Neurocomputing 000 (2017) 1–10

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

On identifying *k*-nearest neighbors in neighborhood models for efficient and effective collaborative filtering

Dong-Kyu Chae[a], Sang-Chul Lee[b], Si-Yong Lee[b], Sang-Wook Kim[a,*]

[a] Department of Computer and Software, Hanyang University, 17 Haengdang-dong, Seongdong-gu, Seoul 133–791, South Korea
[b] Department of Electronics and Computer Engineering, Hanyang University, 17 Haengdang-dong, Seongdong-gu, Seoul 133–791, South Korea

ARTICLE INFO

Article history:
Received 4 December 2016
Revised 14 June 2017
Accepted 16 June 2017
Available online xxx

Keywords:
Recommender system
Collaborative filtering
Efficiency

ABSTRACT

Neighborhood models (*NBMs*) are the methods widely used for collaborative filtering in recommender systems. Given a target user and a target item, *NBMs* find *k* most similar users or items (i.e., *k*-nearest neighbors) and make a prediction of a target user on an item based on the rating patterns of those neighbors on the item. In *NBMs*, however, we have a difficulty in satisfying both the performance and accuracy together. In order to pursue an accurate recommendation, *NBMs* may find the *k*-nearest neighbors at every recommendation request to exploit the latest ratings, which requires a huge amount of computation time. Alternatively, *NBMs* may search for the *k*-nearest neighbors offline, which consequently results in inaccurate recommendation as time goes by, or even may not able to deal with new users or new items, because they cannot exploit the latest ratings generated after the *k*-nearest neighbors are determined. In this paper, we propose a novel approach that finds the *k*-nearest neighbors efficiently by identifying only those users and items necessary in computing the similarity. The proposed approach enables *NBMs* not to require any offline similarity computations but to exploit the latest ratings, thereby resolving speed-accuracy tradeoff successfully. We demonstrate the effectiveness of the proposed approach through extensive experiments.

1. Introduction

The recommender systems have been aggressively incorporated as a core part to most online shops. They help users who are going through numerous items (e.g., movies or music) presented in online shops by quickly capturing each user's preference on items and suggesting a set of personalized items that s/he is likely to prefer [1,2]. *Collaborative filtering* (CF), which exploits the ratings on items given by other users whose preferences are similar to a target user, is one of the most successful and widely-used methods to learn the connection between users and items [3].

CF can be categorized into two techniques: *neighborhood models* (*NBMs*) and *latent factor models* (*LFMs*) [4]. *NBMs* find *k* most similar users (or items), i.e., *nearest neighbors*, to a target user (or an item) and make a prediction of a target user on an item based on the rating patterns of those neighbors on the item. *NBMs* are sub-categorized into *user-based models* (*UBMs*) [5,6] and *item-based models* (*IBMs*) [7–9]. On the other hand, *LFMs* learn hidden patterns from observed ratings by using matrix factorization techniques [10]. In general, *LFMs* have shown better accuracy as well as faster execution time than *NBMs* so far [10–12]. However, if the user-item rating matrix becomes extremely sparse, the accuracy of *LFMs* may decrease significantly, even being lower than that of *NBMs*. This is because learning latent semantic factors in rating patterns becomes more and more difficult as more user-item interactions are missing, which is revealed not only by our experimental results, but also by many previous researches [13–15]. Indeed, most user-item matrices in real-world online shops, especially those in the starting phase are extremely sparse in general [16–19]. In this sense, rather than *LFMs*, we focus our attention on improving *NBMs* in terms of their execution time, so as to make them provide effective and efficient recommendation in such situations.

The main drawback in *NBMs* is their huge computation time on searching for *k*-nearest neighbors from large numbers of users and items. Therefore, most work on *NBMs* identifies the *k*-nearest neighbors of every user (or item) *offline* and aggregates the ratings of the *k*-nearest neighbors on a target item in the *online* stage [20]. This kind of a solution would fail to keep up with the dynamic nature of online data [21]. Therefore, it suffers from accuracy decreased over time since it excludes the latest ratings once

* Corresponding author.
E-mail addresses: kyu899@agape.hanyang.ac.kr (D.-K. Chae), korly@agape.hanyang.ac.kr (S.-C. Lee), charismasy@agape.hanyang.ac.kr (S.-Y. Lee), wook@hanyang.ac.kr (S.-W. Kim).

