Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 3, March 2019, pp. 930-938
DOI: 10.23940/ijpe.19.03.p22.930938

Collaborative Filtering Recommendation Algorithm based on Spark

Jinhong Tao[a], Jianhou Gan[b], and Bin Wen[a,*]

[a]*School of Information Science and Technology, Yunnan Normal University, Kunming, 650500, China*
[b]*Key Laboratory of Educational Informatization for Nationalities of Ministry of Education, Yunnan Normal University, Kunming, 650500, China*

Abstract

With the advent of the era of big data, the problem of information overload has become particularly serious. The recommendation system can provide personalized recommendation services for users by analyzing users' basic information and users' behavior information. How to push information accurately and efficiently has become an urgent issue in the era of big data. Based on the Alternating Least Squares (ALS) collaborative filtering recommendation algorithm, this paper reduces the loss of the invisible factor item attribute information by merging the similarity of the item on the loss function. At the same time, the cold start strategy is introduced into the model to complete the recommendation. The algorithm is implemented on the Spark distributed platform and single node by using the Movie Lens dataset published by the GroupLens Lab. The experiment results show that the proposed recommendation algorithm can preferably alleviate the data sparsity problem compared with the traditional recommendation algorithm. Moreover, the algorithm improves the accuracy of recommendation and the efficiency of calculation.

Keywords: recommendation system; collaborative filtering; matrix decomposition; spark

(Submitted on November 16, 2018; Revised on December 15, 2018; Accepted on January 11, 2019)

1. Introduction

The popularity and rapid development of big data provide users with a large amount of information to meet the information needs of users. However, how to quickly and effectively obtain useful information for users in massive information has become an urgent problem to be solved. An improvement in this problem has been aroused as a personalized recommendation system emerged. Common personalized recommendation systems include collaborative filtering, content filtering, and social recommendation systems [1]. Among them, the recommendation system based on collaborative filtering is mature, effective, and widely used [2]. Collaborative filtering technologies usually include user-based collaborative filtering [3], item-based collaborative filtering, and hybrid collaborative filtering [4-5]. These methods are usually generated by using item similarity or user-like reading [6]. With the increasing number of users and items, collaborative filtering algorithms face serious data sparsity and data size scalability issues.

Aiming at the problem of data sparsity, many researchers have proposed improved methods for collaborative filtering from different perspectives. For example, the data sparsity problem was studied in the perspective of similarity propagation [7], but the algorithm has a higher complexity. Time and space overhead are large. Xu et al. proposed a method of integrating the trust relationship of social network friends, effectively alleviating the sparseness of data [8]. In order to reduce the sparsity of the matrix, Jing et al. used the singular value decomposition (SVD) method to reduce the dimension of the scoring matrix to lower the sparseness of the matrix [9], but the computation of this method is large. Clustering techniques are used to reduce sparsity. For example, Wang et al. proposed to use the k-means algorithm for item-based clustering and then calculated the global similarity of nearest neighbor users based on item clustering [10]. Chun et al. proposed a collaborative filtering recommendation algorithm based on hierarchical clustering, which effectively solved the data sparsity problem by using the hierarchical clustering smooth filling method [11].

Aiming at the scalability problem of the traditional collaborative filtering algorithm in dealing with large-scale data,

* Corresponding author.
E-mail address: wenbin@ynnu.edu.cn

many researchers use distributed computing to implement collaborative filtering algorithms to improve the scalability of the algorithm. Zhao et al. implemented a user-based collaborative filtering algorithm on the Hadoop distributed processing platform [12]. In the paper, the improved item-based collaborative filtering recommendation algorithm is parallelized under the Hadoop platform [13]. Lin et al. proposed a hybrid collaborative filtering recommendation algorithm based on the Hadoop platform k-means and slope one [14]. Kupisz et al. implemented item-based collaborative filtering algorithms on Hadoop and Spark platforms respectively and achieved higher execution efficiency on the Spark platform [15].

In this paper, we mainly focus on the scalability and sparsity of the collaborative filtering algorithm. Based on the ALS collaborative filtering recommendation algorithm, the items are usually limited and the number of users is usually much larger than the number of items in most systems, which makes each item have more users to score. The item similarity calculation can reduce the loss of the attribute information of the stealth factor and mitigate the impact of data sparsity on the similarity calculation. At the same time, the user's cold start strategy is added during the model training, and the final similarity of the item is obtained to improve the accuracy of the recommendation. Therefore, a collaborative filtering recommendation algorithm with improved computational integration of item similarity is proposed. Using the Movie Lens dataset published by the GroupLens Lab, the algorithm is implemented on the Spark distributed platform and single node. Building the Spark environment on the Hadoop platform to implement the recommendation algorithm can make full use of Spark's memory computing advantages based on Resilient Distribution Dataset (RDD). Efficient data sharing in the parallel computing phase makes the algorithm more scalable and efficient.

