **do**
  Calculate the similarity s of user u and user w Sort them in descending order of similarity and take the top n users as the domain
end for
for Each item a for which the neighboring user has a preference and user u has no preference **do**
  for Each of the neighboring users who have a preference for a v **do**
    Calculate the similarity s between user u and user v s as weights to calculate the prediction score
  end for
end for

2.1.4. Similarity Calculation

In this paper, similarity is calculated using modified cosine similarity to calculate the similarity between users [8]. The cosine similarity does not take into account the user rating scale problem, e.g., in the case of rating interval [1, 5], for user A, a rating of 3 or more is his favorite, while for user B, a rating of 4 or more is his favorite. The modified cosine similarity measure improves the above problem by subtracting the average user ratings of the items.

Let I_i and I_j denote the set of items rated by user u1 and u2, respectively, the set of items jointly rated by user U_i and user U_j as $I_{ij} = I_i \cap I_j$, r_{ij} denotes the rating of item r_j by user u_i, and $\bar{r_i}$ and $\bar{r_j}$ denote the average ratings of user u_i and user u_j, respectively. Then the similarity $sim(u_i, u_j)$ of useru_i and user u_j home is

$$\operatorname{sim}(u_i, u_j) = \frac{\sum_{i_c \in I_{ij}} (r_{ic} - \bar{r}_i)(r_{jc} - \bar{r}_j)}{\sqrt{\sum_{i_c \in I_{ij}} (r_{ic} - \bar{r}_i)^2} \sqrt{\sum_{i_c \in I_{ij}} (r_{jc} - \bar{r}_j)^2}} \tag{1}$$

2.1.5. Spark Distributed Platform

Although the MapReduce programming model can handle large amounts of data, show good parallelization and processing efficiency, and provide data

fault tolerance and reliability, because MapReduce is disk-based [9], the results of the map phase will fall to disk, reduce will pull the data it needs from disk, and the results of the reduce phase will be written to disk. The Spark programming model is a solution to the bottleneck problem of MapReduce in fast computing scenarios.

The Spark programming model is a distributed computing framework and can operate on memory, which makes Spark ideal for running complex algorithms such as machine learning and more diverse and complex arithmetic. The most important data structure of Spark is RDD [10], which is a kind of resilient and distributed data set. The basic flow of Spark operation is shown in Figure 4. The Cluster Manager is the soul, responsible for scheduling, and the Appliaction (the Spark program written by the user) in the figure, consists of a driver module and several executor modules, the driver module owns the contextual environment of Spark and can perform the scheduling of tasks, the executor module is independent of the application (SparkContext), i.e., it is our worker nodes, which execute the tasks of the application.

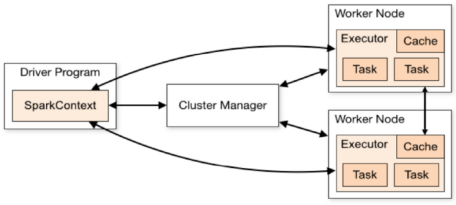

Figure 4: Spark Working Process

Basic Properties of RDD

Resilient Distributed Datasets (RDD) is an abstraction of distributed memory, which is an efficient, highly constrained and fault-tolerant shared memory model, and the basic unit of Spark computation. RDDs have several characteristics.

RDDs are read-only in memory, and can only be created by other RDDs through deterministic operations called transformation, which include map, flatMap, join, filter, and groupBy.

RDDs do not need to be materialized; RDDs maintain an internal RDD transformation relation graph, so that when an RDD is lost or erroneous, the lost RDD partition can be calculated from the physical data through the relation graph, instead of reconstructing the lost data partition through checkpoints.

RDDs can also be cached. By calling cache and persist functions, intermediate results or frequently used data can be cached in memory, thus saving the time of reading intermediate results from disk.

2.2. Optimized User and Slope One Based Collaborative Filtering Algorithm Design

The Slope One algorithm uses a linear regression model to solve the data sparsity problem. The core of the algorithm is to combine the similarity between users and the Slope One algorithm for rating prediction. First, the Slope One algorithm is used to predict the ratings of items by some users and populate the rating matrix, which solves the data sparsity problem and also allows the prediction results to be used as one of the final results. On the populated user-item rating matrix, the similarity of users is calculated, and the similarity ranking is used to obtain the neighborhood set s composed of the first n neighbors, followed by the recommendation ranking of all items in s to the target user to obtain the topN recommendation set.

