International Journal of Grid and Distributed Computing
Vol.7, No.2 (2014), pp.1-10
http://dx.doi.org/10.14257/ ijgdc.2014.7.2.01

Collaborative Filtering Recommendation using Matrix Factorization: A MapReduce Implementation

Xianfeng Yang [1] and Pengfei Liu [2]

[1] *School of Information Engineering, Henan Institute of Science and Technology, Xinxiang, Henan, P.R.CHINA*
[2] *HEBI College Of Vocation And Technology, Hebi, Henan, P.R.CHINA*
[1]*49377535@qq.com*, [2] *274772453@qq.com*

Abstract

Matrix Factorization based Collaborative Filtering (MFCF) has been an efficient method for recommendation. However, recent years have witness the explosive increasing of big data, which contributes to the huge size of users and items in recommender systems. To deal with the efficiency of MFCF recommendation in the context of big data challenge, we propose to leverage MapReduce programming model to re-implement MFCF algorithm. Specifically, we develop a four-step process of MFCF, each of which is implemented as MapReduce tasks. The experiments are conducted on a Hadoop cluster using a real world dataset of Netflix. The empirical results confirm the efficiency of our method.

Keywords: *Recommender system, Matrix Factorization, Collaborative Filtering, MapReduce*

1. Introduction

Collaborative filtering (CF) [1] has been successfully applied to many recommender systems [2-4]. The core of CF is to predict ratings for unseen users based on similar users whose ratings are known already. Koren *et al.*, [5] proposed a method based on Matrix Factorization (MF) to solve CF problem. Then, Matrix Factorization based Collaborative Filtering (MFCF) is widely used by other researchers [6-8].

However, the big data era [9] has arrived. As the numbers of users and items have been explosively growing, this contributes to a dramatic increase in the computation of MFCF. Fortunately, large scale parallel computing technique such as MapReduce [10] could be a efficient solution. The general idea is to distribute computation onto a cluster of nodes and leverage parallel computing.

MapReduce is a programming model for large scale parallel and distributed processing on clusters. One popular open-source implementation is Hadoop [11]. There are two basic procedures in MapReduce: Map and Reduce. Typically, the input and output are both in the form of key/value pairs. As shown in Figure 1, first the input component reads data by splits with appropriate size; Then, the Map procedure takes a series of key/value pairs, and generates processed key/value pairs, which are allocated to a particular reducer by certain partition function; Later, after data shuffling, the Reduce procedure iterates through the values that are associated with specific key and produces zero or more outputs. Typically, programmers only need to implement Map and Reduce procedure, while other details are handled by some mature platforms such as Hadoop.

To face with the challenge of big data and to deal with the time efficiency of MFCF problem, in this paper, we develop a MapReduce implementation of MFCF algorithm. We also evaluate its efficiency using real world Netflix dataset [2].

The remain of this paper is organized as follows. Section 2 provides some related work. Section 3 formulates the problem statement of MFCF, and then our MapReduce implementation method is proposed in Section 4. Empirical experiments are conducted in Section 5. Last the paper is concluded in Section 6.