2. Author Artwork

2.1. Spark Introduction

Spark is a general-purpose processing framework in big data. It was developed by the Algorithms Machines and People Lab. It supports multiple paradigms such as memory computing, stream processing, multiple iterative processing, and graph computing. Spark seeks to demonstrate an open source platform for big data applications through large-scale integration between algorithms, machines, and people [16]. Spark was developed on the basis of Hadoop, but it is very different from Hadoop [17]. The Spark distributed computing model puts the data in the calculation process in memory. Subsequent running jobs are directly calculated from memory using these intermediate results to avoid repeated reading and writing to the disk. Because memory reading and writing are much faster than disk, Spark has faster computing speeds. This will make Spark more suitable for iterative calculations.

2.2. Spark Distributed Computing Framework

The Resilient Distribution Dataset (RDD) was introduced in Spark. RDD is a read-only distributed shared memory model with MapReduce data flow model fault tolerance, which enables efficient data sharing in the parallel computing phase [18]. RDD provides a coarse-grained transformation-based interface that can apply the same operations to multiple datasets. This allows them to record information about the dataset without storing the real data for efficient fault tolerance. Therefore, when an RDD partition is lost, the RDD can use the recorded information to recalculate, so that the lost data can be quickly recovered at a small cost.

The overall structure of Spark is shown in Figure 1. Driver is a data processing logic composed of SparkContext, ClusterManager, WorkNode, and Executor [19]. The ClusterManager is responsible for resource management and scheduling of the cluster. WorkNode is the node in the cluster that can perform computing tasks. The Executor is responsible for executing functions, and the Task is the computing unit [20]. When the cluster is running, SparkContext interacts with the ClusterManager, requests computing resources, and finally performs calculations on the Executor.

3. Collaborative Filtering Recommendation Algorithm based on ALS

The collaborative filtering recommendation algorithm based on ALS takes into account two aspects of both users and items, which belong to the collaborative filtering recommendation algorithm of the implicit semantic model. The user item rating is abstracted into a triple in the collaborative filtering algorithm: <UserID, ItemID, Rating>. Among them, UserID is the user number, ItemID is the item number, and Rating is the user's rating of the item.

Assuming that there is a batch of data containing m users and n items, the user item rating matrix can be defined as $R_{(m\times n)}$, and then the element R_{ui} denotes the rating of the U user for the I item. In practical applications, the numbers of either users or items are very large, so the user item rating matrix $R_{(m\times n)}$ is also very large. In the face of such a huge matrix, the

method of traditional matrix decomposition is difficult to handle. At the same time, in the real world, it is almost impossible for a user to rate all the items, which will make the user item rating matrix $R_{(m\times n)}$ usually a sparse matrix.

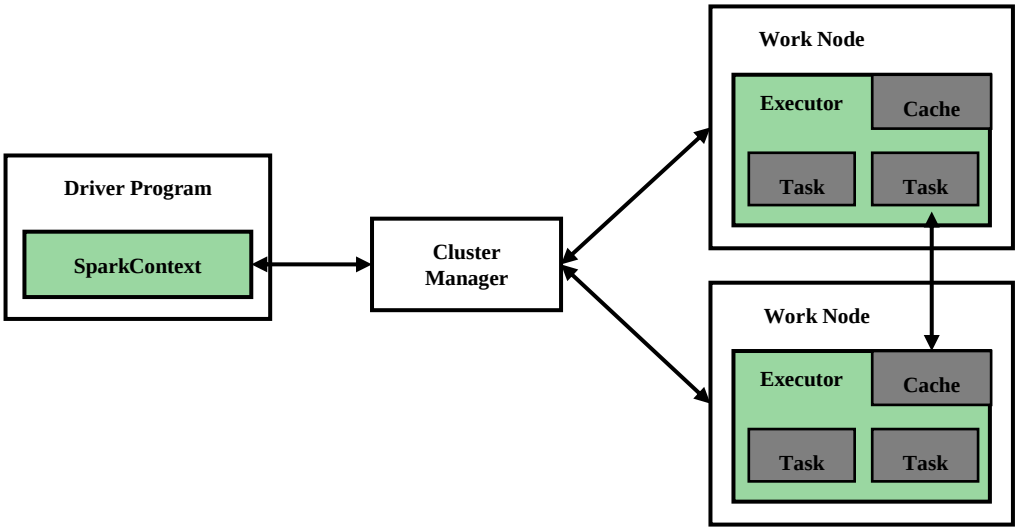

Figure 1. Spark system architecture diagram

The core principle of the matrix decomposition algorithm based on ALS is to solve a series of alternating least squares problems by an iterative method. At each iteration, the user factor matrix or the item factor matrix can be fixed to solve another matrix until the model converges or reaches a preset number of iterations [21]. This is an optimization method for solving matrix decomposition problems, which is easy to implement in the Spark platform. The basic principles are described as follows:

For the $m\times n$ user-item rating matrix $R=(R_{ij})^{m\times n}$, it is desirable that the user-item rating matrix $R_{(m\times n)}$ can be approximated by a low rank matrix X, as in Equation (1).

$$R \approx X = U^T \tag{1}$$

Where $U\in C^{m\times d}$, $M\in C^{n\times d}$, d denotes the number of impacting user rating features value of user and item. In general, $d \ll r$, $r \approx min(m, n)$, where r denotes the rank of the user scoring matrix $R_{(m\times n)}$. Since the user scoring matrix $R_{(m\times n)}$ is approximately equal to the matrix X, the loss function can be expressed as Equation (2).

$$L(U,M) = \sum_{i,j\in T} \left(R_{ij} - X_{ij}\right)^2 = \sum_{i,j\in T} \left(R_{ij} - U_I M_j^{\ T}\right)^2 \tag{2}$$

In order to prevent over-fitting, add a regularization term to Equation (2) and rewrite it as Equation (3).

$$L\left(U,M\right) = \sum_{i,j\in T} \left(R_{ij} - U_i M_j^T\right)^2 + \lambda\left(\left\|U_i\right\|^2 + \left\|M_i\right\|^2\right) \tag{3}$$

In each iteration, M is fixed, the characteristics of each user are updated one by one, and the solution Equation (4) of U_i is obtained by calculating the partial derivative of U_i.

$$U_i = R_i M_{ui}^T \left(M_{ui}^T M + \lambda n_{ui} I\right)^{-1}, \quad i \in \left[1, m\right] \tag{4}$$

In Equation (4), R_i denotes the rating vector of user I for the item and M_{ui} denotes the feature matrix composed of the eigenvectors of the item evaluated by user i. n_{ui} denotes the number of items evaluated by user i. In the same way, obtain Equation (5) for solving M.

$$M_j = R_j^T U_{mj} \left(U_{mj}^T U_{mj} + \lambda n_{mj} I \right)^{-1}, \quad j \in [1, n] \tag{5}$$

In Equation (5), R_j denotes the user's rating vector for item j. U_{mj} denotes a feature matrix composed of user's feature vectors that are overrated for item j. n_{mj} denotes the number of overrated users of item j, where I denotes a $d \times d$ unit matrix in Equations (4) and (5).

Equations (4) and (5) are alternately called to update the calculations U and M until the calculated result converges or the number of iterations reaches the preset number of times. Finally, the approximate matrix X is obtained and recommended by matrix X.

4. Parallelization Improvement of the Recommendation Algorithm based on ALS

4.1. Detailed Improvement Method of Algorithm

The core principle of the recommendation algorithm based on the ALS model is to multiply matrix U by matrix M^T. Matrix U and matrix M are decomposed from user item score matrix R, matrix U denotes the user factor matrix, and matrix M denotes the item factor matrix. The matrices U and M are obtained by continuously optimizing the loss function to replace the original user item scoring matrix R. This can solve the sparseness problem of the user item rating matrix R to a certain extent. The matrices U and M are generated by a method of random initialization. Therefore, it is easy to lose some information of the user or the item in the process of solving the matrices U and M, so that the algorithm ignores the similarity and cold start problem.

On the problem of similarity, the recommendation system has user similarity and item similarity. In the vast majority of systems, the number of users is much larger than the number of items, which is usually limited. This makes each item possess more users to rate it. Based on this article, the calculation of item similarity is added. Through research and comparison, the Pearson correlation coefficient from the common similarity calculation methods is selected in this paper, such as cosine similarity, modified cosine similarity, and Pearson correlation coefficient to calculate the similarity between items. The Pearson correlation coefficient is defined as shown in Equation (6).

$$sim(i, j) = \frac{\sum_{c \in I_{ij}} \left(R_{i,c} - \overline{R}_i \right) \left(R_{jc} - \overline{R}_j \right)}{\sqrt{\sum_{c \in I_{ij}} \left(R_{i,c} - \overline{R}_i \right)^2} \sqrt{\sum_{c \in I_{ij}} \left(R_{jc} - \overline{R}_j \right)^2}} \tag{6}$$

