ICIC Express Letters
Part B: Applications
Volume **7**, Number **11**, November **2016**
 ISSN 2185-2766
pp. **2397–2403**

A HYBRID COLLABORATIVE FILTERING ALGORITHM ON SPARK

Weiwei Xing, Wentao Zhou and Weidong Wang

School of Software Engineering
Beijing Jiaotong University
No. 3, Shangyuancun, Haidian District, Beijing 100044, P. R. China
wtzhou@bjtu.edu.cn

Received May 2016; accepted August 2016

Abstract. *Recommendation algorithms are well recognized as the convenient and flexible way of user's interest discovery in e-commerce industry. As a branch of recommendation systems, collaborative filtering is a classical and widely used technique, which provides personalized recommendations according to users' requirements. However, it may suffer from problem of data sparsity, which commonly impacts on users' preference predictions. To address this problem, in this paper, we propose a novel hybrid collaborative filtering algorithm using MovieLens dataset. The algorithm contains three steps. 1) We employ the co-clustering with augmented matrix model to get user clusters and movie clusters simultaneously. 2) Users' (movies') proper similar user (movie) collections are filtered from appropriate user (movie) clusters using Top-K strategy. 3) Based on the above similar collections, the user-based and movie-based methods are adopted respectively, to obtain individual predictions for an unknown movie's rating, and then we combine these two predictions via a linear combination for accuracy improvement. We implement our algorithm on a Spark platform for good scalability. Experiments show that the proposed algorithm is more robust against data sparsity and scalability, and it achieves better recommendations than traditional recommendation algorithms.*
Keywords: Recommendation system, Collaborative filtering, Information theory, Co-clustering, Spark

1. **Introduction.** Collaborative filtering is widely used in many commercial recommendation systems. However, its widespread applications have encountered serious challenges, such as sparsity and scalability [1]. The main reason for the sparsity problem is a lack of valid data. Meanwhile, the main concern for the scalability problem is the multi-cluster computation.

In order to solve the sparsity problem, many methods (e.g., matrix factorization, clustering and co-clustering) are raised in the past years. Clustering is useful in grouping together similar objects as a common data mining technique. However, most of the cluster methods focus on one-way cluster. Dhillon et al. proposed a new clustering method called information-theoretic co-clustering (ITCC) [2], which was to optimize the objective function based on the loss of mutual information between clustered random variables, before and after co-clustering. Based on ITCC, Liang and Leng proposed a novel collaborative filtering method [3], which applied ITCC to simultaneously cluster the rows and columns of the user-item rating matrix, and subsequently computed the cluster preference similarity between users/items. It showed that co-clustering collaborating filtering method was better than one-way clustering method. The co-clustering with augmented data matrices (CCAM) algorithm is another method to handle sparse data [4]. The CCAM is to minimize the mutual information loss of a linear combination of rating data and content-based information, where the mutual information can measure the dependency between two objects among those matrices. Wu et al. combined CCAM with recommendation algorithm and showed that it could reduce the error of prediction and

sparsity problem [5]. They also realized CCAM algorithm's parallel version in a Hadoop cluster [6]. Although CCAM solved the problem of sparsity, it was too slow because of the I/O operations on HDFS.

In this paper, we propose a novel hybrid collaborative filtering algorithm based on co-clustering with augmented matrix (CCAM) in a Spark cluster, named S-CCAM. Firstly, we apply CCAM to simultaneously cluster the rows and columns of the user-movie rating matrix as well as movie feature matrix and user profile matrix. Secondly, users' (movies') proper similar user (movie) collections are filtered from appropriate user (movie) clusters using Top-K strategy. Finally, based on users' (movies') similar user (movie) collections, we compute the predictions of user-based method and item-based method respectively, and then fuse these two predictions via a linear combination. The contribution of this paper is twofolds:

1) Information theory is used to calculate the distance between a single user (movie) and a user (movie) cluster for similarity calculation.

2) The proposed algorithm can be implemented on a Spark platform, which makes it faster than other traditional algorithms and meanwhile achieves higher scalability.

