Research Article

Improving Accuracy and Scalability of Personal Recommendation Based on Bipartite Network Projection

Fengjing Yin, Xiang Zhao, Xin Zhang, Bin Ge, and Weidong Xiao

National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Fengjing Yin; yinfengjing@nudt.edu.cn

Received 10 April 2014; Accepted 21 August 2014; Published 25 September 2014

Academic Editor: Lu Zhen

Copyright © 2014 Fengjing Yin et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Bipartite network projection method has been recently employed for personal recommendation. It constructs a bipartite network between users and items. Treating user taste for items as resource in the network, we allocate the resource via links between user nodes and item nodes. However, the taste model employed by existing algorithms cannot differentiate “dislike” and “unrated” cases implied by user ratings. Moreover, the distribution of resource is solely based on node degrees, ignoring the different transfer rates of the links. To enhance the performance, this paper devises a negative-aware and rating-integrated algorithm on top of the baseline algorithm. It enriches the current user taste model to encompass “like,” “dislike,” and “unrated” information from users. Furthermore, in the resource distribution stage, we propose to initialize the resource allocation according to user ratings, which also determines the resource transfer rates on links afterward. Additionally, we also present a scalable implementation in the MapReduce framework by parallelizing the algorithm. Extensive experiments conducted on real data validate the effectiveness and efficiency of the proposed algorithms.

1. Introduction

With the rapid growth of the World Wide Web, people are emerged in an overwhelming amount of information, which makes it difficult to obtain relevant information of interest. Personal recommendation, as a promising method to overcome information overload, is employed to suggest products to the consumers who may be interested, such as news, books, music, and movies. As a consequence, a large number of diverse algorithms were proposed to solve the problem. Besides the classic methods, for example, content-based methods [1], collaborative filtering, and variants [2–4], some new paradigms have been introduced lately, including matrix factorizations [5, 6], social filtering [7], and network-based [8–10].

Bipartite network projection is among these approaches, which was initially introduced in physics but found applications in personal recommendation [8]. It relies on a resource distribution process in a bipartite network to provide a top- n recommendation. Particularly, it considers users and items as two types of network nodes, respectively, and treats user

taste as resource to be allocated in the bipartite network. User rating is scaled into a numeral (either 1 or 0), in comparison with the median of the predefined rating range, to indicate user taste. In another word, numeral 1 means the user likes a particular item, while numeral 0 means the user does not like or has not rated the item. Afterwards, two rounds of resource distribution are carried out, regardless of the user ratings. Eventually, the final resource that an unrated item gets indicates its possibility to be recommended to users. The method was shown to outperform the collaborative filtering methods. Albeit, we observe that there are at least two shortcomings that limit its further improvement. Firstly, the current user taste model does not differentiate “dislike” and “unrated” cases, both expressed by numeral 0, which potentially stops the algorithm to provide a more precise recommendation. Secondly, the resource distribution process does not leverage user ratings; that is, the transfer rates on different links are not proportional to the corresponding user ratings. Such allocation, and hence the recommendation, can be inaccurate when a user has a biased preference among the items. We rectify these issues in this work.

This paper first presents a *negative-aware* user taste model to encompass “like,” “dislike,” and “unrated” cases implied by user ratings, so that the user preference is fully reflected during recommendation. Instead of using the predefined rating range, we propose to use an adaptive user rating range as threshold for determining user taste. While the rationale behind is similar to the existing model, we compare a rating with the *average* rating of the user. Hence, a rating above this average is considered as a “like,” expressed by numeral 1; a rating below this average is regarded as a “dislike,” expressed by numeral -1; and unrated cases are noted by numeral 0. Based on this user taste model, we initialize the resource of each item according to the user taste and the proportion of the rating to the sum of all ratings from this user, which can appropriately reflect the dissimilarity of user interest. Similarly, when deriving the transfer rates of links in resource distribution, we design a *rating-integrated* method to allocate resource via a link proportional to the user ratings on the corresponding item. In this way, the final resource allocation is expected to more expressively reflect user preference. To scale the solution towards massive datasets, we further present a MapReduce-based implementation to *parallelize* the computation.

To summarize, we make the following contributions.

- (i) We devise a negative-aware user taste model to encompass “like,” “dislike,” and “unrated” cases implied by user ratings.
- (ii) We propose a rating-integrated method to allocate initial resource and determine transfer rates on network links according to user ratings.
- (iii) We extend the proposed algorithm into the MapReduce programming framework such that it is able to deal with massive volume of rating data.
- (iv) We implement all the proposed algorithms, and extensive experiments are conducted on real public datasets to evaluate their effectiveness and efficiency.

The rest of the paper is organized as follows. We discuss the related work in Section 2; Section 3 introduces the preliminaries and the baseline algorithm. Section 4 presents the new taste model and the method for initialization and distribution of the resource, followed by the implementation on MapReduce. Experimental results are listed and commented in Section 5, and we conclude the paper in Section 6.

2. Related Work

Personal recommendation has been a topic that draws much attention, and most of the existing work falls into the following five categories.