http://dx.doi.org/10.1016/j.neucom.2017.06.081
0925-2312/© 2017 Elsevier B.V. All rights reserved.

the *k*-nearest neighbors are determined [20]. The latest ratings are very useful to learn user's preferences, especially when the rating matrix is so sparse that it lacks clues to learn preferences of each user. Moreover, such offline method may be not able to even deal with new user or new items. Suppose a recommendation model such as *NBM* or *LFM* is built; after a while a new user joins the system and evaluates several items (resp., a new item is registered and then evaluated by several users). In this situation, such offline recommendation systems may not work for the new user or the new item since they could not exploit any ratings generated after the model is built.

Motivated by the above points, in this paper, we propose a novel approach for *NBM*s that can provide fast recommendation with exploiting the latest ratings (i.e., online recommendation). The main idea of our approach is to efficiently identify only those users and items necessary in finding the *k*-nearest neighbors of a target user (or item). *UBM*s for example, we first filter out those users who have not rated any items that the target user has rate. Then, we also filter out those items that have not been rated by the target user. Only the ratings given by the remaining users and items are necessary in computing the similarities for finding the target user's *k*-nearest neighbors. The similar filtering scheme also works for *IBM*s.

To realize this idea, rather than the conventional user-item rating matrix, we propose novel data structures to support the efficient retrieval of only those users and items necessary in similarity computations. The data structures indeed help *NBM*s identify the *k*-nearest neighbors efficiently, making them much more feasible in online recommendation with exploiting the latest ratings. In addition, our data structures allow updates as soon as necessary, e.g., a user rates an item, a new user comes to the shop, or a new item is added to the shop. Moreover, *NBM*s with our data structures provide the recommendation result exactly the same as that from the conventional *NBM*s while improving their efficiency very significantly. Our extensive experiments with various real-world datasets demonstrate that our approach performs much faster than conventional *NBM*s as well as *LFM*s, achieving reasonable tradeoff between accuracy and efficiency in recommendation. We also verify the superiority of our approach from the viewpoints of whether it efficiently performs in recommending under sparser settings and whether it is well-incorporated with data imputation approaches. Regarding that *NBM*s provide more accurate recommendation than *LFM*s when the user-item matrix is extremely sparse, we can say that *NBM*s equipped with our proposed data structure become more useful in various real-world applications.

In summary, our main contributions are itemized as follows:

- We identify four *simple yet important lemmas*, which are foundational to our efficient recommendation approach that quickly filters out unnecessary users and items.
- We propose two types of data structures that effectively realize our filtering approach under *UBM* and *IBM* framework.
- We perform extensive experiments with various real-world datasets, and successfully confirm that our proposed approach is feasible for online recommendation with no loss of recommendation accuracy.

The rest of the paper is organized as follows. In Section 2, we briefly review the algorithms of *NBM*s and the associated similarity measures. In Section 3, we show the results of our preliminary experiments to emphasize the necessity of our approach. In Section 4, we present the details of our approach. In Section 5, we report the results of our extensive experiments. In Section 6, we briefly review several research related to our work. In Section 7, we finally summarize and conclude the paper.

2. Neighborhood models

In this section, we briefly review the methods in *NBM*s and the associated similarity measures. *NBM*s base their prediction on the *k*-nearest neighbors of a target user [22]. Formally, let $r_{u,i}$ be a rating by a user u for item i. Suppose that $r_{u,i}$ is not observed yet. In order to predict $r_{u,i}$, *UBM* looks for k *users* (i.e., *k-nearest neighbors*), denoted as S_u^k, who are most similar to u. Then, for each target item i, it takes a weighted combination of the residual ratings given by the neighbors $v \in S_u^k$, as follows:

$$\hat{r}_{u,i} = b_{u,i} + \sum_{v \in S_u^k} (r_{v,i} - b_{v,i}) w_{u,v} \tag{1}$$