ISSN: 2005-4262 IJGDC

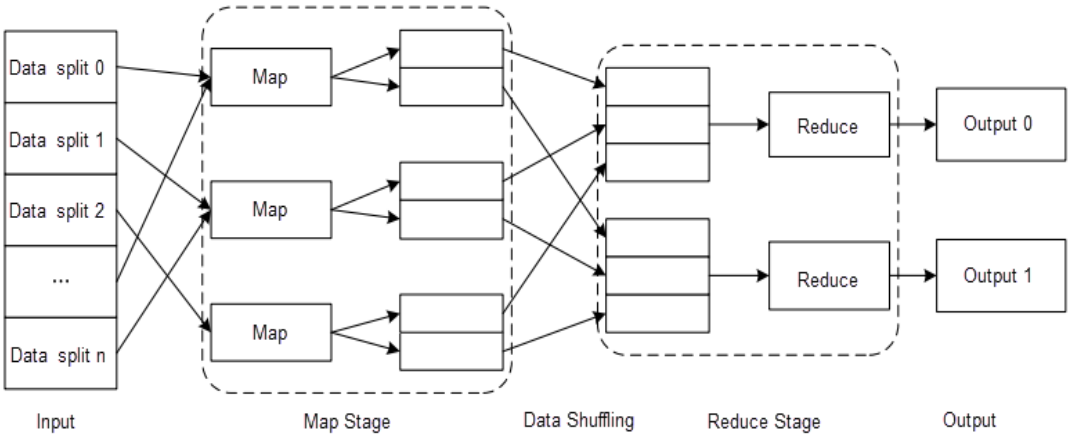

Figure 1. Description of MapReduce Programming Model

2. Related Work

Factor models especially Matrix Factorization (MF) methods have been introduced to CF recommender systems successfully. The general idea is to predict user vectors p_u for each user u and item vectors q_j for each item j. The objective is to minimize a loss function in their inner product explicitly [12] or implicitly [13].

Many MF algorithms such as SVD [14] aims to minimize the least squares loss function [15, 16]. However, Srebro *et al.*, [12] proposed to use a multi-class hinge loss and L_2 regularized to develop a model, and introduced Maximum Margin Matrix Factorization (MMMF) for Collaborative Filtering, which improves the generalization and performance.

The recent advent of parallel computing models such as MapReduce [10] has facilitated the large scale implementation of CF using matrix factorization. Gemulla *et al.*, [17] presented a parallel implementation of Stochastic Gradient Descent (SGD) using MapReduce model. Besides, Zhou *et al.*, [2] developed a parallel version of Alternating Least Squares (ALS) using parallelized MATLAB. Unlike existing efforts, we aims to implement MFCF using ALS method with MapReduce programming model. Instead of describing the details of mathematics, we emphasize on the functional implementation of MapReduce tasks.

3. Problem Statement

Suppose there are N users $\{u_1,u_2,...u_N\}$, M items $\{t_1,t_2,...t_M\}$ and r_{ij} notates the rating of user u_i on item t_j. Then we get a rating matrix $R^{N\times M}$, where the row label denotes user and column denotes item. In real world applications, $R^{N\times M}$ is a sparse matrix. The objective of CF based recommendation is to predict the missing ratings using known ratings. Typically, the precision of recommendation can be evaluated by RMSE (root mean squared error), calculated as:

$$RMSE = \sqrt{\frac{1}{|R|}\sum_{i=1}^{N}\sum_{j=1}^{M}(\hat{r}_{ij} - r_{ij})^2} \quad (1)$$

Where $|R|$ is the cardinality of the set of known ratings, $\hat{r}_{ij}$ is the predicted rating of user u_i on item t_j using CF technique, and r_{ij} is the known rating. The smaller the value of RMSE is, the more accurate the recommendation is.

When we solve the above recommendation problem using MF, two more matrices are introduced: user matrix $P^{N\times K}$ and item matrix $Q^{K\times M}$, where K is the number of features. The rows of matrix P are user features, notated as $p_u, u=1,2,...N$, and the rows of matrix Q are item features, notated as $q_j, j=1,2,...,M$. Therefore, the predicted rating in rating matrix R can be calculated as:

$$\hat{r}_{ij}=\sum_{k=1}^{K} p_{ik}q_{kj}=p_i q_j^T \tag{2}$$

Now we transform the problem into the learning of matrices P,Q given known ratings R. Popular methods to minimize Equation (1) include Stochastic Gradient Descent (SGD) [17] and Alternating Least Squares (ALS) [2]. In this paper, we employ ALS method to solve the Matrix Factorization based Collaborative Filtering (MFCF) recommendation problem, which is proved to be effective in existing research [6, 7].

4. MapReduce Implementation of MFCF

4.1. Overview

We split the data into training set and test set, where the former is used to learn matrices and the latter is used to determine proper parameters when the termination conditions are satisfied.