Content-based method extracts features for items, and a profile is created for a user with the features, which is then utilized to find out the most similar items to those collected. However, feature extraction is not easy, and the recommendation is restricted to items very similar to those who have been liked [1]. Semantic reasoning was used to overcome the shortage of vector space model [11]. This kind

of algorithms needs to describe item content and extract properties of rated items to build user profiles, so the cold start problem is serious.

Contrarily, *collaborative filtering* does not rely on item content. It utilizes rating behavior data to select the most similar neighbor and makes recommendation on the assumption that who behaved similarly in the past will behave similarly in future. The key steps of collaborative filtering are computing the similarity and generating the recommendation from the nearest neighbors. Representative work includes [2–4]. Due to the vast quantity of items and users but quite finite ratings, the sparsity problem turns into the bottleneck to raise the effectiveness.

Matrix factorization [5, 6] characterizes both items and users by vectors of factors inferred from rating patterns, and high correspondence between item and user factors leads to a recommendation. The single value decomposition [12] is the most commonly used method to reduce the matrix dimension. The complexity of the computation is so high that few practice applications adopt this kind of algorithms.

Social-based method recommends the user with what his neighbors like according to the link structure of social network. An InterestMap is built in [7] based on cooccurring keywords for recommendation. The social relation of users was incorporated into collaborative filtering to adjust the nearest neighbor selection strategy [13]. It is suitable for the data set which also describes the friend relations of users.

A line of more closely related work is the algorithms based on *bipartite networks* [8, 10], into which our proposed algorithm falls. High-order correlations were considered in [14]. There are also methods based on heat conduction [9, 15] and random walk [16] on bipartite networks. A hybrid approach containing multistep random walk, and *k*-means clustering was introduced to achieve smaller mean absolute error and root mean square error [17]. It is more precise than traditional recommendation algorithms, yet its accuracy and scalability could be further improved.

In order to improve the scalability of recommender systems, cloud-computing platforms are usually employed, for example, the Dynamo [18] of amazon.com and the Dryad [19] of Microsoft. Hadoop (Apache Hadoop, <http://hadoop.apache.org/>) is the most popular and widely-used open source platform implementing the MapReduce framework [20]. Hadoop is employed for content characterizing and profile matching for content based recommendation in [21]. A scalable online collaborative filtering algorithm for news recommendation is proposed in [22], which combines memory based and model based collaborative filtering algorithm. Recommendation algorithm with matrix factorization is implemented in [23] based on MapReduce, and the efficiency is proved to be higher. To solve the scalability problem, [24] implements the network-based inference algorithm on Hadoop; however, the accuracy remains to be improved.

3. Preliminaries

The baseline algorithm for personal recommendation based on bipartite network projection relies on a bipartite network,

FIGURE 1: Example of resource distribution in baseline algorithm.

consisting of two types of nodes, *user* and *item* nodes, denoted by U and V , respectively. Let u_i denote the i th user in U , and v_j denote the j th item. u_i has a rating $r(u_i, v_j)$ on v_j . All the ratings from user u_i constitute a rating set $R(u_i)$ with cardinality of $|R(u_i)|$, and all the ratings on item v_j constitute a rating set $R(v_j)$ with cardinality of $|R(v_j)|$. *Links* exist between different types of nodes, but not within the same type of nodes. That is, a link always connects a user node and an item node with no exception. Each link models a rating behavior of a user on an item. Assume every rated item by a given user is assigned with certain quantity of resource, that is, initial resource allocation. The resource first flows from item nodes to user nodes, then back to item nodes along the links. Hence, every item gets a final resource allocation after the *two-round resource distribution*. The top n items with bigger resource would be recommended to the current user then the recommendation will be executed for the next user.

Consider the initial resource as recommendation power. The intuition behind is that if an item gets more resource after distribution, it is more highly to be liked based on the user's preference. Thus, the resource distribution process is of importance. Currently, the taste of a user is determined based on a threshold equal to the median of the predefined rating range. Specifically, a user likes an item, expressed by numeral 1, if she rates the item above the threshold; otherwise, the case is concluded into the unrated category, expressed by numeral 0. For example, the threshold is 3, if the predefined rating range is $[1, 5]$; a user rating 4 on an item implies that she likes the item, since the rating is above the threshold; a user rating 2 on an item is diminished to unrated case, not affecting the recommendation thereafter.

After determining user tastes, the resource allocation process in the bipartite network is carried out. Two rounds of resource distribution are conducted. In the first round, resources are transferred from item nodes to user nodes, and all user nodes having links to a particular item share their resource equally. In the second round, resources are transferred back to item nodes, and all item nodes having links to a user share their resource equally. The final resource allocated to each item indicates its probability of being recommended to the given user. For a top- n recommendation, a list of n unrated items with largest resource allocation is created for the given user. We refer to this as "the baseline algorithm" in the rest of the paper when context is clear.

Example 1. Figure 1 shows an example of recommendation for user u_2 by the baseline algorithm. Assume that the constructed bipartite network is in Figure 1(a), with item nodes in the top and user nodes in the bottom. The ratings of user u_2 to items v_1 to v_5 are 5, 0, 2, 0, and 4, respectively, and the median is 3. So the initial resource configuration is 1, 0, 0, 0, and 1, respectively. The initial resource and raw rating value are tagged above the item nodes. In the first round as in Figure 1(b), resources are transferred from item nodes to users nodes; for example, user u_2 gets $1/3$ from v_1 and $1/3$ from v_5 , $2/3$ in total. Then, in the second round as in Figure 1(c), resources are transferred back to item nodes; for example, item v_2 receives $1/6$ from users u_1 and $1/12$ from user u_3 . So the final resource allocation for item v_2 is $1/4$ in total. Similarly, item v_4 gets $11/36$ in total. So item v_4 will be recommended to user u_2 for its allocated resource is the biggest among the unrated items of user u_2 .