In addition this paper also uses user similarity based analysis to optimize the data and quality of user ratings involved in prediction. The screening of k-nearest neighbors of currently active users is added to ensure that only users whose ratings are similar to the evaluation criteria of currently active users are involved in the recommendation. The ratings of the selected k-nearest neighbors are then used to calculate the deviation of the target item from other items, which means that fewer but higher quality ratings help the currently active users to predict their ratings of the target item.

The recommendation process of traditional collaborative filtering algorithm is generally divided into four steps, shown in Fig 5: collecting user rating data, user-item rating matrix construction, item-based or user-based finding the proximity set, and final recommendation result topN recommendation set. In this paper, the improvement work is also mainly between steps 2 and 3, using the KNN-optimized slope one algorithm to do vacancy value filling, so as to construct a less sparse rating matrix.

2.2.1. Optimized Slope One Algorithm

The basic Slope One prediction algorithm is: calculate the average deviation dev_{jk} of the target item I_j from other items I_k; predict the rating $P(u)_j$ of the target item I_j by the currently active user u. In the following equation, I_{jk} is the set of users who have rated item I_j, I_k; u_j denotes the rating of item I_j by user u; R_j is the set of items that have been rated at the same time as item i_j; card(S) denotes the number of elements in the set S.

$$dev_{jk} = \sum_{u_i \in I_{jk}} \frac{r_{ij} - r_{ik}}{\text{card}\left(I_{jk}\right)} \tag{2}$$

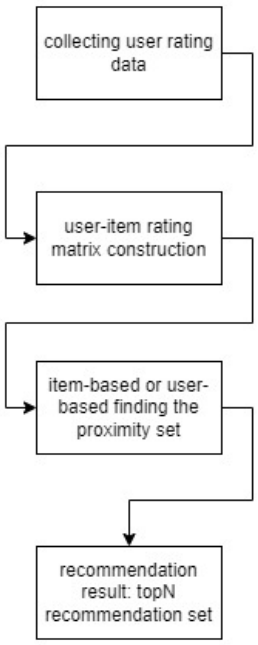

Figure 5: Regular User-Based CF Process

$$P(u)_j = \frac{\sum_{k \in R_j} \left(\text{dev}_{jk} + u_k\right)}{\text{card}\left(R_j\right)} \tag{3}$$

And the optimization with knn is mainly to filter Ijk in the formula and select the user composition Ijk with k most similar to user u based on the similarity calculation.

Process

First, the scoring matrix is initialized. Calculate the user similarity matrix based on the user-item matrix, and then select the k users with the greatest similarity for u as the set of k nearest neighbor users KNN(K) based on the similarity of u to other users. Then calculate the possible rating values of user u for item i.

Process Description

Input: user-item matrix $R(m * n)$, currently active user u, target item

Output: user u predicted rating $P(u)$ for item I_j

1. calculate the similarity matrix $S(m * m)$ of users based on $R(m * n)$

2. select the k users with the greatest similarity for u as the set of nearest neighbor users KNN(n) based on the similarity of u to other users

3. foreach u evaluated items i_k(k!=j)do

4. dev=r_{uj}-r_{uk}

5. foreach v belongs to KNN(n) do

6. if user v has evaluated items i_j and i_k

7. $dev = r_{vj} - r_{vk}$

8. $dev/ = card(I_{jk})$

9. $p(u) = dev + u_k$

10. $p(u)/ = card(R_j)$

2.2.2. User-based Collaborative Filter Algorithm

On the padded user-item rating matrix, the similarity between users is continued to be calculated using the modified cosine similarity, and the similarity ranking is obtained for the top n neighbors to form the neighborhood set S. Then the recommendation rate of all items in S to the target user is sorted to obtain the set of recommendation results consisting of the top n items.

recommendation rate is calculate by:

$$p(u,i) = \sum_{v \in S(u.K) \cap N(i)} W_{u,v} r_{v,i} \tag{4}$$

Process Description

1. Run the Slope One algorithm based on the user-item matrix R to predict the user ratings of some items (fill in the blanks).

2. Combine the original dataset and the predicted dataset into a new dataset to obtain the new user-item rating matrix R.

