Improving Matrix Factorization Recommendations for Problems in Big Data

Xiaohan Tu
College of Computer Science and Electronic Engineering
Hunan University
Changsha 410082, China
tutu16103@hnu.edu.cn

Siping Liu
College of Computer Science and Electronic Engineering
Hunan University
Changsha 410082, China
liusiping@hnu.edu.cn

Renfa Li
College of Computer Science and Electronic Engineering
Hunan University
Changsha 410082, China
lirenfa@hnu.edu.cn

***Abstract*—In big data environment, recommender systems are facing many problems, such as poor extendibility, data sparseness and low efficiency. In this paper, a new collaborative filtering parallel algorithm named NALS-WR is designed in the Linux clusters by using Spark to solve these problems, especially aiming at the bottleneck of processing speed and resource allocation of traditional matrix factorization algorithm in massive data information. Our experiments were on the real movielens datasets. Compared with other such recommendation system based on ALS-WR or sigular value decomposition (SVD), the accuracy rate was improved. The running efficiency was much higher than the ALS-WR in Hadoop, and it is also faster than SVD. The bigger the scale of data, the more efficient it is. This algorithm can improve the execution efficiency of collaborative filtering recommendation algorithm at large data scale, which solves the problem of over-high time of matrix factorization recommendation algorithm. Our experiments indicated that our NALS-WR algorithm was better, whether in extendibility, sparseness resistance or efficiency in the implementation.**

Keywords-component; Recommender systems; Data sparsity problem; Collaborative filtering; Alternating-least-squares with Weighted-λ-regularization (ALS-WR).

I. INTRODUCTION

At present, the most widely used algorithm in recommendation system is based on collaborative filtering recommendation algorithm [1]. Traditional collaborative filtering techniques are often categorized into two categories [2, 4]: model-based methods (e.g. latent factor model) [2, 5] and memory-based methods (e.g. nearest neighborhood) [2, 5, 6]. In general, model-based methods are known to generate more accurate recommendation results, such as the singular value decomposition (SVD) [7] and alternating-least-squares (ALS) [8]. But with the number of users and items in ecommerce services growing quickly, the sparseness of user-to-item rating data deteriorates the rating prediction accuracy of traditional collaborative filtering techniques [2, 3, 4]. ALS is not scalable to large-scale data set in the paper [23]. Then ALS is improved (ALS-WR), which is very simple and has good expansibility for large datasets [8]. While the nearest neighbor method is used to predict score, which can solve the problem of synonyms to some extent. But the result of feature matrix cannot be directly used for grading, because the scores of most rating matrix are filled before the decomposition [9]. A distributed matrix factorization algorithm in the papers is proposed [10, 11], which can manage the parallel computation of matrix factorization. However, the MapReduce framework in iterations of computing nodes used in these papers can generate many operations in reading files, which affects the efficiency of the algorithm. Compared with the previous results, our contributions are summarized as follows:

Firstly, we proposed a method of normalized processing in ALS-WR to solve data sparseness. We imposed bounds on ALS-WR algorithm. The improved ALS-WR algorithm (NALS-WR) and ALS-WR, SVD algorithm were compared. The experimental results showed that the NALS-WR algorithm outperformed the ALS-WR and SVD algorithms under the problems of various data sparseness and the recommended results are a little more accurate.

Secondly, we combined NALS-WR with the Spark parallel computing framework, which specializes in memory computation and iterative computation. These solved the problem of low computational efficiency of general matrix factorization recommendation algorithm in big data environment.

The remainder of the paper was organized as follows. Section 2 briefly introduced Spark platform and ALS-WR and SVD algorithm. Section 3 presented the detailed process of NALS-WR algorithm, and explained the parallel architecture of our NALS-WR. Section 4 experimentally evaluated Spark-based NALS-WR and evaluate the results. Section 5 summarized our contributions and gave future work.

II. PRELIMINARY

In this section, we will briefly review our experiment platform, which is Apache Spark, then the matrix factorization (MF) (the most popular collaborative filtering technique), the ALS-WR and SVD algorithm.

A. Apache Spark

Apache Spark is an open source cluster computing system that aims to make data analytics fast - both fast to run and fast to write. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time [12].