In Eq. (1), $w_{u,v}$ is the similarity weight between two users u and v. $b_{u,i}$ (or $b_{v,i}$) is a biased rating value for user u (or v) to item i, computed as $b_{u,i} = \mu + b_u + b_i$ where μ indicates the average of overall ratings, b_u and b_i does the observed deviations of user u and item i, respectively, from the average [4,23]. The biased rating captures systematic tendencies of each user and item such that some users to give higher ratings than others, and some items to receive higher ratings than other items [23]. We can estimate b_u and b_i by solving the following optimization problem:

$$\min_{b_*} \sum_{(u,i)\in R} (r_{u,i} - \mu - b_u - b_i) + \lambda_1 \Big(\sum_u b_u^2 + \sum_i b_i^2\Big), \tag{2}$$

where the second term $\lambda_1(\sum_u b_u^2 + \sum_i b_i^2)$ indicates the regularizing term that controls the parameter sizes to avoid overfitting; λ_1 is the regularization constant ($\lambda_1 = 0.015$); R stands for the set of existing ratings. The optimization problem is solved by the *alternating least squares* (*ALS*) algorithm.

Second, *IBM* searches for k *items*, denoted as $I_{u,i}^k$, which are most similar to i and rated by u, and then takes a weighted combination of u's ratings on items in $I_{u,i}^k$, as follows[1]:

$$\hat{r}_{u,i} = b_{u,i} + \sum_{j \in I_{u,i}^k} (r_{u,j} - b_{u,j}) w_{i,j} \tag{3}$$

Note that $I_{u,i}^k$ depends on both the target user and the target item while S_u^k only depends on the target user. In other words, given a target user, *UBM* finds only one user set S_u^k while *IBM* needs to find m item sets $I_{u,i}^k$ where m indicates the possible number of target items.

For finding k-nearest neighbors, i.e., S_u^k or $I_{u,i}^k$, *NBM*s need to compute the similarities of all users (or all items) to u (or i). *Pearson correlation coefficient* and *adjusted-cosine similarity* are the most widely used measures to compute similarity between users and items, respectively. First, the *Pearson correlation coefficient* measures the extent to which two variables linearly relate with each other [3,24]. For *UBM*, the *Pearson correlation coefficient* between users u and v, denoted by $w_{u,v}^{pcc}$, is defined as

$$w_{u,v}^{pcc} = \frac{\sum_{i\in I_{u,v}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i\in I_{u,v}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i\in I_{u,v}} (r_{v,i} - \bar{r}_v)^2}}, \tag{4}$$

where $I_{u,v}$ indicates a set of items that both users u and v have rated; $\bar{r}_u$ does the average of ratings by the user u on the items in $I_{u,v}$.

Second, for *IBM*, the *cosine similarity* regards each item as a vector of ratings given by users and computes the cosine score of the angle formed by two rating vectors [3,25]. The *adjusted-cosine*

[1] On the both two predictions, we do not normalize $r_{u,i}$ in order to fall it into a specific range, e.g., from 1 to 5. This is because our goal is not to predict the exact rating values but just to rank items correctly so as to provide top-*N* items to be recommended.

Table 1
Statistics of execution times for each method (in seconds).

Data		IBM				UBM			
		avg	std	min	max	avg	std	min	max
ML-100K	KNN	10.70	0.53	10.92	12.02	7.74	0.04	7.66	7.97
	WS	0.05	0.01	0.05	0.06	0.05	0.001	0.05	0.05
	Total	**10.75**	0.54	10.97	12.08	**7.79**	0.05	7.71	8.02
ML-1M	KNN	89.87	13.67	79.32	102.53	27.30	0.02	27.30	27.30
	WS	0.12	0.07	0.08	0.22	0.19	0.10	0.12	0.31
	Total	**89.99**	13.74	79.40	102.75	**27.50**	0.12	27.40	27.60
ML-10M	KNN	6822.60	7.26	6811.14	6828.32	807.30	0.88	806.30	807.80
	WS	0.49	0.02	0.48	0.53	0.36	0.01	0.35	0.37
	Total	**6823.09**	7.28	6811.62	6828.85	**807.60**	0.89	806.60	808.10
Ciao	KNN	8.43	0.82	8.03	9.54	6.94	0.47	6.11	6.89
	WS	0.006	0.001	0.005	0.008	0.006	0.002	0.004	0.007
	Total	**8.436**	0.824	8.035	9.548	**6.946**	0.472	6.114	6.897