4. Algorithms

This section first improves the accuracy of the baseline algorithm by introducing a negative-aware user taste model, along with the rating integrated method for initializing and distributing the resource. Afterward, the scalability is improved by implementing on the MapReduce.

4.1. User Taste Model. Let us first take a motivating example.

Example 2. Consider a predefined rating range $[1, 5]$ and a user rating 1 on an item. In the baseline algorithm, we take the median of the rating range, that is, 3 as the threshold. As 1 is below the threshold, the user may not like the item. Subsequently, we use a numeral 0 in the initial resource allocation for this item. Recall that a user taste to an item is also expressed by 0 if she has not rated the item.

The current user taste model does not distinguish the aforementioned cases, which hinders the algorithm from reaching a higher recommendation precision. Furthermore, it is intuitive that a rating close to the bottom of the rating range implies a low satisfaction or preference towards an item, that is, negative attitude towards the item. We argue that the rating reflects the user's taste only if it is compared with her own rating range, rather than the predefined rating

range. As a consequence, we distinguish the three cases—“like,” “dislike,” and “unrated”—in the new *adaptive* user taste model.

Firstly, we adopt the user’s rating range as the reference, instead of the predefined rating range. That is, we compare a user rating with respect to the user’s own rating range; for example, given a predefined rating range $[1, 5]$, a user always rates within $[2, 4]$ and then the latter is the user’s own rating range. This is intuitive, as users may have different rating habit. Some users are harsh when rating and, hence, always give ratings across the whole predefined rating range, while some are soft towards the items and, hence, usually give ratings within a small subdomain of the predefined rating range.

Subsequently, it is further observed that, as to an item, even if her rating is above some others’, this may indicate that she dislike the item, since she has a narrow ranting range. Therefore, instead of using median of the predefined rating range, we propose to use the average of a user’s ratings as the threshold to determine the user’s taste towards an item. That is, we take

$$\bar{r}(u_i) = \frac{\sum_{v_j \in V} r(u_i, v_j)}{|R(u_i)|} \quad (1)$$

as the threshold for determining user taste, where $R(u_i)$ is the set of ratings from u_i . Consequently, we have the taste model regarding user u_i and item v_j :

$$T(u_i, v_j) = \begin{cases} 1, & r(u_i, v_j) \geq \bar{r}(u_i) \\ -1, & \text{otherwise.} \end{cases} \quad (2)$$

The taste model is an indicator function such that 1 means the user likes the item, while -1 means the user dislikes the item. Note that we follow the convention to denote as 0 the user’s taste for an unrated item.

Example 3. Consider a predefined rating range of $[1, 5]$, and a user rates five items as 2, 3, 3, 4, and 5, respectively. Thus, according to the new model, we first take the average $(2 + 3 + 3 + 4 + 5)/5 = 3.4$ as the threshold. Hence, rating 3 is considered as a dislike, since it is close to the bottom of the rating range according to the user’s rating habit. On the other hand, rating 4 implies a like, since it is above the threshold.

One may argue that there are subtle cases that a user always rates high on items, where our model would consider those ratings below her average rating as “dislike.” Actually, to the best of our knowledge, there is no existing rating system in-use that allows negative ratings, and hence, the “dislike” rating behavior is difficult to capture. We argue that the proposed user taste model is able to capture more information implied by user rating behaviors and demonstrate better performance (cf. Section 5.1).

Our user taste model is adaptive in the sense that it adjusts the threshold according to different users so that the user’s taste can be well reflected. Introducing the taste of dislike can make full use of user taste information. Moreover, this distinguishes the dislike and unrated cases that are

treated identically in the baseline, and hence the recommendation performance is expected to be improved due to these improvements. We will see shortly how this affects the resource distribution process.

4.2. Initial Resource Allocation. A higher rating implies a stronger recommendation from a user towards an item. To reflect this information in the initial resource allocation, we weight the initial resource for every item with a coefficient. In particular, we multiple the taste numeral with the ratio of the user’s rating to current item to the average of the user’s ratings for all items. For a given user u_i and an item v_j , the initial resource allocated to v_j is

$$R_{\text{ini}}(v_j) = \hat{r}(u_i, v_j) \times T(u_i, v_j), \quad (3)$$

where $T(u_i, v_j)$ is as the taste model formulated in Section 4.1, $\hat{r}(u_i, v_j)$ denotes the weight we put on the user taste to generate the initial resource, which is

$$\hat{r}(u_i, v_j) = \begin{cases} \frac{r(u_i, v_j)}{\bar{r}(u_i)} & \text{if } (r(u_i, v_j) > \bar{r}(u_i)) \\ \frac{tr - r(u_i, v_j)}{\bar{r}(u_i)} & \text{otherwise,} \end{cases} \quad (4)$$

where tr is the top of the user rating range, and $tr - r(u_i, v_j)$ is to make sure that a smaller rating will get a smaller initial resource under the negative taste, when the rating is less than the average of user ratings.