The rest of this paper is organized as follows. Section 2 presents the details of the proposed algorithm. The experiments and the conclusions are presented in Section 3 and Section 4, respectively.

2. The Proposed Algorithm.

Traditional collaborative filtering algorithms often confront the problem of sparsity and scalability. To overcome these problems, we propose a novel hybrid collaborative filtering algorithm based on CCAM in a Spark cluster, named S-CCAM, since the CCAM is a co-clustering-based algorithm that can effectively solve the problem of sparsity. Currently, in the Big Data computing field, for the sake of the computation speed, CCAM has been well implemented on Hadoop [7]. However, it may have a barrier on the frequent I/O operation and data locality when handling a great number of intermediate results. Therefore, we employ the Spark technique [8] to further improve the CCAM for time saving and scalability. Compared with the CCAM algorithm on Hadoop, the proposed S-CCAM has better performance on processing speed, which will be proved in later experiment section. The S-CCAM algorithm is mainly divided into three steps: 1) co-clustering; 2) similarity calculation; and 3) prediction.

2.1. Co-clustering.

The first step of S-CCAM algorithm is co-clustering, to get item clusters and user clusters, simultaneously. The MovieLens dataset consists of three input data, the user-movie rating matrix, user profile matrix and the movie feature matrix.

Firstly, these three matrices are normalized into $f(I,U)$, $h(U,L)$ and $g(I,S)$ respectively. Then CCAM is used to get user clusters and movie clusters simultaneously. The goal of CCAM is to find a co-clustering $\left(\hat{I},\hat{U}\right)$ that minimizes:

$$q\left(\hat{I},\hat{U}\right) = D\left(f(I,U)\middle\|\hat{f}(I,U)\right) + D\left(g(I,S)\middle\|\hat{g}(I,S)\right) + D\left(h(U,L)\middle\|\hat{h}(U,L)\right) \tag{1}$$

where $D(x \parallel y)$ means the K-L divergence between x and y.

The co-clustering optimization task can be solved using the following update rules:

$$C_I^{(t+1)}(i) = \arg\min_{\hat{i}} \left[f(i)D\left(f(u \mid i)\middle\|\hat{f}^{(t)}\left(u \mid \hat{i}\right)\right) + \lambda \cdot g(i)D\left(g(s \mid i)\middle\|\hat{g}^{(t)}\left(s \mid \hat{i}\right)\right)\right] \tag{2}$$

$$\begin{aligned} C_U^{(t+2)}(u) = \arg\min_{\hat{u}} \Big[&f(u)D\left(f(i \mid u)\middle\|\hat{f}^{(t+1)}(i \mid \hat{u})\right) \\ &+\varphi \cdot h(u)D\left(h(l \mid u)\middle\|\hat{h}^{(t+1)}(l \mid \hat{u})\right)\Big] \end{aligned} \tag{3}$$

For each row (movie), we use Formula (2) to find its new cluster index, and then the probability distribution $\hat{f}^{(t+1)}(I,U)$ and $\hat{g}^{(t+1)}(I,S)$ are updated. After this, for each

column (user), we use formula (3) to find its new cluster index, and then the probability distribution $\hat{f}^{(t+2)}(I,U)$ and $\hat{h}^{(t+2)}(U,L)$ are updated. The iteration process is terminated when the result of $q^{(t)}\left(\hat{I},\hat{U}\right) - q^{(t+2)}\left(\hat{I},\hat{U}\right)$ is convergent. Spark is very suitable for this iteration process, because the intermediate results stored on memory will be used frequently. The function $groupByKey()$ in Spark can send the information that contains the same key to the same nodes, which is very convenient to find the optimal cluster index for movie or user. After the co-clustering process, we obtain a set of movie clusters and a set of user clusters, we save the result in movie cluster RDD with formatter (movieID, movieClusterID) and user cluster RDD with formatter (userID, userGroupID). Wu et al. proved that the above iteration process could monotonically decrease the loss of mutual information [4].