Spark has proposed a Resilient Distributed Dataset (RDD). The essence of RDD is the definition of parallel data containers. Different data set format corresponds to different types of RDD. If a RDD fragment is lost, Spark can reconstruct it from the log information, So RDD is resilient; Spark can operate the local dataset in a way that manipulates the local set, So RDD is also distributed. In addition. RDD can be cached in memory, it can be read directly from memory in the subsequent calculation process while eliminating the need for a large number of disk access costs, it is befitting for the iterative calculation in the matrix factorization algorithm. Each Spark application has its own executor process, the life cycle of executor and the entire life cycle of application are the same, and it maintains multiple threads inside in order to perform parallel the task assigned to it. This mode of operation is contributed to the separation of resources and scheduling isolation between different applications. The basic workflow of our experiment platform is shown in Figure 1.

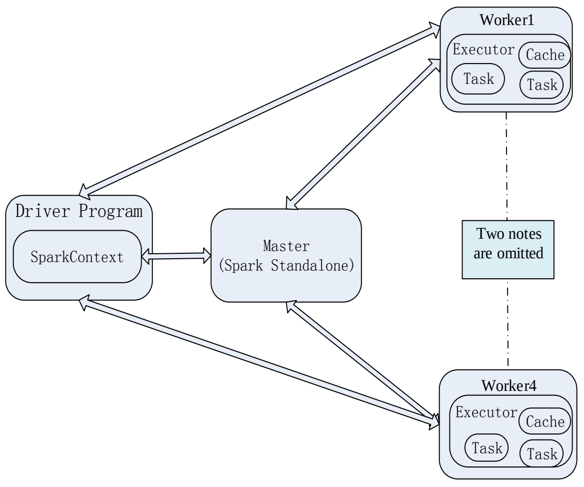

Figure 1. The workflow of our experiment platform

B. Basic definitions

In this paper, the matrix is indicated in italic capital letters (e.g. ***P***) and the scalar is denoted by lowercase letters (e.g. i, j). Given a matrix ***P***, ***P***$_{ij}$ represents a of its elements, ***P***$_{i.}$ represents the ith row of the matrix ***P***, ***P***$_{.j}$ represents the jth column of the matrix ***P***, **,** and ***P***^{T} represents the transpose of the matrix ***P***. ***P***$^{-1}$ represents the inverse of the matrix ***P***.The matrix ***P*** given in this paper represents a rating matrix with m users and n items, and the matrices ***U*** and ***V*** respectively represent the characteristic matrix of the users and the characteristic matrix of the recommendation objects.

C. Matrix Factorization

The goal of matrix factorization is to find latent models of users and items on a shared latent space in which the strengths of user-item relationships (i.e., rating by a user on an item) are computed by inner products [2, 5]. In matrix factorization, latent models of user i and item j are denoted as k-dimensional models, and the rating of user i on item j is approximated by the inner-product of corresponding latent models of item j and user i. The widespread way of training latent models is to reduce a loss function L. It consists of sum-of-squared-error terms between the predicted ratings and the actual ratings. In order to avoid the over-fitting problem, they also add some regularization terms.

D. Sigular Value Decomposition

One of the most favorite collaborative filtering recommendation algorithm based on matrix factorization is the singular value decomposition. The user rating matrix is decomposed into a user feature vector matrix and a project feature vector matrix. In order to make recommendations from these characteristics we extract some essential features with using the singular values of the initial rating matrix. SVD is a common method of matrix dimension simplification, which decomposes a matrix ***P*** of m rows and n columns into three matrices [15]. The formula discussed above is as follows:

$$P = U * C * V^{T} . \tag{1}$$

Where ***U*** is an orthogonal matrix of m * m, ***C*** is an orthogonal matrix of m * n, whose diagonal elements are descending in descending order, which is diagonal matrix (c_1, c_2, ..., c_n), known as the singular value. Its non-diagonal elements are all 0. ***V*** is an orthogonal matrix of n * n. Through the decomposition of this matrix, a simplified matrix can be found approximately, which is the diagonal matrix ***C***, and retain its K largest singular values to form a k-dimensional space. Thus, a new diagonal matrix ***C***$_k$ is obtained, and the dimensions of matrices ***U***, ***C***, and ***V*** respectively become *m*k*, *k*k*, *k*n*. Then the approximate matrix we get is:

$$P_k = U_k * C_k * V_k^{T} ,\ P_k \approx P . \tag{2}$$

The singular value decomposition can produce one of the matrices with all ranks equal to k, which is most approximate to the matrix ***P***.