This initial resource allocation tries to emphasize the distinction of user taste to different items that the user likes. Thus, the initial resources become more distinguishable and accordant with user taste.

Example 4. In the baseline algorithm, the initial resource is the same as the user taste for items. Since the ratings of the user to the items with same user taste are different, we have good reasons to doubt the way of initializing resource by simply equating the initial resource with the user taste for them. In Figure 2(a), the numbers next to the links are the ratings. To use the aforementioned model to allocate initial resource, we first determine the user taste for u_2 by formula (2), as 1, 0, -1 , 0, and 1, respectively, for all items. Then, we allocate initial resource by weighting $5/3.7$, $2/3.7$, and $4/3.7$ for the nonzero user taste, respectively, as formulae (3) and (4) represented.

4.3. Resource Distribution. Resource distribution based degree of nodes ignores the difference of links, so it cannot reflect the distinctness in the extent of interest from different users to the same item or that from the same user to different items. During resource transferring from item nodes to user nodes and then back to items, we make it aware of the transfer rates of all links, which is the ratio of user rating to the sum of ratings from all users who have rated the given item in the first round, or the ratio of user rating for an item to the sum of

FIGURE 2: Example of resource distribution in proposed algorithm.

ratings by the user to all items in the second round. This can improve the accuracy of resource distribution by making the distribution process more specific. In particular, in the first round of transfer from item nodes to user nodes, the ratings are normalized with each user's average rating to avoid the influence of rating bias from different users. Thus, for each user node u_i , the resource transferred from item v_j to this node is

$$R(u_i, v_j) = \frac{\hat{r}(u_i, v_j)}{\sum_{u_k \in U} \hat{r}(u_k, v_j)} \times R_{\text{ini}}(v_j). \quad (5)$$

Adding the contributions from all item nodes connecting to u_i ,

$$R(u_i) = \sum_{v_j \in V} R(u_i, v_j). \quad (6)$$

Example 5. Recall Example 4. In Figure 2(b), the numbers around the edges are the transfer rates from item nodes to user nodes. The initial resource is distributed from item nodes to user nodes in the first round by formula (5). Particularly, the initial resource of item v_1 is distributed to users u_1 , u_2 , and u_4 according to the transfer rate of each link. The resource allocation of u_1 , u_2 , and u_4 from item v_1 is 0.48, 0.52, and 0.4, respectively. After the other items finish resource distribution, the resource allocation of every user can be obtained. For example, 0.48 in total for user u_1 with resource 0.48 transferred from item v_1 and 0 from v_2 , respectively.

Similarly, in the second round, resource transfers from user nodes back to item nodes according to transfer rate from user nodes to item nodes. Nonetheless, no normalization needs to be done in the round, as these ratings are all given by the same user. Hence, we can derive the final resource allocation to an item v_j by

$$R_{\text{fin}}(v_j) = \sum_{u_i} R_{\text{fin}}(u_i, v_j), \quad (7)$$

where $R_{\text{fin}}(u_i, v_j)$ denotes the final resource from user u_i to item v_j , which is

$$R_{\text{fin}}(u_i, v_j) = \frac{r(u_i, v_j)}{\sum_{v_j} r(u_i, v_j)} \times R(u_i). \quad (8)$$

Example 6. Recall Example 4. In Figure 2(c), the numbers around the edges are the transfer rates from user nodes to item nodes. The resource is distributed from user nodes to item nodes in the second round by formula (8). After every user distributes the resource to the items with which there is a link, the final resource for every item can be summed up. For example, the final resource is 0.19 for item v_2 and 0.05 for item v_4 . So, in this bipartite network sample, the recommendation result for user u_2 in our proposed algorithm is item v_2 , and this is different from the result in the baseline algorithm which is item v_4 with a resource value of 0.3.

The correctness of the method remains, since the different weighting of initial resource allocation and distribution only affects the volume of transferred resource but not the distribution process. Furthermore, this rating-aware resource distribution allocates the user taste resource discriminatively. Therefore, the final resource allocation is expected to provide a more accurate suggestion on the top- n recommendation. We verify the effectiveness of the proposed algorithm in Section 5.

4.4. Implementation on MapReduce. Along with the surge of data collected during the use, there is a rapid proliferation of data available that can be utilized for recommendation, which is considered as a leap on top of the big data. Seeing the opportunity, we are also aware of the scalability issue involved in the massive computation. We address the challenge in the sequel by incorporating the MapReduce programming paradigm.

MapReduce distributes the computation by splitting a given task into several small subtasks and processes them simultaneously on different computers. In MapReduce, the whole calculation of a job is divided into map and reduce phase. Mapper decomposes the process into several parts and assigns them to available nodes. The input of map phase is a set of key/value pairs. The form of the output is also key/value pairs but both the key and the value could be different from the input ones. The intermediate values with the same key would be grouped together by MapReduce library. The reducers sum up the values corresponding to the same key or do some other operations. The output of the reduce phase is also key/value pairs.