The loss function after integrating into the similarity calculation is shown in Equation (7).

$$f(U, M) = L(U, M) + sim(i, j) = \sum_{i, j \in I_{ij}} \left(R_{ij} - U_i M_j^T \right)^2 + \lambda (\| U_i \|^2 + \| M_i \|^2) + \frac{\sum_{c \in I_{ij}} \left(R_{i,c} - \overline{R}_i \right) \left(R_{jc} - \overline{R}_j \right)}{\sqrt{\sum_{c \in I_{ij}} \left(R_{i,c} - \overline{R}_i \right)^2} \sqrt{\sum_{c \in I_{ij}} \left(R_{jc} - \overline{R}_j \right)^2}} \tag{7}$$

In Equation (7), I_{ij} denotes a set of items that are jointly scored by user i and user j. $R_{i,c}$ denotes the rating of user i for item c. $R_{j,c}$ denotes the score of user j for item c.

4.2. Spark Parallelization Description of the Improved Algorithm

When implementing the algorithm, Spark reads the experimental dataset from Hadoop's Distributed File System (HDFS) and converts it into a compression matrix [22]. The improved algorithm flow is shown in Figure 2, where the dashed line is the improved part in this article.

Spark creates a resilient distribution dataset based on the transformed matrix data to perform iterative calculation to solve the matrices U and M. The improved algorithm is as follows:

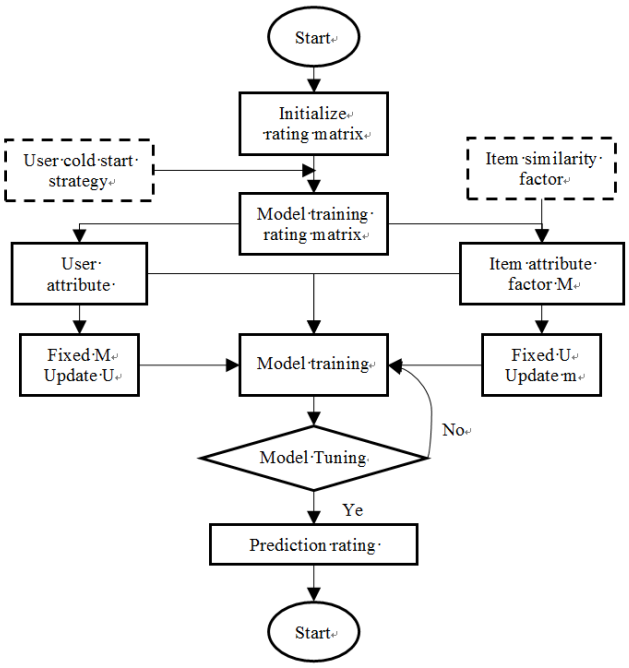

Figure 2. Improved recommendation algorithm flow based on ALS

Algorithm 1 CsSCF Recommendation Algorithm

Input: Ratings and myRatings
#Ratings is the experimental dataset - user item scoring matrix
#myRatings is the current user's rating of the 10 most popular random items in all Ratings
Output: Item factor matrix *M* and user factor matrix *U*

1. Read the experimental dataset to generate the user item rating matrix Ratings
   Sc = new SparkContext()
   Ratings = sc.textFile(dataDir)#Rating is the sequence <User ID, Item ID, rating>
2. User cold start strategy

#Extract the most popular 100 items from all the items, and randomly select 10 items from them to recommend to the current user

   topMovies = ratings.movies.top(100)
   selectMovies = random.topMoviesval

#Get the current user's rating on the currently recommended item to generate myRatings

   myRatings = elicitateRating(selectMovies)
   myRatingsRDD = sc.parallelize(myRatings, 1)

3. Split dataset into training set (80%) and test set (20%)
   valnumPartitions = 10
   R = ratings.filter(x=>x._1<8).map(_._2).union(myRatingsRDD).repartition(numPartitions).persist()
   testR = ratings.filter(x=>x._1>=8).map(_._2).persist()
4. Generate ms matrix and us matrix
   M = matrix(rand(m,F))
   U = matrix(rand(u,F))
   Rb = sc.broadcast(R)
   Mb = sc.broadcast(M)
   Ub = sc.broadcast(U)
5. Similarity calculation
   Item_sims = Rb.map(lambdax:calcSim(x[0],x[1])).map(lambda:keyOnFistItem(x[0],x[1]))
   groupByKay().map(lambdax:nearestNeighbors(x[0],x[1],20))
6. Solve according to the preset number of iterations