E. ALS-WR

Another collaborative filtering recommendation algorithm based on matrix factorization is alternating-least-squares with weighted-λ-regularization. ALS-WR is the abbreviation of least squares alternating. Different from that traditional recommendation algorithm based on matrix factorization which uses the SVD method to decompose a matrix ***P***. Here, it is desirable to find a low rank matrix ***X*** to approximate the matrix ***P***, which is as follows:

$$X = UV^{T} ,\ U \in C^{m*d} ,\ V \in C^{n*d} . \tag{3}$$

Where d denotes the number of features, and generally d << r. r denotes the rank of matrix ***P***, and r is less than or equal to the minimum value of m or n. In order to find the minimum-rank matrix ***X*** and ***Y*** as close as possible to ***P***, we need to minimize the square error loss function. The loss function needs to add a regularization term to avoid the over-fitting problem [8].

III. NALS-WR

Since SVD is decomposed into three matrices, that shares many variables and can only be applied to a small-scale or large-capacity shared memory solution, so it is difficult to parallelize. On the contrary, ALS-WR can be extended to compute more on large-scale clusters, and used to solve distributed parallel computing with a burst of speed. We use $\boldsymbol{P_{i.}}$ to represent the vector composed by the movie score which has been evaluated by the user i, and $\boldsymbol{V_{ui}}$ represents the feature matrix composed by the feature vector of the movie that the user i has evaluated. Supposing that n_{ui} represents the number of movies that user i has reviewed. $\boldsymbol{P_{.j}}$ denotes the vector composed by ratings of the users who reviewed the movie j, and $\boldsymbol{U_{mj}}$ denotes the characteristic matrix composed by the eigenvectors of the user who has evaluated the movie j. Supposing that n_{mj} denote the number of users who have commented on the movie j. Since we are attempting to find two low-dimensional matrices to approximate the matrix P (m * n), the common loss function is:

$$L(U,V)=\sum_{ij}(P_{ij}-U_{i.}V_{j.}^{\mathrm{T}})^2 . \tag{4}$$

In order to prevent over-fitting, we add the regularization term to (4). Then (4) can be rewritten as below:

$$L(U,V)=\sum_{ij}(P_{ij}-U_{i.}V_{j.}^{\mathrm{T}})^2+\lambda(\left\|U_{i.}\right\|_F^2+\left\|V_{j.}\right\|_F^2) \tag{5}$$

Then we fix $V_{j.}$, derive $U_{i.}$, and make the derivative be equal to 0, this can be expressed as follows:

$$\frac{\partial L(U,V)}{\partial U_{i.}}=0 . \tag{6}$$

We obtain the following formula for solving $U_{i.}$:

$$U_{i.}=P_{i.}V_{ui}(V_{ui}^{\mathrm{T}}V_{ui}+\lambda n_{ui}I)^{-1}, \quad i\in[1,m] . \tag{7}$$

Similarly, we fixed $U_{i.}$, and obtain (8) for solving $V_{j.}$:

$$V_{j.}=P_{.j}^{T}U_{mj}(U_{mj}^{\mathrm{T}}U_{mj}+\lambda n_{mj}I)^{-1}, \ j\in[1,n] . \tag{8}$$

Use iterative method to get U and V, then we normalize them as follows:

$$\begin{gathered}A=\frac{P_{i.}V_{ui}}{V_{ui}^{\mathrm{T}}V_{ui}+\lambda n_{ui}I}-(P_{i.}V_{ui}(V_{ui}^{\mathrm{T}}V_{ui}+\lambda n_{ui}I)^{-1})_{\min},\\ B=(\frac{P_{i.}V_{ui}}{V_{ui}^{\mathrm{T}}V_{ui}+\lambda n_{ui}I})_{\max}-(P_{i.}V_{ui}(V_{ui}^{\mathrm{T}}V_{ui}+\lambda n_{ui}I)^{-1})_{\min},\\ U_{i.}=\frac{A}{B}*s+t,\\ i\in[1,m].\end{gathered} \tag{9}$$

$$\begin{gathered}A=\frac{P_{.j}^{T}U_{mj}}{U_{mj}^{\mathrm{T}}U_{mj}+\lambda n_{mj}I}-(P_{.j}^{T}U_{mj}(U_{mj}^{\mathrm{T}}U_{mj}+\lambda n_{mj}I)^{-1})_{\min},\\ B=(\frac{P_{.j}^{T}U_{mj}}{U_{mj}^{\mathrm{T}}U_{mj}+\lambda n_{mj}I})_{\max}-(P_{.j}^{T}U_{mj}(U_{mj}^{\mathrm{T}}U_{mj}+\lambda n_{mj}I)^{-1})_{\min},\\ V_{j.}=\frac{A}{B}*\mathrm{s}+\mathrm{t},\\ j\in[1,n].\end{gathered} \tag{10}$$

$$\hat{p}_{ij}=\frac{(U_{i.}^{\mathrm{T}}V_{j.}-(U_{i.}^{\mathrm{T}}V_{j.})_{\min})}{(U_{i.}^{\mathrm{T}}V_{j.})_{\max}-(U_{i.}^{\mathrm{T}}V_{j.})_{\min}}*s+t. \tag{11}$$