Recall the algorithm introduced in previous subsections. It measures how the initial resource would be distributed

to the user nodes from the item nodes, and how much is transferred back to item nodes from the user nodes. In another words, it tries to find a value to measure the resource allocation from one item rated by the user to another one on which the user has not expressed his/her attitude. It is easy to put all the formulae that describe the process of resource transfer into one, which calculates the resource allocated from an item to another, when the initial resource configuration is still the same. We use w_{ij} to represent the resource transferred from item v_j to item v_i , with item v_j rated by the target user while item v_i did not; that is,

$$W_{ij} = \sum_{l=1}^m \left(\frac{\hat{r}(u_l, v_j)}{\sum_{k=1}^m \hat{r}(u_k, v_j)} \cdot \frac{r(u_l, v_i)}{\sum_{t=1}^n r(u_l, v_t)} \cdot (E_{li} \cdot E_{lj}) \right), \quad (9)$$

where the first fraction represents the transfer rate from item v_j to user u_l with the user average rating to eliminate effect of the user rating bias, the second denotes the transfer rate from user u_l to the item v_i , and m and n are the total number of users and items, respectively. E_{li} and E_{lj} denote a binary function for user rating; that is,

$$E_{kt} = \begin{cases} 1, & \text{if } (r(u_k, v_t) = 1, 2, 3, 4, 5) \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

In essence, it is a classification indicator judging if there is a rating behavior between the corresponding user and item. Consider

$$R_{\text{fin}} = W \times R_{\text{ini}}, \quad (11)$$

where W denotes the resource transfer matrix, R_{fin} is the final resource of items, and R_{ini} is the initial resource of the items for specific user as defined in previous subsections. The items to be recommended for the target user are selected according to the final resource in the descending order.

In the proposed implementation, we use four jobs for our purpose.

JOB 1. The first job prepares a list for each user which contains all the rated items as well as the corresponding ratings; it also calculates the sum and the average of each user's ratings, which are tagged as sum_u and avg_u for short. The input is the rating record containing user id, item id and rating, which would be treated as value of the key/value pair. We use uid , iid_i , and rat_{ui} to represent the user id, the identity of item v_i , and the rating from user u to item v_i . The default key is the offset of the row containing the rating record. These raw records are distributed to all mappers, and every mapper reorganizes the records in the form of user id and the pair of item id and rating. The key becomes the user identity after map phase while the value turns into item id coupled with the rating. Afterward, the intermediate results are gathered according to the key. Reducers obtain all item-rating pairs of the same user, and compute the sum along with the average of these ratings. Here the key is still the user id, but the value turns into the sum and the average of the user. The functions for Map and Reduce are as follows:

Map: $\langle \text{offset}, \text{record} \rangle \rightarrow \langle \text{uid}, (\text{iid}_i, \text{rat}_{ui}) \rangle$;

Reduce: $\langle \text{uid}, \text{list}((\text{iid}_i, \text{rat}_{ui})) \rangle \rightarrow \langle \text{uid}, (\text{sum}_u, \text{avg}_u) \rangle$.

JOB 2. The second job reorganizes the records according to the item identity. For every item, the ratio of the rating by each user to the average rating of this user is computed, which is named as *unbiased user rating* (uur_i , for short), and then the sum of the unbiased ratings from all users is calculated. In the map phase, the records will be sorted by item id and unbiased user rating is calculated. The key of output is item id, and the value is the user id paired by the unbiased user ratings. Reducers gather the intermediate results and output the unbiased rating sum for item v_i (suur_i , for short). The item id along with the sum of unbiased rating for this item is the key, while value is a string remarked as str_i , which contains all the users that rated this item and the unbiased rating from this user. All the results would be stored for the following jobs. The Map and Reduce functions are as follows:

Map: $\langle \text{offset}, \text{record} \rangle \rightarrow \langle \text{iid}_i, (\text{uid}, \text{uur}_{ui}) \rangle$;

Reduce: $\langle \text{iid}_i, \text{list}((\text{uid}, \text{uur}_{ui})) \rangle \rightarrow \langle (\text{iid}_i, \text{suur}_i), \text{str}_i \rangle$.

JOB 3. The third job uses the outputs of the first two jobs to calculate the value of all the cells in the resource allocation matrix. Noticing that the value of resource allocated from item v_j to item v_i is calculated by summing up the resource quantity from every user, the given items v_j passing through to get to item v_i , we assign the mappers to compute the quantity for every user with different combination of item v_i and item v_j . The key turns into the identities of the two items; the value is the resource quantity of each user, res_m for short. The reducers sum these intermediate values to get the final allocation quantity of resource between two items. The key of the reduce phase is the pair of item identities, and the value is the resource allocation from item v_j to item v_i . The Map and Reduce functions are as follows:

Map: $\langle (\text{iid}_i, \text{iid}_j, \text{uid}), (\text{sum}_u, \text{uur}_{uj}, \text{suur}_j, \text{rat}_{ui}) \rangle \rightarrow \langle (\text{iid}_i, \text{iid}_j), \text{res}_u \rangle$;

Reduce: $\langle (\text{iid}_i, \text{iid}_j), \text{list}(\text{res}_u) \rangle \rightarrow \langle (\text{iid}_i, \text{iid}_j), w_{ij} \rangle$.