Data		IBM_pre				UBM_pre				SVD			
		avg	std	min	max	avg	std	min	max	avg	std	min	max
ML-100K	KNN	55.30				139.10							
	WS	**0.06**	0.01	0.05	0.06	**0.05**	0.001	0.05	0.05	**5.73**	0.00	5.73	5.73
	Total												
ML-1M	KNN	166.60				568.30							
	WS	**0.12**	0.05	0.08	0.18	**0.19**	0.10	0.12	0.31	**58.30**	0.00	58.30	58.30
	Total												
ML-10M	KNN	14337.50				26976.80							
	WS	**0.51**	0.02	0.48	0.53	**0.36**	0.01	0.36	0.37	**607.00**	0.00	607.00	607.00
	Total												
Ciao	KNN	41.03				97.21							
	WS	**0.006**	0.002	0.003	0.007	**0.005**	0.002	0.004	0.008	**4.42**	0.00	4.42	4.42
	Total												

similarity remedies the drawback in the original cosine similarity measure that it could not consider the differences in rating scale between different users [9]. Formally, the *adjusted-cosine similarity* between two items i and j, denoted by $w^{acs}_{i,j}$, is defined as

$$w^{acs}_{i,j} = \frac{\sum_{u\in U_{i,j}}(r_{u,i} - \bar{r}_u)(r_{u,j} - \bar{r}_u)}{\sqrt{\sum_{u\in U_{i,j}}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{u\in U_{i,j}}(r_{u,j} - \bar{r}_u)^2}}, \tag{5}$$

where $U_{i,\,j}$ represents a set of users who rated both items i and j; $\bar{r}_u$ does the average of ratings by those users in $U_{i,\,j}$ on item i [3].

3. Preliminary experiments

Before explaining our approach, we briefly report the results of our preliminary experiments for the purpose of emphasizing the necessity of the proposed approach. The key observations that motivate our approach are as follows:

- O1: The main performance bottleneck in *NBMs* is a step to search for k-nearest neighbors.
- O2: *NBMs* are generally more accurate than *LFMs* if a dataset is extremely sparse.

Note that this section skips the details of the experimental environment including datasets, accuracy measures, and characteristics of computing machines, which can be found in Section 5.1. This section focuses on presenting our experimental results.

Aiming to confirm O1, we measured the execution time from a recommendation request by a target user to its completion, performed by two *NBMs* (denoted as *IBM* and *UBM*) under MovieLens [26] and Ciao [27] datasets. We also measured the execution time of *NBMs* that pre-compute the k-nearest neighbors of all users (and items) and store them offline, denoted as *IBM_pre* and *UBM_pre*, and the execution time of *SVD* [10], which is the most popular method in *LFMs*, as baseline methods.

Table 1 shows the average, standard deviation, minimum, and maximum of execution times (in seconds) consumed by each method. In the table, *WS* indicates the time for taking the weighted combination and *KNN* does the time for finding k-nearest neighbors[2]. The biased ratings of every user and item, i.e., $b_{u,\,i}$ in Eqs. (1) and (3), are also estimated in parallel with the *KNN* step. Since this computation could be performed independently of the *KNN* step while taking much less time[3], we do need to not consider its running as a part of the total recommendation time. As shown in the table, *NBMs* spend most of the execution time in finding k-nearest neighbors. Moreover, it is dramatically increasing especially in MovieLens 10M dataset, which is the largest one among the three datasets. Due to their significantly large running time, it is essential for *NBMs*, especially for those employed by the stores having large numbers of users and items, to pre-compute similarities between all pairs of users (or all pairs of items) offline, which is doomed to result in inaccurate recommendation results [20].

In order to confirm O2, we compare the accuracies of *NBMs* and *LFMs* under different datasets. The results are shown in Tables 2 and 3 where the float number in each cell represents recommendation accuracy of each method, computed by different metrics, e.g., *Pr*@10 indicates the *precision* of top-10 recommendations. We highlight the winner's accuracy score in bold type. We also note each dataset's percentage of sparsity. The datasets in Table 2 are the original ones while those in Table 3 are the three variations of MovieLens 100K dataset: we randomly removed ratings from the original MovieLens 100 K dataset so as to make their sparsity 99.3%, 99.5% and 99.7%, respectively. Here, since we randomly removed ratings, we repeated the sparse data generation and the accuracy measurement 100 times and then took the average of the measured accuracy scores. We observe that the accuracy of *SVD* is only comparable to that of *IBM* in the MovieLens 100K dataset and falls to lower than *NBMs* in the other datasets, all of which are sparser ones than the original 100K dataset. This is because identifying latent semantic factors in rating patterns becomes more and more difficult as more ratings are missing [13–15].