The parameters s and t are taken the appropriate value, then the results will be mapped to a corresponding interval. In the rating process, if the score is mapped to the interval [0.5, 5.0], we can take s = 4.5 and t = 0.5. By using the normalization operation, the weakness of original ALS-WR is overcome. Experiments will show that the boundary introduced will make the results more accurate. The specific process of our NALS-WR is as follows:

1) First, the matrix $\boldsymbol{V}$ is initialized with a Gaussian random number of mean zero and a deviation of 0.01.
2) Then we obtain $\boldsymbol{U}$ by the objective function (7).
3) Normalize the $\boldsymbol{U}$ obtained above.
4) Similarly, use function (8) to update $\boldsymbol{V}$. Continue to iterate, and normalize each iteration.
5) If RMSE value calculated by the algorithm converges or the number of iterations is enough, we terminate the iteration.
6) According to $X = UV^T$, we get the matrix $\boldsymbol{X}$.
7) Normalize the $\boldsymbol{X}$.

The key of NALS-WR is to update $\boldsymbol{U}$ and $\boldsymbol{V}$ iteratively by using formulas (7) and (8), which are invoked each time that leads to only the row values of matrix $\boldsymbol{U}$ and $\boldsymbol{V}$ are calculated and updated. So the matrix $\boldsymbol{U}$ and $\boldsymbol{V}$ can be divided into a number of some sub-matrixes which are of equal column length. So the NALS-WR can complete the parallel operation. The following experiments show that, our NALS-WR is superior to SVD and ALS-WR in recommending performance.

IV. Experiment and Evaluation

A. *Experimental Environment and Dataset*

We use the machines in our lab, which are all configured to 4G memory, 4-core CPU. The deployment of 5-node Spark cluster and Hadoop cluster are ready. We use Hadoop Distributed File System to store experimental data, and use MovieLens [13, 14] as the experimental datasets. We choose MovieLens-1m, 10m, and 20m to test. Among that MovieLens-20m contains 20000263 ratings, 27,278 movies and 138,493 users. User numbers are form 1 to 138493, film numbers are from 1 to 131262. According to statistics, a single user ultimately participate in the 9254 film ratings.

While only 26,744 films were actually rated, the score density is 0.54%.

B. *The Evaluation Criteria of Experiment*

In our paper, the RMSE is used as the evaluation criterion [2]. The RMSE measures accuracy of prediction. Assuming that one of the rating vectors predicted is p_i, and the corresponding actual user rating set is r_i. Number of rating is N, then the RMSE is expressed as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(p_i - r_i)^2}{N}} \tag{12}$$

C. *Experimental Results and Analysis*

In this section, we compare the performance of NALS-WR with other such CF algorithms. The algorithms involved in the comparison are SVD, NALS-WR and ALS-WR. Figure 2 shows the performance of NALS-WR and ALS-WR when RMSE is used as the performance evaluation index in MovieLens-20m. Each algorithm is trained on the training set, and RMSE is tested on the test set. The number of iterations is 18 times. The vertical axis represents the RMSE. The abscissa represents the numbers of features of the feature matrix. Two algorithms both achieve the minimum value when the number of features is 8. However, the RMSE of NALS-WR is always lower than that of ALS-WR, which shows that the performance of NALS-WR is better than that of ALS-WR. The experimental data are as follows：

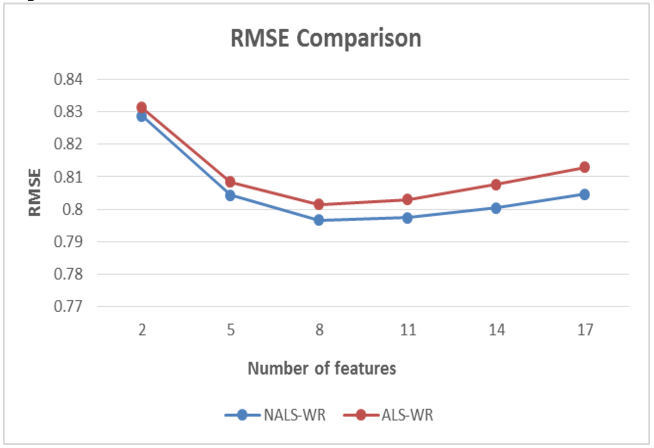

Figure 2: RMSE Comparison of NALS-WR and ALS-WR algorithm

Similar to the above, we compare the performance between NALS-WR and another similar matrix decomposition algorithm (SVD). The RMSE of NALS-WR is the lowest of all, which is 0.796586 when the number of features is 8. The RMSE of NALS-WR is all lower than that of SVD. Experiments show that the performance of NALS-WR is also superior to SVD. The experimental data are as follows in figure 3:

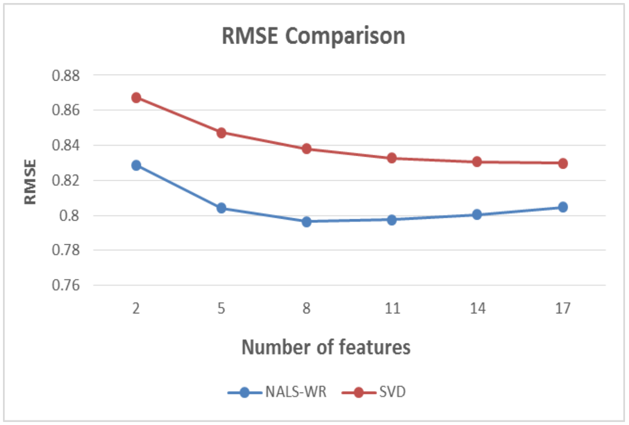

Figure 3: RMSE Comparison of NALS-WR and SVD algorithm

We use MovieLens-1m, 10m and 20m to prove the processing speed of NALS-WR. The evaluation index is runtime (ms). SVD is stand-alone and serial, there is no communication overhead of multiple nodes. In the case of less data, SVD is faster. NALS-WR is based on memory, and parallel distributed computing. There is communication overhead of multiple nodes. So in the case of less data, NALS-WR may be more time-consuming than the SVD. ALS-WR is based on Hadoop which uses Mapreduce to process data, and MapReduce in each execution must read data from the disk. After the completion, the data will be stored to disk. The iterative data on many occasions will consume a lot of time, which above is as shown in TableⅠ. As the amount of data increases, the advantage of NALS-WR appears. The experimental data is shown in Tables Ⅱ and Ⅲ. In a large amount of data, the speed of NALS-WR algorithm will be far more than the single-node SVD and disk-based ALS-WR algorithm. Therefore, NALS-WR can improve the efficiency of collaborative filtering recommendation system at large data scale, and solve the problem of time cost in matrix factorization recommendation algorithm.

Figure 4 shows the efficiency evaluation of NALS-WR, ALS-WR and SVD algorithm in MovieLens-20m. The horizontal axis represents the number of features of the feature matrix. The vertical axis represents the runtime (ms) value. Since the ALS-WR operation under Hadoop uses Mapreduce to process data, MapReduce reads data from disk every time when it is executed. After the calculation, the data is stored on the disk. The constant iteration will consume a lot of input and output. Spark is based on memory. So when the number of iterations is 18, the runtime of Spark-based parallel algorithm NALS-WR is much shorter than that of ALS-WR, also shorter than that of SVD serial operation. The runtime of NALS-WR linearly increase with the augment of the number of features. It shows that execution efficiency can bring the highest speedup ratio.

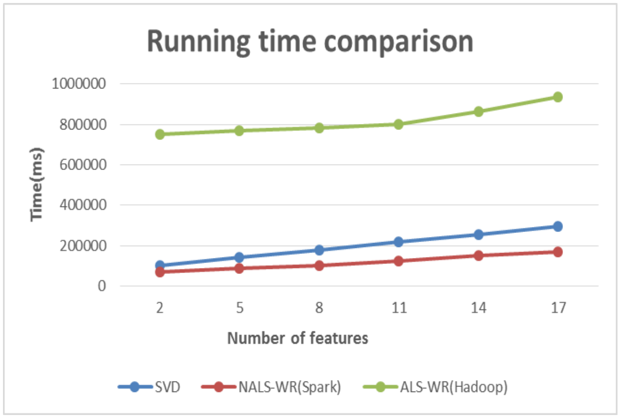

Figure 4: Running time of three algorithms when iterations is confirmed

V. SUMMARY

This paper proposed a new parallel algorithm based on matrix factorization, and analyzed its extendibility, anti-sparseness and efficiency of implementation. Our experiments indicated that our NALS-WR algorithm was better, whether in extendibility, sparseness resistance or efficiency. The bigger the scale of data, the more efficient it is. NALS-WR can improve the execution efficiency of collaborative filtering recommendation algorithm at large data scale, and solves the problem of over-high time in matrix factorization recommendation system. Because we can use Spark to do real-time processing (Hadoop can't do this). We also provide a transition from batch to real-time processing. In future work, we will combine with other algorithms to statistically improve RMSE and real - time.