JOB 4. The last job firstly gets the initial resource for every user using the output of the first job and then calculates the final resource (FR, for short) allocated to the items by multiplying resource allocation matrix with the initial resource, noticing that the computation for the items which have been rated by this current user will be skipped. Besides, the items will be ordered according to the final resource descending and the top n items of the rank will be recommended to the user. In the map phase, the key of output is the identities of the user and the destination item, the value is the temp resource (TR_j , for short) from item v_j . The key of output in the reduce phase is the user identity and the item identity, the value is the final resource of this item for the user. The Map and Reduce functions are as follows:

Map: $\langle (uid, iid_i, iid_j), (rat_{uj}, avg_u, w_{ij}) \rangle \rightarrow \langle (uid, iid_i), TR_j \rangle;$
 Reduce: $\langle (uid, iid_i), list(TR_j) \rangle \rightarrow \langle (uid, iid_i), FR \rangle.$

It is easy to verify that the four MapReduce jobs correctly compute the results, inheriting from the correctness of the implementation on single machine. We evaluate the effect of the MapReduce-based implementation in Section 5.4.

5. Experiments

This section presents the experimental results and analyses.

5.1. Experiments Setup. We run two series of experiments on distinct data sets for different purposes.

To evaluate the accuracy improvement, we conduct experiments on a real dataset *MovieLens* (<http://www.grouplens.org/>), which is one of the most famous datasets for evaluating personal recommendation. It consists of 943 users, 1,682 movies, and about 100,000 rating records. Users rate movies according to their interest with discrete numerals from 1 to 5. The data is cleaned by removing users who rated less than 50 movies. The sparsity of the dataset is about 94.6%, meaning that the ratings by users to movies are rather insufficient. Further, we select the records if the rating exceeding the average rating of the user as primary data set, in which there is about 54,800 ratings in number. Then, we pick out ratings from the primary data set randomly by a ratio of 20% to construct the test set with a scale of 10,960 records, which would be done five times for cross-validation. The rest ratings in the primary data set and the others records which are not included in the primary data set are the members of our training set with a scale of 83,640 rating records.

For measuring the accuracy of top- n recommendation, *hit ratio* and *average rank score* are two popular evaluation metrics while comparing different algorithms.

- (i) Hit ratio counts the number that movies of the test set occur in the recommendation list and then uses the total length of recommendation list as the divisor to get the final ratio value; that is,

$$hr = \frac{\sum_{i \in M} \sum_{j \in N} \sum_{k \in T} H_i(j, k)}{M * N}, \quad (12)$$

where M denotes the number of users, N denotes the number of items recommended for each user, T denotes the number of items in test set for each user, and $H_i(j, k)$ is an indicator function,

$$H_i(j, k) = \begin{cases} 1, & rl(i, j) = tl(i, k) \\ 0, & \text{otherwise,} \end{cases} \quad (13)$$

where $rl(i, j)$ is the j th item for user i in the recommendation list and $tl(i, k)$ is the k th item of user i in the test set.

- (ii) Average rank score measures the position of each movie of the test set in the sequence of all unrated

movies, and smaller score value means better recommendation result. The average rank score is the average of these scores for each record in the test set which is shown as follows:

$$r_{ik} = \frac{\text{loc}(tl(i, k))}{Q'_i}, \quad (14)$$

where $\text{loc}(tl(i, k))$ is the location of item $tl(i, k)$ in the sequence for user i in the test set and Q'_i is the number of unrated items by user i in the sequence.

To validate the efficiency of the implementation on MapReduce, we use the *Netflix* data set which contains about 480,189 users, 17,770 items, and 100,480,507 rating records. We will compare the efficiency of the proposed algorithm implemented on single computer and the clusters within MapReduce on Hadoop platform. The Hadoop computer-cluster consists of 5 computers, with one name node and other four data nodes. The configurations for all computers are all same, INTEL Dual-core 2.73 GHZ CPU, 4 GB memory, 500 GB hard disk and UBUNTU operation system. We will change g , which is the number of computers that are employed for the parallel computing, from 3 to 5 step by step to observe the runtime trends.

The time of single computer cost is marked as T_1 . The time consumed by the cluster containing 3, 4, and 5 computers is denoted as T_3 , T_4 , and T_5 , respectively. We could expect that the computation complexity is $O((mn + n + mn^2 + n^3)/g)$, if we ignore the communication in computation between these computers. However, it is impossible to not take into account the communication time. In fact, the complexity for the communication is $O(mn^2 \cdot \log(g))$ sensitive to the number of both items and users. So the practical time complexity would be higher than expected.

To show the comparison results more clearly, we use the ratio of the time consumed by the single computer version to the one by the implementation on MapReduce, which is denoted as

$$Q_g = \frac{T_1}{T_g}, \quad (15)$$

where g can be 3, 4, or 5, which lies on the number of data nodes employed by the MapReduce-based implementation. The bigger the result is, the more efficient the Implementation on MapReduce is.

5.2. Comparing with the Baseline Algorithm. In this set of experiments, we measure the comparison with hit ratio and average rank score.