2.2. **Similarity calculation.** The idea of collaboration filtering is that people with similar preferences would rate items similarly, so we need to get users' and movies' similarity. In order to get higher accuracy, we first define the distance between a movie and a movie cluster as well as a user and a user cluster. We use the KL-divergence that Formula (1) used to define the above two distances:

$$dist\left(i,\hat{i}\right) = f(i)D\left(f(u \mid i)\middle\|\hat{f}\left(u \mid \hat{i}\right)\right) + \lambda \cdot g(i)D\left(g(s \mid i)\middle\|\hat{g}\left(s \mid \hat{i}\right)\right) \tag{4}$$

$$dist\left(u,\hat{u}\right) = f(u)D\left(f(i \mid u)\middle\|\hat{f}\left(i \mid \hat{u}\right)\right) + \varphi \cdot h(u)D\left(h(l \mid u)\middle\|\hat{h}\left(l \mid \hat{u}\right)\right) \tag{5}$$

To calculate the distance between a movie and a movie cluster, user-movie rating file, movie feature file and movie cluster RDD are used by Formula (4) and the results are saved in movie-movie-cluster RDD with formatter (movieID, movieClusterID, and dist). The same way is used to calculate the distance between a user and a user cluster by Formula (5) and the results are saved in user-user-cluster RDD with formatter (userID, userClusterID, and dist).

Then we calculate the movies' and users' similarities using cosine similarity defined as:

$$sim(i_1,i_2) = \frac{\sum_{u \in U_{i_1,i_2}} (R_{u,i_1} \times R_{u,i_2})}{\sqrt{\sum_{u \in U_{i_1,i_2}} R_{u,i_1}^2} \times \sqrt{\sum_{u \in U_{i_1,i_2}} R_{u,i_2}^2}} \tag{6}$$

$$sim(u_1,u_2) = \frac{\sum_{i \in I_{u_1,u_2}} (R_{u_1,i} \times R_{u_2,i})}{\sqrt{\sum_{i \in I_{u_1,u_2}} R_{u_1,i}^2} \times \sqrt{\sum_{i \in I_{u_1,u_2}} R_{u_2,i}^2}} \tag{7}$$

where $R_{u,i}$ is the rating of user u on movie i. Given a specified movie, we pick up the top $K1$ nearest movie clusters, and from which we use Formula (6) to select the top $k1$ nearest movie neighbors. The output of this step is movie-sim RDD with formatter (movieID, List(movieID, similarity)). We use the same method to get the user-sim RDD with formatter (userID, List(userID, similarity)). We make use of $groupByKey()$ and $sortByKey()$ functions in Spark to parallel our algorithm and get higher accuracy as much as possible.

2.3. **Prediction.** The purpose of prediction is to predict the possible rating of a given user on a given movie. For item-based method, we use movie-sim RDD and user-movie rating file to get item-based-pre RDD with formatter (userID, movieID, and p-rating) by

$$P_{u,i}^{(i)} = \bar{R}_i + \frac{\sum_{j \in s(i)} \left(sim(i,j) \times \left(R_{u,j} - \bar{R}_j\right)\right)}{\sum_{j \in s(i)} |sim(i,j)|} \tag{8}$$

where $\bar{R}_i$ and $\bar{R}_j$ mean the average rating on movie i and movie j. And for user-based method, we use user-sim RDD and user-movie rating file to get user-based-pre RDD with

formatter (userID, movieID, and p-rating) by

$$P_{u,i}^{(u)} = \bar{R}_u + \frac{\sum_{n \in s(u)} \left(sim(u,n) \times \left(R_{n,i} - \bar{R}_n \right) \right)}{\sum_{n \in s(u)} |sim(u,n)|} \tag{9}$$

where $\bar{R}_u$ and $\bar{R}_n$ mean the average rating of user u and user n.

Relying on item-based prediction or user-based prediction only is undesirable, especially when the ratings from these two predictions are often not available. To improve the accuracy of prediction, we fuse item-based and user-based predictions. By linearly combining the previous two types of predictions, we obtain the final prediction result as

$$P_{u,i} = \alpha \cdot P_{u,i}^{(i)} + (1 - \alpha) \cdot P_{u,i}^{(u)} \qquad (0 \leq \alpha \leq 1) \tag{10}$$

where α determines the relative importance of the item-based and user-based predictions.