3. Based on the new rating matrix, calculate the similarity matrix S using the modified cosine similarity

4. Assume that to make a recommendation to user u, the set of users KNN (k) with the highest similarity to u is taken out from the similarity matrix S

5. From Equation 4, calculate the degree of interest of user u in the items purchased by the k users most similar to him, rank them, and take TOP N set as the recommendation.

2.3. Parallelization Implementation

Since this Improved User-Based Collaborative Filtering algorithm requires iterative computation of data in the computation process, Spark's RDD mechanism can fit this need exactly. Therefore, the RDD transformation in this paper is designed to parallelize the Spark-based user-based collaborative filtering algorithm, and the RDD caching feature in Spark is used to store some intermediate results that are extremely computationally consuming, and the Spark broadcast variable mechanism is used to reduce the network data transmission during the computation. The scheme is as follows.

2.3.1. Spark configuration and data source reading

Spark driver will read the relevant configuration file to generate SparkConf object, and then generate SparkContext object based on this object to

connect to Spark cluster. In the process of computation, one RDD partition will generate one computation task. If the number of RDD partitions does not match the computational resources allocated to the program by Spark cluster, the parallelized computation efficiency of Spark will be reduced. Therefore, the number of partitions of the source RDD needs to be reasonably adjusted.

2.3.2. Multi-node parallelized execution of collaborative filtering algorithms

Based on the above analysis, the execution process of the user-Based collaborative filtering algorithm is divided into three major parts: the similarity calculation between users and the Slope One-based padding and update module and the user-based collaborative filtering module.

(a) similarity calculation between users

1. This algorithm needs to involve the history of user ratings of items, which requires three fields: userId, itemId, and user rating rate of items. Therefore, the original data is preprocessed to generate a source file containing these three fields. Then the source file is read, field sliced, and converted into an RDD_{source} consisting of tuples of the format (userId, itemId, rate).

2. The calculation of inter-user similarity using modified cosine similarity involves the average rating of each user. Therefore, after obtaining the useId and rate fields in the RDD_{source} and aggregating the tuples with the same userId using the reduceByKey operation, the avgRateCalculate (), a custom average score calculation method, is used to calculate the average user rating and store it in the $RDD_{avgRate}$. The $RDD_{avgRate}$ consists of a tuple of the form (userId, avgRate).

   As a distributed computing framework, Spark sends the same variable for multiple parallel operations by default, which leads to a large amount of data transmission on the network during the computation and affects the computation efficiency. To address this problem, we adopt Spark's broadcast variable mechanism to improve the computation efficiency, and encapsulate the data into a broadcast variable to be saved and sent to each computation node only once, so that the data can be retrieved directly from the local node when it is needed by subsequent computation nodes without relying on network transmission. Since the data in $RDD_{avgRate}$ will be repeatedly queried during the subsequent computation, the data in $RDD_{avgRate}$ is first aggregated from each worker node to Driver by the collect operation, transformed into Map collection $Map_{avgRate}$ by the toMap operation, and then broadcasted to each worker node as a broadcast variable.

3. Get the userId, itemId, rate fields in RDD_{source}, and connect the userId and rate with "-" symbol to form "user-rating " field. Then use the reduceByKey operation to aggregate the tuples with the same itemId together to form the $RDD_{histRate}$ which stores the user history rating information. the RDD consists of the format (itemId, userId1-rate1, userId2-rate2, ..., userIdN- rateN), and N is the number of users who have rated the item of itemid. Since the subsequent rating calculation also involves querying the user's historical rating, the toMap operation is used to transform it into $Map_{histRate}$, and then encapsulate it as a broadcast variable and distribute it to each worker node.

4. Step4: Based on the $RDD_{histRate}$, for each itemId, the corresponding set composed of "userId-rate" fields is merged into a Cartesian product. First, the set of fields in the format (itemId, userId1- rate1, userId2-rate2 , ..., userIdN- rateN) into a list of tuples of the form (userIdi- userIdj, ratei- ratej), $i, j \in [1, N]$using a map operation. The tuple is transformed into a list of tuples of the form (userIdi - userIdj, ratei- ratej) , $i, j \in [1, N]$. Then, we use flatMap operation to extract the elements from the List set, and then use reduce ByKey operation to aggregate them, and finally get the summary of ratings of two users who review an item at the same time, and store them in $RDD_{rateSum}$. $RDD_{rateSum}$ is composed of tuple of the format (userIdi-userIdj, ratei- ratej), $i, j \in [1, N]$. ratei- ratej), $i, j \in [1, N]$