ACKNOWLEDGMENT

This paper was supported by the National Natural Science Foundation of China (No: 61173036 and 61202289), the Science and Technology Plan of Hunan Province (No. 2015GK3015) and The National High-Tech Research and Development Plan of China under Grant (No.2012AA01A301-01).

REFERENCES

[1] Koren, Yehuda. "Collaborative filtering with temporal dynamics." Communications of the ACM 53.4 (2010): 89-97.

[2] Kim, D., Park, C., Oh, J., Lee, S., & Yu, H. "Convolutional matrix factorization for document context-aware recommendation." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016: 233-240.

[3] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. "Evaluating collaborative filtering recommender systems." ACM Transactions on Information Systems (TOIS) 22.1 (2004): 5-53.

[4] Adomavicius, Gediminas, and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." IEEE transactions on knowledge and data engineering17.6 (2005): 734-749.

[5] Koren, Yehuda. "Factorization meets the neighborhood: a multifaceted collaborative filtering model." Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.

[6] Deshpande, Mukund, and George Karypis. "Item-based top-n recommendation algorithms." ACM Transactions on Information Systems (TOIS) 22.1 (2004): 143-177.

[7] Paterek, Arkadiusz. "Improving regularized singular value decomposition for collaborative filtering." Proceedings of KDD cup and workshop. Vol. 2007. 2007.

[8] Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. "Large-scale parallel collaborative filtering for the netflix prize." International Conference on Algorithmic Applications in Management. Springer Berlin Heidelberg, 2008.

[9] Ko, S. Y., Hoque, I., Cho, B., & Gupta, I. "On Availability of Intermediate Data in Cloud Computations." HotOS. 2009.

[10] Hu, Peng, and Wei Dai. "Enhancing fault tolerance based on Hadoop cluster." International Journal of Database Theory and Application 7.1 (2014): 37-48.

[11] ZHU H. Fault tolerance for MapReduce in the cloud environment [D]. Shanghai: Shanghai Jiao Tong University, 2012

[12] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. "Spark: cluster computing with working sets." HotCloud10 (2010): 10-10.

[13] MovieLens: http://grouplens.org/datasets/movielens/

[14] Harper, F. Maxwell, and Joseph A. Konstan. "The movielens datasets: History and context." ACM Transactions on Interactive Intelligent Systems (TiiS) 5.4 (2016): 19.

[15] Koren, Yehuda. "Factor in the neighbors: Scalable and accurate collaborative filtering." ACM Transactions on Knowledge Discovery from Data (TKDD) 4.1 (2010): 1.

TABLE I. RUNNING TIME OF THREE ALGORITHMS WHEN ITERATIONS IS CONFIRMED IN MOVIELENS-1M

Number of features	2	5	8	11	14	17
Runtime (ms) of NALS-WR (Spark)	18000	19640	20176	20203	21691	22719
Runtime (ms) of SVD	5034	6563	8382	10379	12191	14146
Runtime (ms) of ALS-WR (Hadoop)	474548	482602	485783	486582	487137	491830

TABLE II. RUNNING TIME OF THREE ALGORITHMS WHEN ITERATIONS IS CONFIRMED IN MOVIELENS-10M

Number of features	2	5	8	11	14	17
Runtime (ms) of NALS-WR (Spark)	46161	55467	73185	75082	83838	99143
Runtime (ms) of SVD	50416	70196	88636	107783	127000	146134
Runtime (ms) of ALS-WR (Hadoop)	619625	623885	626749	639887	660833	693666

TABLE III. RUNNING TIME OF THREE ALGORITHMS WHEN ITERATIONS IS CONFIRMED IN MOVIELENS-20M

Number of features	2	5	8	11	14	17
Runtime (ms) of NALS-WR (Spark)	72000	90000	102000	126000	150000	168000
Runtime (ms) of SVD	101889	140637	179521	217848	256726	295984
Runtime (ms) of ALS-WR (Hadoop)	751387	769453	781549	800101	865819	937248