For range(numIters):
fixed U and parallel update M

   M = sc.parallelize(range(m),partisons).map(lambda x:update(x,Mb.value[x,:],Ub,value,Rb.value)).collect()
   M = matrix(np.array(M[:,:,0]))
   Mb = sc.broadcast(M)

#fixed M and parallel update U

   U = sc.parallelize(range(u,partisons).map(lambda x:update(x,Ub.value[x,:],Mb.value,Rb.value.T)).collect()
   U = matrix(np,array(U)[:,:,0])
   Ub = sc.broadcast(U)
   Error = rmse(R,M,U)

7. Sc.stop

5. Experiments

5.1. Spark Lab Environment Description

The Spark distributed cluster experimental platform was built using three PCs. The three machines are all installed with the Centos7 system, which includes a master node and two slave nodes. The nodes are connected through the LAN. The node IP address details are shown in Table 1.

Table 1. Spark cluster details

Host name	IP address	Function
Master	10.10.114.41	NameNode,JobTracker
Slave1	10.10.114.42	DataNode,TaskTracker
Slave2	10.10.114.43	DataNode,TaskTracker

The Hadoop version is hadoop-2.6.5. The jdk-8u151-linux-x64.tar.gz and scala-2.10.5.tgz are installed in the Hadoop distributed cluster. After configuring Secure Shell (SSH) to login without a password and installing the Hadoop cluster, install Spark on the basis of which the Spark version is spark-2.2.2-bin-hadoop 2.6. The experimental development environment is ideaIU-2017.2.7.tar.gz. After all the above software is installed on three machines and the corresponding configuration is completed, the Spark distributed cluster can be started. The process of Master node after cluster startup is shown in Figure 3.

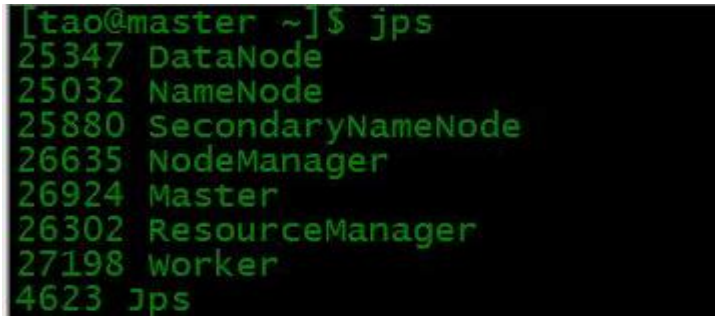

Figure 2. Process of the master node after the Spark cluster is started

5.2. Experiment Dataset

This experiment used the MovieLens dataset published by the GroupLens Laboratory of the Department of Computer Science and Engineering at the University of Minnesota [23]. The data format is shown in Table 2. The user's rating on the movie is from one to five. One expresses dislike, five indicates the user likes it very much, and zero does not take part in it.

Table 2. MovieLens data format

User ID	Movie ID	Rating	Timestamp
User number	Movienumber	Rating value	Time stamp

5.3. Experiment Evaluation Indicators

This experiment uses Root Mean Squared Error (*RMSE*) as a measure of the accuracy of predictive rating [24]. For each user u and item i in the test set, let T denote the rating set of user u for item i, r_{ui} is the actual rating user u for item i and the predicted score given by the recommendation algorithm, and then *RMSE* is defined as Equation (8).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{T}\left(r_{ui} - \hat{r}_{ui}\right)^2}{T}} \tag{8}$$

5.4. Analysis of Experiment Results

For the convenience of description in this paper, the improved algorithm is named CsSCF for the time being. The relevant experimental results are as follows. In the experiment, the following three tests are designed to evaluate the improved algorithm.

- Test the recommended accuracy of the improved algorithm under different data scales.
- Compare the recommended accuracy of the improved algorithm with the traditional algorithm.
- Test the timeliness of recommended systems in single-node and distributed environments.

Experiment one: to test the *RMSE* values in different data sizes. The improved algorithm was verified repeatedly on the Spark distributed computing platform with 100,000, 1 million, 10 million, and 100 million pieces of data. The *RMSE* stability value of the improved algorithm was obtained by repeatedly testing 100,000, 1 million, 10 million, and 100 million pieces of data on Spark, as shown in Figure 4. The experimental results show that the *RMSE* of the rating gradually decreases with the expansion of the data size. This means that the more rating data it has, the more accurate results the algorithm will likely achieve. Thus, it is necessary to prevent overfitting as well.