5. With $RDD_{rateSum}$ and $Map_{histRate}$ as parameters, the scoring is calculated using a custom function similarityCalculate() to finally obtain the result $RDD_{similarity}$ containing the similarity among all users. the RDD consists of tuples of the form (userIdi - UserIdj, Sim(i, j)), $i, j \in [1, N]$, where M is the total number of users. Since the calculation of similarity consumes a lot of resources, the caching mechanism of RDD in Spark is adopted to cache the $RDD_{similarity}$ to avoid the recalculation due

to data loss in the subsequent computation. Since the data in the

$RDD_{similarity}$ is also repeatedly queried during the subsequent computation, the $RDD_{similarity}$ is also encapsulated as a broadcast variable and sent to each worker node.

(b) the Slope One-based padding and update module

1. Rate all for rating module of user u. Based on the $RDD_{similarity}$, using the filter operation, the tuple whose key value contains user u in the $RDD_{similarity}$ is filtered out, converted into a list set, sorted from largest to smallest according to the similarity value, and finally the top K values of the sorted List are taken to form the nearest neighbor KNNu of user u.

2. Using KNNu as well as $Map_{histRate}$ as input parameters, the function SlopeOne() is used to predict the fill of the unrated item i for each user u. The algorithmic idea of this function is given in 2.2.1, and the core formulas are Eq. 2 and Eq. 3. This results in a populated rating matrix.

3. The similarity tuple $RDD_{padsimilarity}$ and the rating history tuple $Map_{padhistRate}$ after populating are again calculated following the procedure described in the similarity calculation module.

(c) the user-based collaborative filtering module

1. Rate all the for rating modules of user u. Based on $RDD_{padsimilarity}$, the tuple whose key value contains user u in $RDD_{padsimilarity}$ is filtered out using the filter operation, converted into a list set, sorted from largest to smallest according to the similarity value, and finally the first K values of the sorted List are taken to form the nearest neighbor KNNu of user u.

2. With KNNu and $Map_{padhistRate}$ as uninput parameters, the prediction score is calculated using the self-defined function rateCalculate(). This function uses Equation 4 as the core of the calculation.

3. Experiments and Results Analysis

3.1. Dataset

The dataset for this experiment is the MovieLens dataset provided by the GroupLens group, which contains three kinds of datasets of 100K, 1M, and 10M sizes, recording 100,000, 1 million, and 10 million user rating records, respectively. The dataset of size 100K is used in this experiment. It mainly consists of four files: movies.dat, rating.dat, users.dat, README, where users.dat records all user-related information, movies.dat records all movie-related information, and rating.dat records user-movie rating information and time stamp. Eighty percent of the rating dataset is used as the training set for training the model, and the remaining 20% is used as the test set. Cross-tabulations are used to avoid the influence of randomness and chance on the experimental results.

3.2. Experimental Environment

OS	Ubuntu20.04
Spark	3.2.1
JAVA	11
Python	3.9
Hadoop	3.3

Table 1: Experimental environment

3.3. Result and Analysis

3.3.1. MAE

The ways to evaluate the quality of recommendation results are mainly divided into two types: statistical accuracy methods and decision support accuracy methods. Among them, Mean Absolute Error (MAE) is a widely used prediction accuracy evaluation index due to its simple calculation method and easy to understand. MAE measures the accuracy of prediction results by calculating the deviation between the actual user ratings and the predicted user ratings. In order to objectively evaluate the prediction accuracy of the improved algorithm in this paper, MAE is used as an evaluation index. Its calculation formula is:

$$MAE = \frac{\sum_{u,i \in T} |r_{ui} - \overline{r_{ui}}|}{|T|} \tag{5}$$

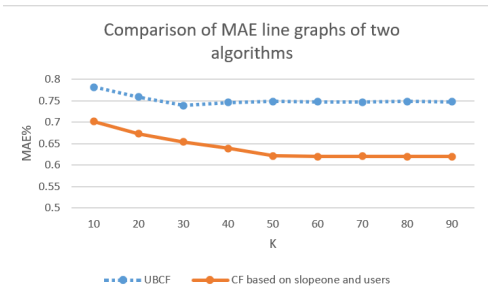

Figure 6: Comparison of MAE line graphs of two algorithms

Figure 6 shows the trend of MAE with k for the UBCF algorithm and the algorithm improved in this paper. It can be seen from the figure that the mean deviation of the UBCF algorithm is larger, while the MAE of the algorithm with improved matrix filling by slope one is smaller. Also it can be seen that the MAE of the improved algorithm in this paper decreases as k increases when k ¡ 80. When k is greater than or equal to 80, this MAE value tends to be smooth.