[2] We set the number of nearest neighbors k as 80, following the most recent paper [4].

[3] It takes only 0.034 seconds, 0.101 seconds, 0.994 seconds, and 8.913 seconds on the Ciao dataset, MovieLens 100K, 1M, and 10M dataset, respectively.

Table 2
Accuracy of CFs in real-world datasets.

Original datasets		UBM	IBM	SVD
MovieLens 100K (95%)	Pr@10	0.001	**0.037**	0.036
	Pr@20	0.002	**0.035**	0.031
	Re@10	0.003	0.060	**0.071**
	Re@20	0.010	**0.117**	0.113
	nDCG@10	0.002	0.048	**0.058**
	nDCG@20	0.004	0.069	**0.074**
	MRR	0.018	0.098	**0.122**
MovieLens 1M (97%)	Pr@10	0.123	**0.133**	0.021
	Pr@20	0.117	**0.118**	0.014
	Re@10	0.042	**0.045**	0.006
	Re@20	0.075	**0.076**	0.009
	nDCG@10	0.133	**0.137**	0.019
	nDCG@20	**0.133**	**0.133**	0.016
	MRR	**0.315**	0.282	0.052
Ciao (99%)	Pr@10	0.003	**0.004**	**0.004**
	Pr@20	**0.004**	**0.004**	**0.004**
	Re@10	0.016	**0.021**	0.018
	Re@20	0.018	**0.039**	0.122
	nDCG@10	0.009	**0.012**	0.011
	nDCG@20	0.015	**0.017**	0.015
	MRR	**0.063**	0.023	0.022

Table 3
Accuracy of CFs in variations of the 100 K dataset.

Variations of 100K		UBM	IBM	SVD
MovieLens 100K (99.3%)	Pr@10	**0.031**	0.024	0.019
	Pr@20	**0.028**	0.025	0.017
	Re@10	**0.053**	0.046	0.028
	Re@20	**0.095**	0.094	0.052
	nDCG@10	**0.048**	0.036	0.028
	nDCG@20	**0.061**	0.053	0.035
	MRR	**0.107**	0.084	0.074
MovieLens 100K (99.5%)	Pr@10	**0.039**	0.032	0.019
	Pr@20	**0.034**	0.032	0.016
	Re@10	**0.067**	0.058	0.028
	Re@20	0.111	**0.112**	0.047
	nDCG@10	**0.060**	0.048	0.027
	nDCG@20	**0.074**	0.067	0.032
	MRR	**0.129**	0.105	0.069
MovieLens 100K (99.7%)	Pr@10	0.041	**0.043**	0.016
	Pr@20	0.038	**0.040**	0.014
	Re@10	0.072	**0.077**	0.024
	Re@20	0.134	**0.137**	0.043
	nDCG@10	0.062	**0.066**	0.024
	nDCG@20	0.082	**0.086**	0.030
	MRR	0.129	**0.136**	0.069

4. The proposed approach

The experimental results presented in the previous section motivate us to focus on improving the efficiency of *NBMs*, by speeding up the step of finding the *k*-nearest neighbors, rather than pre-computing the *k*-nearest neighbors or using *LFMs*. If the performance issue in *NBMs* can be relieved, they may provide more effective and efficient recommendations than conventional *NBMs* as well as *LFMs*, particularly under extremely sparse datasets that are very common in real-world situations [18,28]. Towards this goal, we develop novel data structures representing users, items, and the associated ratings, rather than using the conventional user-item rating matrix. We identify that the number of ratings used in similarity computation can be significantly reduced while the resulting similarities unchanged. This is due to the inherent characteristics of the similarity measures shown in Eqs. (4) and (5), which are leading to the following lemmas. Note that Lemmas 1 and 2 support *UBM*, and Lemmas 3 and 4 support *IBM*.

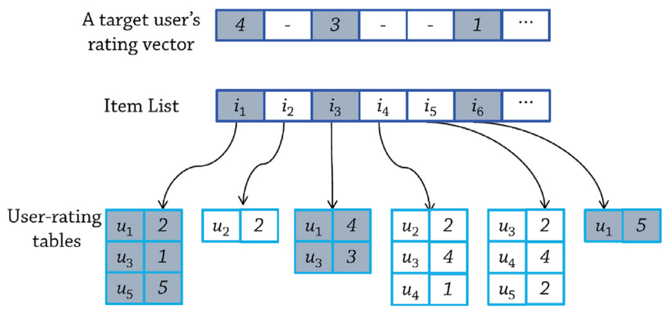


Fig. 1. Example of our data structures for *UBM*.