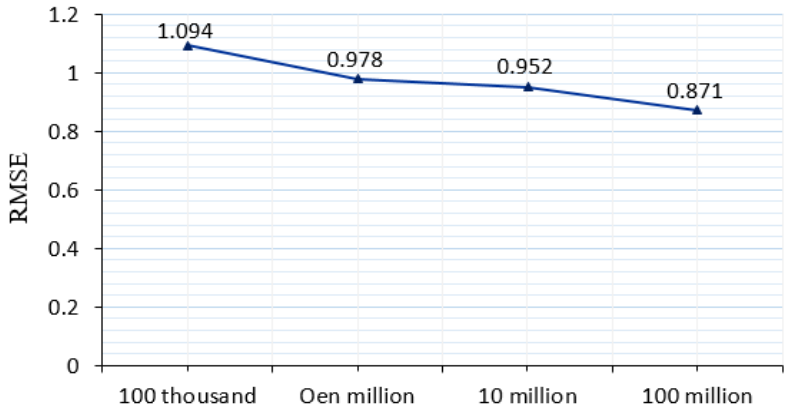

Figure 3. *RMSE* values trend at different data sizes

Experiment two: to compare the recommended accuracy of different algorithms. In the Spark environment, the improved recommendation algorithm of this paper is compared with the traditional ALS, UerCF, and ItemCF recommendation algorithms. The values of the parameters set by each algorithm are the optimal values determined after multiple experiments. Under the same data scale, the *RMSE* values of the rating prediction accuracy indicators of different algorithms are calculated separately. The experimental comparison results are shown in Figure 5.

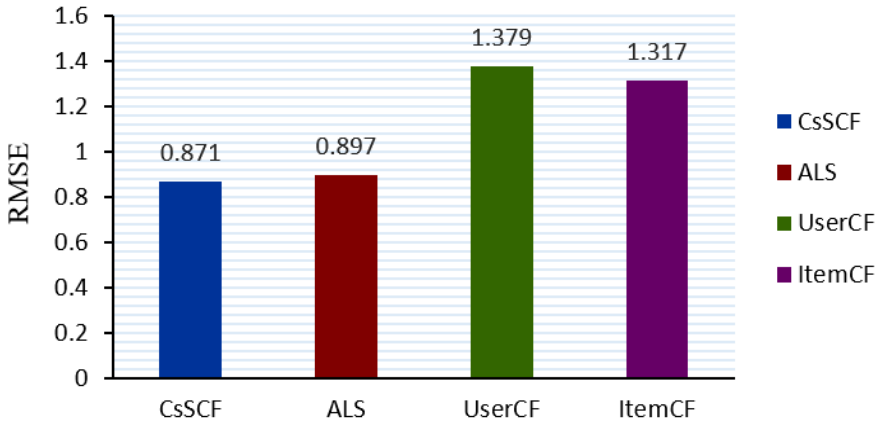

Figure 4. *RMSE* experimental results of different algorithms

It can be seen from Figure 5 that the *RMSE* of the improved algorithm is smaller than that of the traditional UserCF and ItemCF, and it is also smaller than that of the recommendation algorithm based on ALS, which illustrates that the improved algorithm in this paper has a certain improvement in prediction accuracy compared with the above classical algorithms.

Experiment three: to test the timeliness comparison between stand-alone and distributed environments. The timeliness verification of the improved algorithm was tested on a stand-alone and Spark platform. The results are shown in Figure 6. By comparing the calculation times of different modes under the same dataset, it can be seen from the figure that the calculation time of the fully distributed mode based on Spark is the least in the case of the same dataset. It can be found that

as the data size grows, the time-consuming growth of the Spark framework tends to be smoother, and its computing efficiency advantage is more obvious. This reflects the real-time advantage of Spark based on memory computing. It also can be seen from the figure that when the data size is 100,000, the stand-alone mode takes less time than the distributed mode. This shows that Spark needs to start the entire distributed computing system when starting the calculation, which will cost a certain time.