First, initialize item matrix Q using training data R_1:

$$q_{0j}=\frac{\sum_{i,j\in R_1} r_{ij}}{n_j}, 1\le j\le M \tag{3}$$

where r_{ij} is the ratings of users over items, and

$$q_{ij}=random(0.01,0.001), 0<i<k, 1\le j\le M \tag{4}$$

where q_{ij} is the i-th feature of item j.

We employ ridge regression [18] for learning characteristic matrices and calculating characteristic vectors using ALS, where regularization factor λ should be properly set. In this paper, the process works as follows. First, multiple λ will be initialized for learning characteristic matrices in parallel using MapReduce programming model. Then, λ and associated characteristic matrix with smallest RMSE value will be chosen for rating prediction. Therefore, after initialization, the characteristic vector in characteristic matrix for item matrix can be represented as:

$$\left\langle (j,\lambda_a), (q_{0j}, q_{1j}, ..., q_{ij}) \right\rangle \tag{5}$$

where j is the label of item, i is the label of feature, and $0\le a<num$, where num is the number of regularization factor λ.

4.2. Algorithm

The algorithm used in our MFCF problem using MapReduce programming model is composed of four tasks. Each item in dataset is represented as a triple (j,u,r), where j the

label of item is, u denotes user and r denotes corresponding rating.

The MapReduce implementation of MFCF is composed of four steps, as shown in Figure 2. First, update user matrix P with given training rating matrix R_1; Second, update item matrix Q with given R_1; Third, calculate the RMSE value using Equation (1), to learn the best parameter λ^*; Last, calculate the predicted rating for new user/item pairs using λ^*. Every step is implemented as MapReduce tasks, and will be discussed in following subsections.

4.2.1. User Matrix Update (P-Update): In this step, item matrix Q serves as input and is sent to multiple nodes. Then use training rating matrix R_1 to calculate user matrix P. Figure 3 gives the pseudocode description of P-Update. The input for this step includes training rating matrix R_1, item matrix Q, a sorted list of parameters $\lambda_1, \lambda_2, ..., \lambda_a$, the number of features k, and the algorithm returns user matrix P.

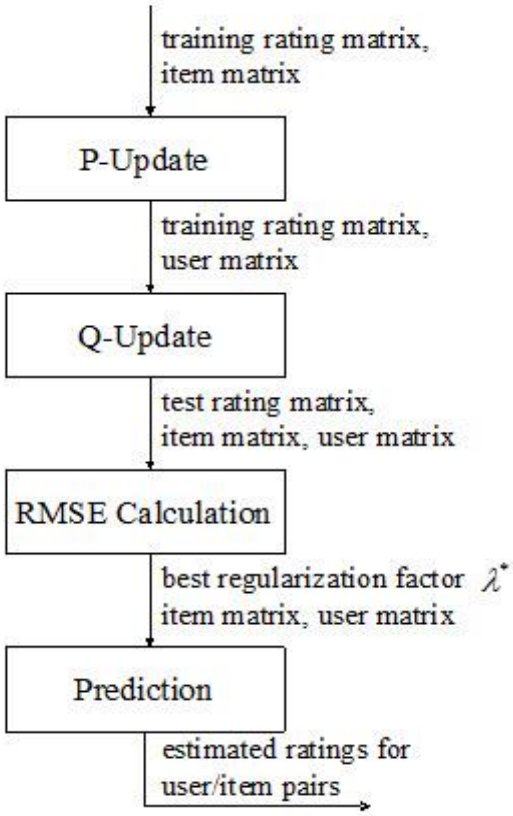

Figure 2. Flow Chart of MapReduce Implementation of MFCF

The complexity of P-Update mainly stems from the process of ridge regression during REDUCE stage. The average time complexity of REDUCE at each iteration is approximately $O(K^3 + 2Kn_u + \log|\lambda||I|)$, where $|I|$ is the number of items, n_u is the number of items that have been rated by users, K is the number of features, and $|\lambda|$ is the number of pre-defined parameters. Since the matrix is typically sparse, we suppose $n_u << |I|$ and is a constant. Therefore, the total time complexity of algorithm 1 can be calculated as $O(\frac{|\lambda||U|\cdot(K^3 + 2K + N\log|\lambda||I|)}{N})$, where $|U|$ is the number of users, and $|N|$ is

the number of nodes processing REDUCE tasks.