3. Experiment Results.

3.1. Dataset.

To evaluate the proposed algorithm, we adopt Movielens dataset provided by GroupLens Research. Two datasets, 100K and 1M datasets with user-movie rating data, as well as movie features and user profiles, are available to be used. For 100K dataset, the main user-movie rating matrix comprises of 100000 ratings records (scaled from 1 to 5) provided by 943 users on 1682 movies. As for 1M dataset, there are 6040 users and 3676 movies with 1 million ratings. In order to perform the five-fold cross validation, we divide each of the datasets into 80% training set and 20% testing set.

3.2. Evaluation metric.

To evaluate the accuracy of our proposed algorithm, we adopt Mean Absolute Error (MAE). MAE is defined as

$$MAE = \frac{\sum_{i=1}^{N} |p_i - r_i|}{|N|} \tag{11}$$

where N denotes the number of ratings, and p_i is the predicted rating whereas r_i is the real rating. A smaller value of MAE means a better result.

3.3. Parameter tuning.

For many hybrid recommendation systems, parameter tuning is an unavoidable issue that we need to deal with. The CCAM model, which makes use of content-based technique, has two parameters λ and φ that control item features and user profiles. We also need to know the number of item clusters s and the number of user groups t, and usually we set $s = t$. For prediction, we need to specify the number of nearest items k_1 in K_1 nearest item clusters and the number of nearest users k_2 in K_2 nearest user groups. In our algorithm, we set $k_1 = k_2 = 20$ and $K_1 = K_2 = 5$. We also need to specify the effect of fusion coefficient α.

3.3.1. *Impact of cluster size s.* Different clustering algorithms have their own optimal parameters of cluster size s. Therefore, we tune the cluster size s for K-means, S-ITCC and S-CCAM under $s = 5$, 10, 20 and 40. For S-CCAM, we set $\lambda = 0.01$ and $\varphi = 0.01$. Figures 1(a) and 1(b) show the MAEs of different clustering algorithms on ML-100K and ML-1M. From the figures, we can see when the cluster size s ranges from 10 to 20 for all algorithms, the MAEs are lower. So we choose s to be 10 for all algorithms on ML-100K and ML-1M in the following experiments.