First, we evaluate the effects of Average-Rating Criterion and Taste with Dislike (labeled by “ARC” and “TD”, resp.) in the user taste model, Rating-Aware Initial Configuration and Rating-Aware Transfer (labeled by “RAIC” and “RAT”, resp.) in the resource allocation with hit ratio metric.

As Figure 3 shows, the user-specific average ratings criterion outperforms the median one when specifying the taste of users. Median method ignores the difference of rating habits

FIGURE 3: Effect of proposed techniques.

between different users while average rating method judges whether users like an item according to more particular and personal criterions.

When the tastes of users are specified as dislike together with like, higher hit ratio was achieved than using only like information as user tastes. The dislike information as negative resource can spread over the bipartite network to rectify the single taste resource allocation. A comprehensive taste considering both like and dislike of users got a more accurate recommendation result.

The initial configuration of resource based on comprehensive user taste model sets the values of items which had been rated by current user as 1 or -1 representing like or dislike. Considering the extent of a user likes or dislikes these items could still be very different, the ratings by users to items are utilized to weight the initial resource. This is the rating-aware initial configuration. We can tell from Figure 3 that rating-aware initial resource configuration got better result.

The distribution based on degree considered only the relationship of user and item nodes, but the weight for each link between item and user is not measured. The rating-aware transfer uses the ratio of the rating by a user to an item to the sum of ratings by the particular user to all items or all users to the given item as the importance of the edge between this user and the item. The rating-aware transfer weights degree-based transfer with the importance of links, so its recommendation accuracy is higher.

The proposed algorithm emphasizes the taste of dislike as much as the taste of like which are specified according to the average rating of each user; the resource allocation and distribution are adjusted to consider the rating ratios and the transfer rates of the edges. Compared with the uniform algorithm, which is also named network based inference (NBI), our algorithm performs obviously better with higher hit ratio.

Then we compare the NBI and our proposed algorithm with average ranking score metric. Avg-rating, Com-taste, RA-initial, and RA-transfer denote the average rating based

FIGURE 4: Comparison on average rank score.

coarse-graining, the comprehensive taste with both like and dislike, rating-aware initial configuration, and rating-aware transfer of resource, respectively. They are used to indicate the algorithm which adopted the improvements that they represent and the ones before them. $\langle r \rangle$ denotes the average rank score.

As Figure 4 shows, rank is the total scale of items in the test set which are used to validate the recommendation results while the sequence attribute is ignored temporarily and $\langle r \rangle$ is the rank score. We can see that the rank scores for the five curves increase exponentially while the length of rank grows, especially when the items number is around 6000. Averaged over records in the test set, the average rank scores for the five implementations are shown, respectively, in the legend. It is easy to find that all the improvements are effective, and the introduction of dislike as taste got the biggest improvement. Our proposed algorithm achieves better performance than NBI by 16.7% using average rank score metric, which is in accordance with the conclusion using hit ratio metric.

5.3. Comparing with Other Three Algorithms. We compare the proposed algorithm with an algorithm based on mass diffusion and two popular variants of bipartite network projection to further validate effectiveness with both hit ratio and average rank score metrics.

- (i) IMD [25] is an improvement of mass diffusion which takes into account the average degree of user nodes to weight the initial resource distribution;
- (ii) E-NBI [26] is a variant of uniform NBI which depresses the impact of high-degree with a negative exponential function to improve the accuracy;
- (iii) INBI [27] combines weighted bipartite network with a tunable parameter to depressing high-degree nodes for top- n recommendation.

In Figure 5, the left part demonstrates the comparison with hit ratio metric and while the recommend list varies from 10 to 50 the proposed algorithm achieves the best result all the time; the right part is the result of comparison with

FIGURE 5: Comparisons with other three algorithms.

FIGURE 6: Comparisons of implementations on single computer and Hadoop cluster.

average rank score; the proposed algorithm outperforms the other ones by 13.0%, 11.7%, and 3.9% on average, respectively.

5.4. Evaluating the Implementation on MapReduce. First, we make the number of items fixed at 17770 while the number

of users changing from 10000 to 100000 by a step length of 10000. Then, we fix the number of users at 480000 and change the number of items from 2000 to 16000 by an interval of 2000. All these samples are randomly chosen. The results for the two sets of experiments are shown as Figure 6.

With the increase in the number of users or items, the computation costs more time for both the implementation on single computer and the one on Hadoop cluster; however, the former one grows far more than the latter, so the ratio of time consuming increases fast. We can learn the result from Figure 6. Besides, when the number of data nodes increases, the efficiency becomes higher. All the curves nearly stop rising and converge at different sizes of the users or items because the finite data nodes restrict the further improvement of the time performance. To propose huge data set, more cluster nodes with MapReduce framework should be employed. However, our implementation on Hadoop is effective and the parallel computing in the MapReduce framework can improve the efficiency of our proposed algorithm.