- ***Lemma 1.****If $I_{u,v} = \emptyset$ for two users x and y, their similarity cannot be defined.*
- ***Lemma 2.****Any items outside $I_{u,\,v}$ are not involved in computing the similarity score.*
- ***Lemma 3.****If $U_{i,\,j} = \emptyset$ for two items i and j, their similarity cannot be defined.*
- ***Lemma 4.****Any users outside $U_{i,\,j}$ are not involved in computing the similarity score.*

It is trivial to prove the lemmas by referring to the definitions of the similarity measures defined in Eqs. (4) and (5). Based on Lemma 1, *UBM* can safely ignore those users who do not have any ratings on the items that have been rated by a target user. Based on Lemma 2, *UBM* keeps only those items that are commonly rated by both the target user and one of the rest users, and then discards the rest items. Lemmas 3 and 4 support *IBM* in a very similar way as Lemmas 1 and 2 support *UBM*. Based on Lemma 3, *IBM* can safely ignore those items rated by none of users who rate a target item[4]. Based on Lemma 4, *IBM* keeps only those users who have rated both the target item and the one of the rest items. Following those lemmas, we can safely dismiss the users and items that are unnecessary in computing the similarity without any loss of accuracy.

4.1. Data structures

Now, the question is *how to retrieve such necessary users and items efficiently in order to filter out the rest?* To answer this, we introduce two types of data structures, each of which supports *UBM* and *IBM*, respectively, to identify only the necessary users and items efficiently in finding *k*-nearest neighbors.

4.1.1. Data structure for UBM

The data structure for *UBM* consists of two types of lists: (1) the *item list* and (2) the *user-rating tables*. Refer to Fig. 1 as a working example. The *item list* plays a role of an index by using item IDs. The *user-rating table* associated with each item stores pairs of a user and his/her rating score on the corresponding item. Given a target user's rating vector, *UBM* first identifies the items that the target user has rated and then accesses their corresponding user-rating tables by referring to our *item list*. Then, it computes similarities between the target user and the users only in the retrieved tables. In addition, only the ratings in the tables are necessary in finding the *k*-nearest neighbors.

In Fig. 1, suppose the given target user u_t has rated items i_1, i_3 and i_6. *UBM* accesses the user-rating tables pointed by i_1, i_3 and

[4] Here, it also ignores the items that have not been rated by the target user, following the definition of *IBM* in Eq. (3)

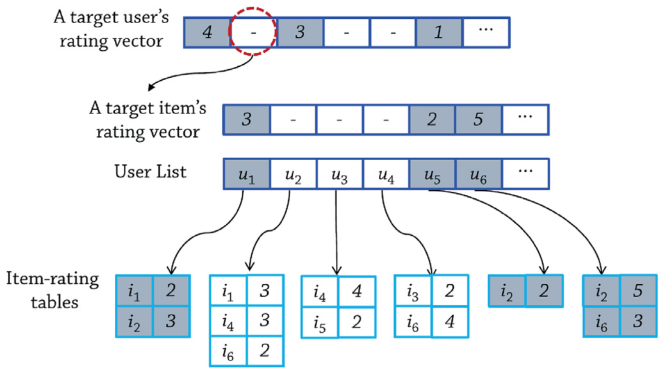


Fig. 2. Example of our data structures for *IBM*.

i_6 in *item list*. The ratings by target user u_t and some user u_k in the retrieved tables are the only information necessary for computing similarities between u_t and some user u_k so as to find u_t's k-nearest neighbors.

4.1.2. Data structure for IBM

As mentioned in Section 2, *IBM* needs to repeatedly find $I^k_{u,i}$ for each target item. Given a target user's rating vector, target items are those not rated by the target user yet. Given each target item, now the proposed data structures for *IBM* work very similarly to those for *UBM*. It consists of two types of lists: (1) the *user list* and (2) the *item-rating tables*, which are shown in Fig. 2. The *user list* plays a role of an index by using user IDs. The *item-rating table* associated with each user stores pairs of an item and the rating score on the item given by the corresponding user. For each given target item, *IBM* first identifies the users who have rated the target item and then accesses their corresponding item-rating tables by referring to our *user list*. Then, it computes similarities between the target item and the items only in the retrieved item-rating tables by using only the ratings in the tables, eventually finding top-k most similar items. Such procedure is repeated for each target item.