3.3.2. Speedup

The speedup, which is the ratio of the time consumed by the same task running in a uniprocessor system and a parallel processor system, is used to measure the performance and effectiveness of parallel systems or program parallelization.

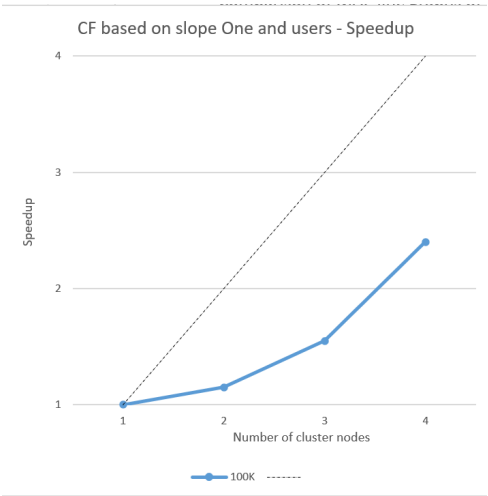

Figure 7: Speedup

The speedup curve from this improved user-based collaborative filtering algorithm is essentially linear, Fig 7 And as the size of the dataset increases, the speedup of the algorithm is closer to the ideal value. At the same time, for the same dataset, the speedup growth at 3 nodes is faster than that at 2 nodes, which makes the speedup growth faster because the two-node spark cluster includes a Master node and a Worker node, and the Master node is responsible for the task allocation and resource scheduling of the cluster in addition to the computational tasks.

4. Conclusion

The predicted values of the recommendation list and test set are given with the padding user-item rating matrix, and compared with other collaborative filtering recommendation algorithms. The results show that this algorithm can alleviate the data sparsity problem and push up the recommendation quality of the recommendation system. The parallelized implementation of the algorithm is also investigated, and the execution speed and efficiency of the algorithm are significantly improved with the help of spark platform, which also verifies that the algorithm has good speedup.

References

[1] 项亮· 推荐系统实践［M］· 北京: 人民邮电出版社，2012· (Xiangliang· Recommender system practice［M］· eijing: The Peo-ple's Posts and Telecommunications Press，2012·)

[2] 冷亚军,陆青,梁昌勇.协同过滤推荐技术综述[J].模式识别与人工智能,2014,27(08):720-734.DOI:10.16451/j.cnki.issn1003-6059.2014.08.005.

[3] 孟祥武,刘树栋,张玉洁,胡勋.社会化推荐系统研究[J].软件学报,2015,26(06):1356-1372.DOI:10.13328/j.cnki.jos.004831.

[4] M. G. Vozalis and K. G. Margaritis, "Applying SVD on item-based filtering," 5th International Conference on Intelligent Systems Design and Applications (ISDA'05), 2005, pp. 464-469, doi: 10.1109/ISDA.2005.25.

[5] 林德军. 基于Slope One改进算法推荐模型的设计与实现[D].北京邮电大学,2013.

[6] 柴华,刘建毅.一种改进的slope one推荐算法研究[J].信息网络安全,2015(02):77-81.

[7] 耿丽娟,李星毅.用于大数据分类的KNN算法研究[J].计算机应用研究,2014,31(05):1342-1344+1373.

[8] 张瑞典,钱晓东.用余弦相似度修正评分的协同过滤推荐算法[J].计算机工程与科学,2020,42(06):1096-1105.

[9] 吴信东,嵇圣.MapReduce与Spark用于大数据分析之比较[J].软件学报,2018,29(06):1770-1791.DOI:10.13328/j.cnki.jos.005557.

[10] 张明敏. 基于Spark平台的协同过滤推荐算法的研究与实现[D].南京理工大学,2015.