6. Conclusion

In this paper, we have proposed a negative-aware and rating-integrated personal recommendation algorithm based on bipartite network projection. It takes the advantages of the information implied by user ratings to differentiate dislike cases of user tastes, and weight the resource distribution process. Better empirical results on real data are obtained regarding both hit ratio and average ranking score. To cope with large amount of data, we implement the proposed algorithm on MapReduce which is confirmed to be more efficient by our experiments. As future work, we plan to conduct more comprehensive comparison between the proposed algorithm and other state-of-the-art algorithms for recommendation. The other possibility is to devise more precise models for depicting user behaviors.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (61302144 and 61303062). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

References

- [1] M. Balabanović and Y. Shoham, "Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.
- [2] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "Applying collaborative filtering to usenet news," *Communications of the ACM*, vol. 40, no. 3, pp. 77–87, 1997.
- [3] M. Khoshneshin and W. N. Street, "Collaborative filtering via Euclidean embedding," in *Proceedings of the 4th ACM Recommender Systems Conference (RecSys '10)*, pp. 87–94, Barcelona, Spain, September 2010.
- [4] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (ACM KDD '09)*, Paris, France, July 2009.
- [5] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [6] Y. Shi, M. Larson, and A. Hanjalic, "Mining contextual movie similarity with matrix factorization for context-aware recommendation," *ACM Transactions on Intelligent Systems and Technology*, vol. 4, no. 1, article 16, 2013.
- [7] H. Liu and P. Maes, "InterestMap: harvesting social network profiles for recommendations," in *Proceedings of the International Conference on Intelligent User Interfaces (IUI '05)*, San Diego, Calif, USA, January 2005.
- [8] T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang, "Bipartite network projection and personal recommendation," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, no. 4, Article ID 046115, 2007.
- [9] Y.-C. Zhang, M. Blattner, and Y.-K. Yu, "Heat conduction process on community networks as a recommendation model," *Physical Review Letters*, vol. 99, Article ID 154301, 2007.
- [10] Y. Zhang, M. Medo, J. Ren, T. Zhou, T. Li, and F. Yang, "Recommendation model based on opinion diffusion," *Europhysics Letters*, vol. 80, no. 6, p. 68003, 2007.
- [11] Y. Blanco-Fernández, J. J. Pazos-Arias, A. Gil-Solla, M. Ramos-Cabrer, and M. López-Nores, "Providing entertainment by content-based filtering and semantic reasoning in intelligent recommender systems," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 727–735, 2008.
- [12] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental SVD-based algorithms for highly scalable recommender systems," in *Proceedings of the 5th International Conference on Computer and Information Technology (ICCIT '02)*, April 2002.
- [13] F. Liu and H. J. Lee, "Use of social network information to enhance collaborative filtering performance," *Expert Systems with Applications*, vol. 37, no. 7, pp. 4772–4778, 2010.
- [14] J.-G. Liu, T. Zhou, H.-A. Che, B.-H. Wang, and Y.-C. Zhang, "Effects of high-order correlations on personalized recommendations for bipartite networks," *Physica A*, vol. 389, no. 4, pp. 881–886, 2010.
- [15] J.-G. Liu, T. Zhou, and Q. Guo, "Information filtering via biased heat conduction," *Physical Review E*, vol. 84, Article ID 037101, 2011.
- [16] J. Quan and Y. Fu, "A novel collaborative filtering algorithm based on bipartite network projection," *International Journal of Digital Content Technology and its Applications*, vol. 6, no. 1, pp. 391–397, 2012.
- [17] S. Sawant, "Collaborative filtering using weighted bipartite graph projection: a recommendation system for yelp," in *Proceedings of the CS224W: Social and Information Network Analysis Conference*, December 2013.
- [18] G. DeCandia, D. Hastorun, M. Jampani et al., "Dynamo: amazon's highly available key-value store," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pp. 205–220, Stevenson, Wash, USA, October 2007.
- [19] M. Isard, M. Budi, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*, pp. 59–72, Lisboa, Portugal, March 2007.

- [20] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [21] T. de Pessemer, K. Vanhecke, S. Doooms, and L. Martens, "Content-based recommendation algorithms on the Hadoop mapreduce framework," in *Proceedings of the 7th International Conference on Web Information Systems and Technologies (WEBIST '11)*, pp. 237–240, May 2011.
- [22] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceeding of the 16th International World Wide Web Conference (WWW '07)*, pp. 271–280, Alberta, Canada, May 2007.
- [23] Y. Zhang and J.-J. Cheng, "Study on recommendation algorithm with matrix factorization method based on MapReduce," *Computer Science*, vol. 40, no. 1, pp. 19–36, 2003.
- [24] Z. Tang, Q. Wang, and S. Cai, "Network-based inference algorithm on hadoop," in *Proceedings of the 20th International Symposium (ISMIS '12)*, Macau, China, December 2012.
- [25] J.-G. Liu, T. Zhou, B.-H. Wang, Y.-C. Zhang, and Q. Guo, "Effects of user's tastes on personalized recommendation," *International Journal of Modern Physics C*, vol. 20, no. 12, pp. 1925–1932, 2009.
- [26] T. Zhou, L.-L. Jiang, R.-Q. Su, and Y.-C. Zhang, "Effect of initial configuration on network-based recommendation," *Europhysics Letters*, vol. 81, article 58004, 2008.
- [27] J. Xia, J. Wu, C. Xie, and J. Tu, "INBI: an improved network-based inference recommendation algorithm," in *Proceedings of the IEEE 7th International Conference on Networking, Architecture and Storage (NAS '12)*, pp. 99–103, June 2012.