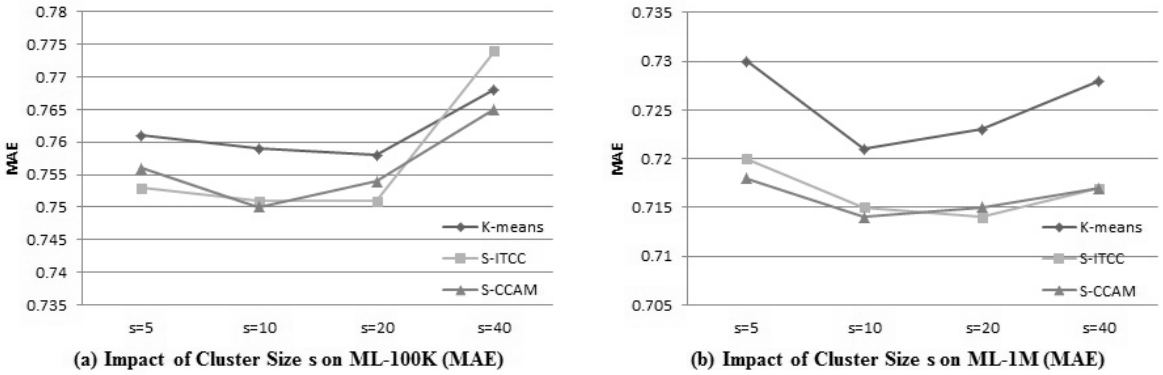

FIGURE 1. Impact of cluster size s on ML-100K and ML-1M

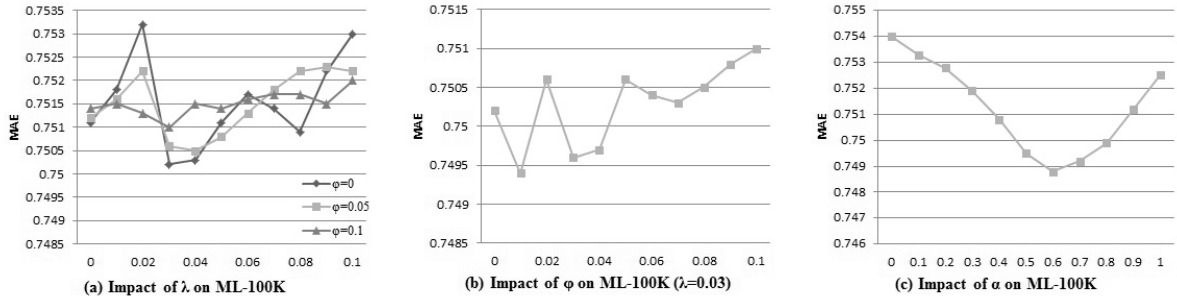

FIGURE 2. Impact of λ, φ and α on ML-100K

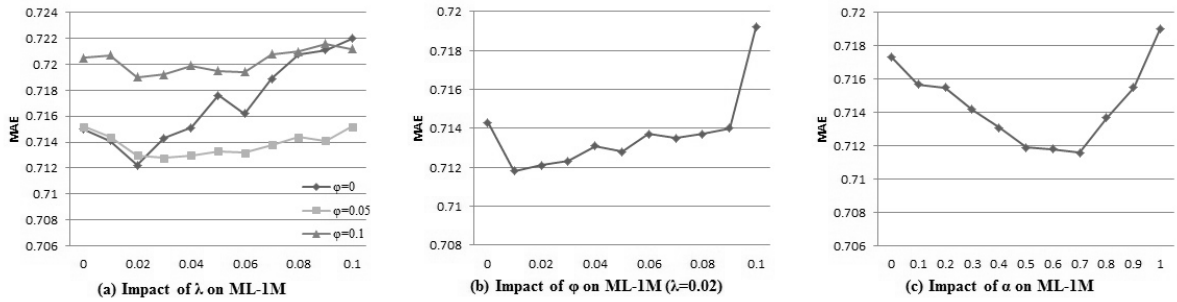

FIGURE 3. Impact of λ, φ and α on ML-1M

3.3.2. *Impact of augmented matrix weight λ and φ.* The parameter λ for item feature weight and φ for user profile weight are used to control the influence of augmented matrices. In our algorithm, we first set cluster size $s = 10$ and user profile weight $\varphi = 0.1$ when tuning item feature weight λ from 0 to 0.1 with step size 0.01. As we can see from Figures 2(a) and 3(a), the optimal λ for ML-100K is 0.03 and the best setting for ML-1M is $\lambda = 0.02$. Then we use the same way to tune φ. As shown in Figures 2(b) and 3(b), we find the best setting for both ML-100K and ML-1M is $\varphi = 0.01$; therefore, we use $\varphi = 0.01$ in the following experiments.

3.3.3. *Effect of fusion coefficient α tuning.* As described in Section 2, the parameter α determines the relative importance of the item-based and user-based predictions. We conduct experiments to identify the optimal fusion coefficient. We set $s = 10$, $\lambda = 0.03$ and $\varphi = 0.01$ for ML-100K, and meanwhile we set $s = 10$, $\lambda = 0.02$ and $\varphi = 0.01$ for ML-1M. Our results are shown in Figures 2(c) and 3(c). In the case of ML-100K dataset, the prediction accuracy increases as we increase α from 0 to 0.6; after 0.6, it becomes

worse. In the case of ML-1M dataset, the accuracy is improved as we increase α from 0 to 0.7. So it proves that the combined prediction can be more accurate than the pure rating prediction.

3.4. **Performance comparison.** In addition to the three clustering algorithms (K-Means, S-ITCC, and S-CCAM), we also compare our algorithm with pure item-based collaborative filtering and pure user-based collaborative filtering. Figure 4 shows the performance comparison on ML-100K and ML-1M, respectively. We observe that S-CCAM outperforms all the other algorithms on both ML-100K and ML-1M datasets.