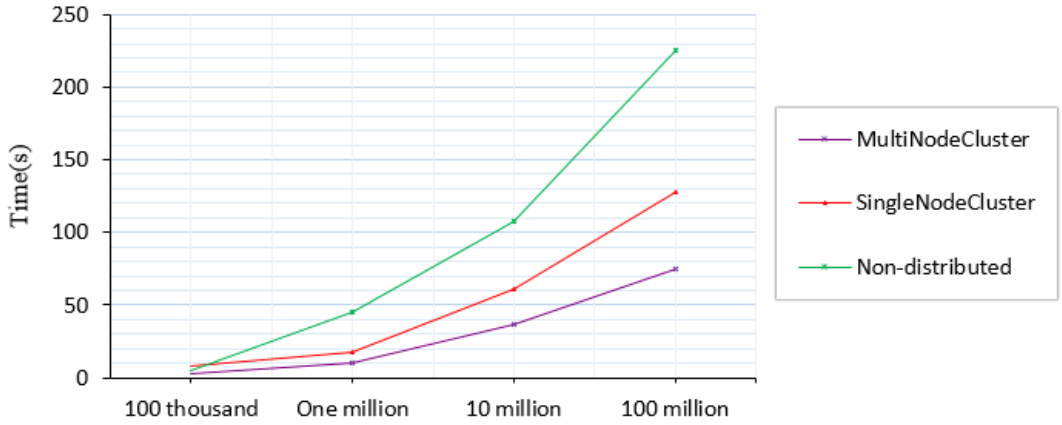

Figure 5. Comparison of computational efficiency between single and distributed environments

In summary, compared with several classical algorithms, a certain improvement in recommendation accuracy and computational efficiency has been presented in the improved algorithm.

6. Conclusions

On the basis of the collaborative filtering algorithm based on ALS, an improved algorithm that combines item similarity calculation and user cold start strategy is proposed. The improved algorithm is implemented in a single machine and Spark distributed platform. The experimental results show that the improved algorithm enhances the recommendation accuracy and computational efficiency. Nevertheless, there are also shortcomings. The algorithm studied in this paper only deals with the situation of display feedback, whereas in real life, explicit feedback and implicit feedback coexist. Integrating these two resources to improve the quality of recommendation needs further study. Therefore, the combination of implicit feedback and the use of larger datasets to further improve the quality of recommendations will be the next major direction for research.

Acknowledgments

The research is supported by the National Nature Science Fund Project (No. 61562093, 61661051), Key Project of Applied Basic Research Program of Yunnan Province (No. 2016FA024), Program for Innovative Research Team (in Science and Technology) in University of Yunnan Province, Application Infrastructure Projects of Science and Technology Plan in Yunnan Province (No. 2016FD022), and Starting Foundation for Doctoral Research of Yunnan Normal University (No. 2017ZB013).

References

1. F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems Handbook," 2nd Edition, Springer-Verlag New York Inc., New York, USA, November 2015
2. D. S. Li, C. Chen, Q. Lv, and H. S. Gu, "An Adaptive Learning Rate Method for Matrix Approximation-based Collaborative Filtering," in *Proceedings of 2018 World Wide Web Conferences*, pp. 741-751, Lyon, France, July 2018
3. N. E. I. Karabadji, S. Beldjoudi, H. Serid, S. Aridhi, and W. Dhifli, "Improving Memory-based User Collaborative Filtering with Evolutionary Multi-Objective Optimization," *Expert Systems with Applications*, Vol. 98, pp. 153-165, May 2018
4. B. Shams and S. Haratizadeh, "Item-based Collaborative Ranking," *Knowledge-based Systems*, Vol. 152, pp. 172-185, July 2018
5. T. M. Huynh, H. H. Huynh, V. T. Tran, and H. X. Huynh, "Collaborative Filtering Recommender System base on The Interaction Multi-Criteria Decision with Ordered Weighted Averaging Operator," in *Proceedings of the 2nd International Conference on Machine Learning and Soft Computing*, pp. 45-49, New York, USA, February 2018
6. E. Karydi and K. G. Margaritis, "Parallel and Distributed Collaborative Filtering: A Survey," *ACM Computing Surveys Surveys Homepage Archive* (*CSUR*), Vol. 49, No. 2, pp. 1-46, New York, USA, November 2016