4.2.2. Item matrix update (Q-Update): This step does similar execution as P-Update. Given user matrix P and training rating data R_1, the objective is to learn item matrix Q. Figure 4 illustrates the pseudocode of Q-Update. The input for this step includes user matrix P, training rating matrix R_1, a sorted list of parameters $\lambda_1, \lambda_2, ..., \lambda_a$, the number of features k, and the algorithm returns item matrix Q. Similarly, the total time complexity of algorithm 2 can be calculated as $O(\frac{|\lambda \| I| \cdot (K^3 + 2K + N \log |\lambda \| U|)}{N})$, where $|U|$ is the number of users, and $|N|$ is the number of nodes processing REDUCE tasks.

Algorithm 1 P-Update

```
1: procedure MAP(line_key, (j, u, r))
2:     emit (u, (j, r))
3: end procedure
4: procedure REDUCE((u, [(j_0, r_0), (j_1, r_1), ..., (j_n, r_n)]))
5:     for λ in (λ_0, λ_1, ..., λ_a) do
6:         for x in range (0, n) do
7:             set (j_x, λ, Q) to Qu_0k
8:             set (λ × n × I + Qu^T Qu)^-1 to R_k×k
9:             for x in range (0, k) do
10:                set Σ_z=0^n (Σ_y=0^k-1 R_xy Qu_yz) × r_z to P_ux
11:            end for
12:            emit ((u, λ), [P_u0, P_u1, ..., P_uk])
13:        end for
14:    end for
15: end procedure
```

Figure 3. P-Update Algorithm Description

4.2.3. RMSE Calculation: Once matrices P, Q are learned from Sections 4.2.1 and 4.2.2, estimated ratings $\hat{r}_{ij}$ can be predicted by Equation (2). Then, RMSE values can be calculated using test rating dataset by Equation (1). The best λ and corresponding RMSE will be emitted after the reduce procedure. Figure 5 illustrates the process of RMSE calculation. The input of this algorithm includes test rating data R_2, user matrix P, item matrix Q, a sorted list of parameters $\lambda_1, \lambda_2, ..., \lambda_a$, the number of features k, and the output should be each λ and their corresponding RMSE values.

The computation cost of RMSE calculation mainly comes from the MAP stage. The average time complexity of MAP function at each iteration is approximately calculated as $O(\frac{|\lambda \| R_2| \cdot (K + N \log |\lambda \| I| + N \log |\lambda \| U|)}{N})$, where $|R_2|$ is the size of test instances in test rating data.

Algorithm 2 Q-Update

```
1: procedure MAP(line_key, (j, u, r))
2:     emit (j, (u, r))
3: end procedure
4: procedure REDUCE((j, [(u_0, r_0), (u_1, r_1), ..., (u_n, r_n)])
5:     for λ in (λ_0, λ_1, ..., λ_a) do
6:         for x in range (0, m) do
7:             set (u_x, λ, P) to q_{k0} P
8:             set (λ × m × I + q^T P q P)^{-1} to R_{k×k}
9:             for x in range (0, k) do
10:                set Σ_{z=0}^{m} (Σ_{y=0}^{k-1} R_{xy} q_{yz} P) × r_z to Q_{xj}
11:            end for
12:            emit ((j, λ), [Q_{0j}, Q_{1j}, ..., Q_{kj}])
13:        end for
14:    end for
15: end procedure
```