3.5. **Scalability testing.** In this part, we evaluate the scalability of our algorithm on Spark platform. Our Spark cluster is composed of 4 computers, and each computer has 4-core CPU and 8GB memory. In this experiment, we use the ML-1M dataset. Figure 5 gives the scalability among different computers in terms of speed-up. The speed-up value is calculated based on the run-time on n computers T_n divided by that on one computer T_1.

$$\text{Speed-up} = \frac{T_1}{T_n} \tag{12}$$

As shown in Figure 5, the scalability performs well when the number of computers is greater than two.

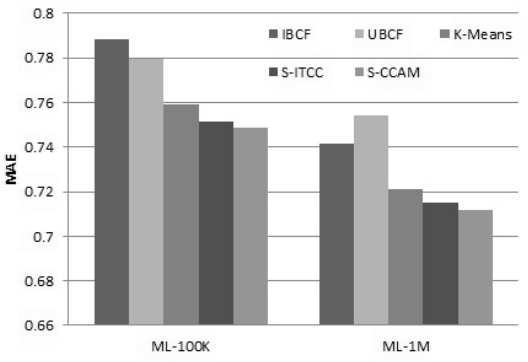

FIGURE 4. Result comparison on IBCF, UBCF, K-Means, S-ITCC and S-CCAM

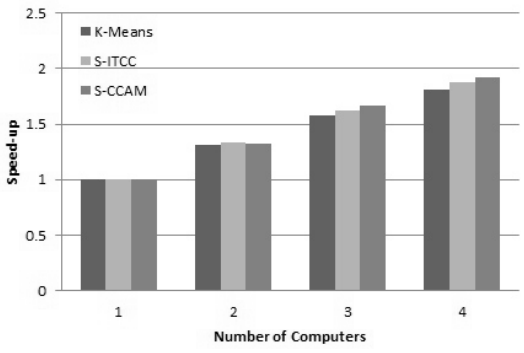

FIGURE 5. Speed-up comparison on K-Means, S-ITCC and S-CCAM

4. **Conclusions.** In this paper, we propose a novel hybrid collaborative filtering algorithm in a Spark cluster, which can solve the problem of sparsity and scalability. The algorithm is composed of three steps: co-clustering, similarity calculation and prediction. Experiments show that the proposed algorithm can improve the accuracy of recommendation on sparse datasets. Besides, it has obvious advantages in speed and scalability because of a large number of iteration processes. In the future, we will investigate better co-clustering models and improve the performance of the proposed algorithm further.

Acknowledgment. This work is supported in part by the National Natural Science Foundation of China (Nos. 61100143, 61272353, 61370128, and 61428201), the Program for New Century Excellent Talents in University (NCET-13-0659), the Beijing Higher Education Young Elite Teacher Project (YETP0583), and the Fundamental Research Funds for the Central Universities (2014JBZ004).

REFERENCES

[1] A. Kumar and A. Sharma, Alleviating sparsity and scalability issues in collaborative filtering based recommender systems, *Proc. of the International Conference on Frontiers of Intelligent Computing: Theory and Applications*, 2013.

[2] I. S. Dhillon, S. Mallela and D. S. Modha, Information-theoretic co-clustering, *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.89-98, 2003.

[3] C. Liang and Y. Leng, Collaborative filtering based on information-theoretic co-clustering, *International Journal of Systems Science*, vol.45, no.3, pp.589-597, 2013.

[4] M.-L. Wu, C.-H. Chang and R.-Z. Liu, Co-clustering with augmented matrix, *Applied Intelligence*, vol.39, no.1, pp.153-164, 2013.

[5] M.-L. Wu, C.-H. Chang and R.-Z. Liu, Integrating content-based filtering with collaborative filtering using co-clustering with augmented matrices, *Expert Systems with Applications*, vol.41, no.6, pp.2754-2761, 2014.

[6] M.-L. Wu and C.-H. Chang, Parallel co-clustering with augmented matrices algorithm with mapreduce, *Data Warehousing and Knowledge Discovery*, pp.183-194, 2014.

[7] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 2012.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, Spark: Cluster computing with working sets, *Usenix Conference on Hot Topics in Cloud Computing*, vol.15, no.1, pp.1765-1773, 2010.