7. Y. Wang and L. He, "Research and Optimization of Data Sparsity in Collaborative Filtering Algorithms," *Recent Developments in Intelligent Computing, Communication and Devices*, Vol. 752, pp. 87-92, Singapore, August 2018
8. J. Xu, Y. S. Zhong, W. Q. Zhu, and F. F. Sun, "Trust-based Context-Aware Mobile Social Network Service Recommendation," *Wuhan University Journal of Natural Sciences*, Vol. 22, No. 2, pp. 149-156, Wuhan, China, April 2017
9. Y. C. Jing, W. Jiang, G. Y. Su, Z. S. Zhou, and Y. F. Wang, "A Learning Automata-based Singular Value Decomposition and its Application in Recommendation System," Springer International Publishing, pp. 26-32, Taiyuan, China, August 2014
10. Z. J. Wang, N. N. Yu, and J. X. Wang, "User Attributes Clustering-based Collaborative Filtering Recommendation Algorithm and its Parallelization on Spark," in *Proceedings of Asian Simulation Conference*, pp. 442-451, Beijing, China, September 2016
11. L. Zhang, X. Song, and Y. J. Wu, "Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems," in *Proceedings of 16th Asia Simulation Conference and SCS Autumn Simulation Multi-Conference*, Beijing, China, October 2016
12. C. Guan and K. K. F. Yuen, "Towards a Hybrid Approach of Primitive Cognitive Network Process and Agglomerative Hierarchical Clustering for Music Recommendation," in *Proceedings of 11th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness* (*QSHINE*), pp. 206-209, Taipei, Taiwan, November 2015
13. Z. D. Zhao and M. S. Shang, "User-based Collaborative Filtering Recommendation Algorithms on Hadoop," in *Proceedings of International Conference on Knowledge Discovery & Data Mining*, pp. 478-481, Phuket, Thailand, January 2010
14. L. Fan, H. Li, and C. F. Li, "The Improvement and Implementation of Distributed Item-based Collaborative Filtering Algorithm on Hadoop," in *Proceedings of 34th Chinese Control Conference* (*CCC*), pp. 9078-9083, Hangzhou, China, July 2015
15. K. H. Lin, J. J. Wang, and M. H. Wang, "A Hybrid Recommendation Algorithm based on Hadoop," in *Proceedings of 9th International Conference on Computer Science and Education*, pp. 540-543, Vancouver, BC, Canada, August 2014
16. B. Kupisz and O. Unold, "Collaborative Filtering Recommendation Algorithm based on Hadoop and Spark," in *Proceedings of 2015 IEEE International Conference on Industrial Technology* (*ICIT*), pp. 1510-1514, Seville, Spain, June 2015
17. S. Salloum, R. Dautov, X. J. Chen, P. X. G. Peng, and J. Z. X. Huang, "Big Data Analytics on Apache Spark," *International Journal of Data Science and Analytics*, Vol. 1, No. 3-4, pp. 145-164, November 2016
18. Y. Samadi, M. Zbakh, and C. Tadonki, "Performance Comparison Between Hadoop and Spark Frameworks using HiBench Benchmarks," *Concurrency and Computation Practice & Experience*, pp. 1-13, Wiley, USA, November 2017
19. R. K. Mishra, "Spark Architecture and The Resilient Distributed Dataset," *PySpark Recipes*, pp. 85-114, Apress, Berkeley, CA, December 2017
20. D. G. Garc á, S. G. Ram ŕez, and S. García, "A Comparison on Scalability for Batch Big Data Processing on Apache Spark and Apache Flink," *Big Data Anal*, Vol. 2, No. 1, pp. 1, March 2017
21. I. S. Wahyudi, A. Affandi, and M. Hariadi, "Recommender Engine using Cosine Similarity based on Alternating Least Square-Weight Regularization," in *Proceedings of 15th International Conference on Quality in Research* (*QiR*), *International Symposium on Electrical and Computer Engineering*, pp. 256-261, Nusa Dua, Indonesia, July 2017
22. C. Verma and R. Pandey, "Big Data Representation for Grade Analysis Through Hadoop Framework," in *Proceedings of 6th International Conference - Cloud System and Big Data Engineering* (*Confluence*), pp. 312-315, Noida, India, January 2016
23. R. Katarya and O. P. Verma, "Recommender System with Grey Wolf Optimizer and FCM," *Neural Computing & Applications*, Vol. 30, No. 5, pp. 1679-1687, September 2018
24. C. Selvi and E. Sivasankar, "A Novel Singularity based Improved Tanimoto Similarity Measure for Effective Recommendation using Collaborative Filtering," in *Proceedings of 2018 8th International Conference on Cloud Computing, Data Science & Engineering* (*Confluence*), pp. 256-262, Noida, India, January 2018

Jinhong Tao is a master's student in the School of Information Science and Technology at Yunnan Normal University. His research interests include machine learning and data mining.

Jianhou Gan received his Ph.D. in metallurgical physical chemistry from Kunming University of Science and Technology in 2016. In 1998, he was a faculty member at Yunnan Normal University. Currently, he is a professor at Yunnan Normal University. His research interests cover education informatization for nationalities, semantic Web, database, and intelligent information processing.

Bin Wen received his Ph.D. in computer application technology from China University of Mining & Technology in 2013. In 2005, he was a faculty member at Yunnan Normal University. Currently, he is an associate professor at Yunnan Normal University. His research interests cover intelligent information processing and emergency management.