Figure 4. Q-Update Algorithm Description

Algorithm 3 RMSE calculation

```
1: procedure MAP(line_key, (j, u, r))
2:     for λ in (λ_0, λ_1, ..., λ_a) do
3:         set (u, λ, P) to [p_1, p_2, ..., p_k]
4:         set (j, λ, Q) to [q_1, q_2, ..., q_k]
5:         set Σ_{x=0}^{k} p_x q_x to r_c
6:         set (r_c − r)^2 to SE
7:         emit (λ, SE)
8:     end for
9: end procedure
10: procedure REDUCE((λ, [SE_1, SE_2, ..., SE_n])
11:     set √((Σ_{x=1}^{n} SE_x)/n) to RMSE
12:     emit (λ, RMSE)
13: end procedure
```

Figure 5. RMSE Calculation Algorithm Description

4.2.4. Prediction: When RMSE value calculated in last step does not change any more, or after the iteration times exceed the maximum pre-defined parameter, we can get the best λ^* with smallest RMSE value $RMSE^*$ together with the characteristic matrices P^*, Q^*. As shown in Figure 6, the prediction algorithm is a MAP only task. Note that unlike the MAP function in algorithm 3, here the input rating data is unknown. The output is the predicted rating of each user/item pair.

Algorithm 4 Prediction

```
1: procedure MAP(line_key, (u, j))
2:     set (u, λ*, P*) to [p_1, p_2, ..., p_k]
3:     set (j, λ*, Q*) to [q_1, q_2, ..., q_k]
4:     set Σ_{x=0}^{k} p_x q_x to r̂
5:     emit ((u, j), r̂)
6: end procedure
```

Figure 6. Rating Prediction Algorithm Description

5. Experiments

We configure the environment for experiments as follows. We have 4 PCs with 3.00G Hz Intel dual-core processors, 2GB RAM and 160G disk storage for our MapReduce cluster. We assign one as *Namenode* and *JobTracker*, and the rest three as computing nodes.

5.1. Speedup Performance Evaluation

We set 10 λ values, starting from 0.01 to 0.10 in steps of 0.01, and the number of features $k = 400$. The dataset we used in the experiments is sampled from Netflix [2]. The rating data is 5-scale numbers, *i.e.*, from 1 to 5. We sampled two different datasets, as described in Table 1, where item matrix Q is initialized using features, and user matrix P is obtained after the P-Update process.

Table 1. Description of Datasets

Dataset	Number of users	Number of items	Data size	Size of Q	Size of P
1	3,486	3,547	2.8MB	115.2MB	110.4MB
2	24,102	12,255	28.1MB	401.5MB	837.0MB

The speedup performance of P-Update and RMSE calculation tasks are depicted in Figures 7 and 8 respectively, with varied numbers of nodes in comparison of two datasets. From the figures, we have several observations. (1) As the number of nodes increases, the process time reduces and the speedup increases obviously. Especially for P-Update task in Figure 7, the performance with different numbers of nodes is close to linear speedup. This observation proves that the more the numbers of nodes are deployed, the less the time cost is. (2) Larger dataset outperforms smaller one, especially for RMSE calculation task, as shown in Figure 8. For small dataset, although the time complexity is low, extra costs on communication between nodes and task scheduling are significant for the performance as well.

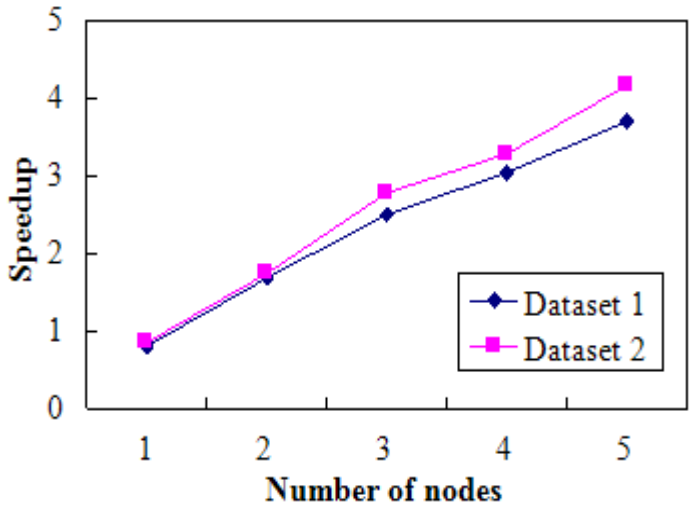

Figure 7. Speedup of P-Update Task

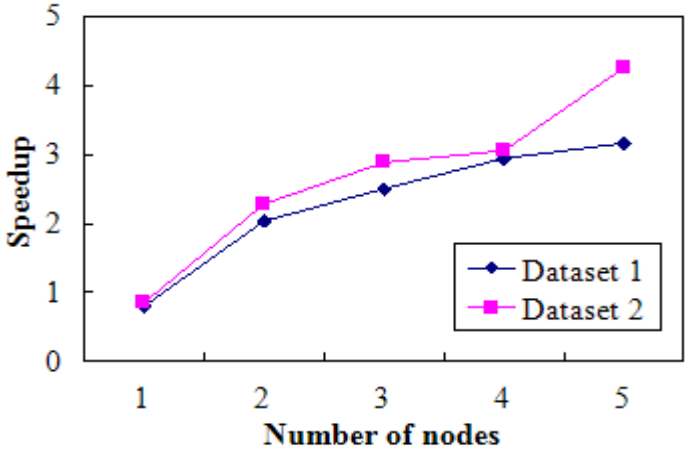

Figure 8. Speedup of RMSE Calculation Task

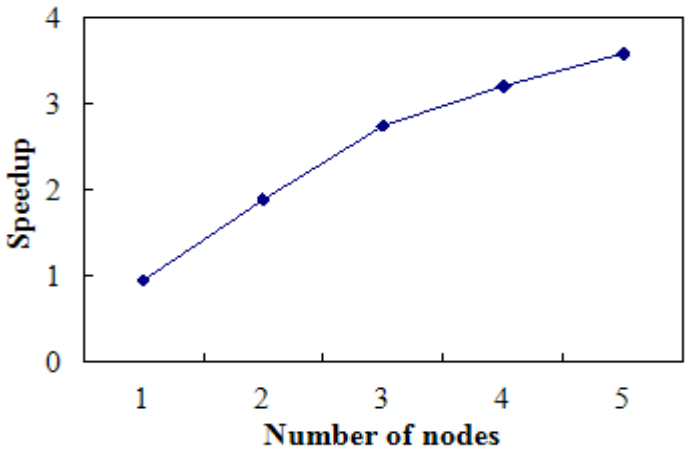

Figure 9. Speedup of Test Data

Table 2. λ and RMSE at Each Iteration

λ	RMSE
0.01	0.9704
0.02	0.9598
0.03	0.9412
0.04	0.9303
0.05	0.9211
0.06	0.9225
0.07	0.9228
0.08	0.9229
0.09	0.9324

The overall speedup for testing new dataset is shown in Figure 9. The overall performance is good because the cost is mainly from the iterative processing during P-Update and Q-Update. As shown in Table 2, the values of λ and RMSE are calculated at each iteration, and here the number of features is set as $k = 100$. We can see $\lambda = 0.05$ should be chosen because the corresponding RMSE value is smallest (RMSE = 0.9211).

5.2. I/O Cost

Note that (u, λ, P) and (j, λ, Q) are stored in MapFile [11]. Since those files are too big to fit memory, the I/O cost comes from reading data from external files, such as line 7 in Algorithms 1 and 2. We compute the I/O cost T_{io} as the average of 10 times processing, that is, 1.72 minutes, while the total time cost of the overall process is about 35.88 minutes. Therefore, the proportion of I/O cost is approximately 4.8%, which is not significant compared to the computing cost. We can see that our MapReduce implementation is efficient enough in that I/O cost is not a concern.

6. Conclusion

In this paper, we propose a MapReduce implementation of Collaborative Filtering using Matrix Factorization (MFCF) for recommender systems. Specifically, we split the target objective into four steps, and implement each step as MapReduce tasks. The proposed algorithm is deployed on a cluster of Hadoop nodes. Experiments are conducted on real world Netflix dataset, and the results prove the efficiency of our method. In future, we will try to extend our work by exploring MapReduce implementation of MFCF on streaming data.

References

[1] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques", Advances in artificial intelligence, **(2009)** April.
[2] Y. Zhou, D. Wilkinson, R. Schreiber and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize", Algorithmic Aspects in Information and Management, Springer Berlin Heidelberg, **(2008)**, pp. 337-348.
[3] J. B. Schafer, D. Frankowski, J. Herlocker and S. Sen, "Collaborative filtering recommender systems", The adaptive web, Springer Berlin Heidelberg, **(2007)**, pp. 291-324.
[4] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-based collaborative filtering recommendation algorithms", Proceedings of the 10th international conference on World Wide Web, ACM, **(2001)**, pp. 285-295.
[5] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights", Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference, IEEE, **(2007)**, pp. 43-52.
[6] J. D. M. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction", Proceedings of the 22nd international conference on Machine learning, ACM, **(2005)**, pp. 713-719.
[7] K. Yu, S. Zhu, J. Lafferty and Y. Gong, "Fast nonparametric matrix factorization for large-scale collaborative filtering", Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, ACM, **(2009)**, pp. 211-218.
[8] Y. Koren, R. Bell and C. Volinsky, "Matrix factorization techniques for recommender systems", Computer, vol. 42, no. 8, **(2009)**, pp. 30-37.
[9] P. Zikopoulos and C. Eaton, "Understanding big data: Analytics for enterprise class hadoop and streaming data", McGraw-Hill Osborne Media, **(2011)**.
[10] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Communications of the ACM, vol. 51, no. 1, **(2008)**, pp. 107-113.
[11] T. White, "Hadoop: the definitive guide", O'Reilly, **(2012)**.
[12] N. Srebro, J. Rennie and T. S. Jaakkola, "Maximum-margin matrix factorization", Advances in neural information processing systems, **(2004)**, pp. 1329-1336.
[13] T. Hofmann, "Latent semantic models for collaborative filtering", ACM Transactions on Information Systems (TOIS), vol. 22, no. 1, **(2004)**, pp. 89-115.
[14] C. Boutsidis and E. Gallopoulos, "SVD based initialization: A head start for nonnegative matrix factorization", Pattern Recognition, vol. 41, no. 4, **(2008)**, pp. 1350-1362.
[15] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, "Scalable collaborative filtering approaches for large recommender systems", The Journal of Machine Learning Research, vol. 10, **(2009)**, pp. 623-656.
[16] Y. Koren, "Collaborative filtering with temporal dynamics", Communications of the ACM, vol. 53, no. 4 **(2010)**, pp. 89-97.

[17] R. Gemulla, E. Nijkamp, P. J. Haas and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent", Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, **(2011)**, pp. 69-77.
[18] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems", Technometrics, vol. 12, no. 1, **(1970)**, pp. 55-67.

Authors

Xianfeng Yang, He received his bachelor's degree in Computer Application Technology from Henan Normal University, Xinxiang, Henan, China, in 2001,the master's degree in Computer Application Technology from China University of Petroleum, Dongying, China, in 2007. He is now a lecturer at School of Information Engineering, Henan Institute of Science and Technology, Xinxiang, China. His current research interests include pattern recognition, image processing, neural networks, natural language processing.

Pengfei Liu, He received his bachelor's degree in Computer Science and Technology from Anyang Normal University, Anyang, China, in 2004, the master degree in Computer Application from Huazhong University of Science and Technology, Wuhan, China, in 2010. He is now a lecturer at Hebi College of Vocation And Technology. His current research interests include computer network, web application, network operating system.