In Fig. 2, suppose a target user u_t as the same guy in Fig. 1, *IBM* starts to find k most similar items for each target item that is not rated by the target user. The first target item is i_2, on which users u_1, u_5 and u_6 have rated. Then, *IBM* accesses the item-rating tables pointed by the u_1, u_5 and u_6 in *user list*. The ratings given to the target item i_2 and some item i_k in the retrieved tables are the only information necessary for computing similarities between the target item i_2 and i_k so as to find i_2's the k-nearest neighbors.

4.2. Dynamic update

Our data structures enable *NBMs* to reflect the latest ratings of users to their recommendation by updating any possible changes in real time. First, as for the data structures for *UBM*, when a user gives a new rating to an item, the system accesses the user-rating table associated with the rated item and adds a new user-rating entry to the table. If a new user joins the store, it adds her/him as soon as s/he rates an item. It is also possible to cope with situations in which users, items, and ratings are deleted, even if they occur rarely. When a user withdraws the store, it accesses the user-rating tables belonging to the items rated by him, and then removes his user-rating entries one by one. When an item is added or deleted, the item's ID is simply added to or deleted from the item list.

Similarly, as for the data structures for *IBM*, when a user gives a new rating to an item (or deletes a previous rating), the system accesses the item-rating tables associated with the user and adds a new item-rating entry to the table (or deletes the corresponding item-rating pair). If a user is added or deleted, the user's ID is simply added to or deleted from the user list. When the store brings out a new item, it is added as soon as a user rates it. It is clear that all the update mechanisms mentioned above can be performed in real time, requiring a very small overhead. In this sense, the proposed data structures for both *UBM* and *IBM* are capable of handling dynamic situations efficiently.

5. Evaluation

In this section, we evaluate our proposed approach through extensive experiments. Throughout our experiments, we aim at answering the following key questions: "*How much efficient are NBMs with our data structures compared to existing methods without them? How much does our approach improve conventional NBMs?*"

5.1. Setup

In our experiments, we used MovieLens 100K, 1M, and 10M datasets and Ciao dataset [27]. Their detailed characteristics are shown in Table 4. In both of two datasets, the ratings are integer values from 1 (i.e., worst) to 5 (i.e., best). Among the items, those rated as 5 are considered as our ground truth. In order to evaluate each recommendation method, we chose one user among all users in a dataset as a target user and let every method to recommend the top-10 and top-20 items to the user. We measured the recommendation accuracy and execution time provided by each method. As accuracy metrics, we used the following ones: *precision, recall, normalized discounted cumulative gain* (*nDCG*), and *mean reciprocal rank* (*MRR*). All of their details can be found in [11]. All the accuracies were averaged by using 5-cross validation. We also averaged all the execution times taken for each method to provide recommendation to every user in a dataset. All the experiments were conducted on a Windows7 64-bit operating system equipped with i7 3770k Intel CPU and 32 GB RAM.

We compare the following CF methods: (1) conventional *NBMs* that search for k-nearest neighbors *sequentially*, denoted as *IBM* and *UBM*; (2) *NBMs* with our proposed data structures, *IBM_ours* and *UBM_ours*; (3) *SVD* that is a representative of *LFMs*. Both *IBM* and *IBM_ours* use the *adjusted-cosine similarity* as a similarity metric for *items*, and both *UBM* and *UBM_ours* use the *Pearson correlation coefficient* as a similarity metric for *users*.

5.2. Efficiency

Table 5 reports the average, standard deviation, minimum, and maximum of execution times in recommending items to a target user by *NBMs* with our data structures. Note that we already have reported the results of conventional *NBMs* and *SVD* on Table 1 in Section 3. Comparing them with conventional *NBMs*, we confirm significant speedup in the *KNN* step with using our proposed data structures. More specifically, *IBM_ours* performs 11.1 times, 6.8 times, 10 times, 58.5 times faster than conventional *IBM* under Ciao, MovieLens 100K, 1M, 10M datasets, respectively. Also, *UBM_ours* does 10.2 times, 5.7 times, 6 times, 72.9 times faster than conventional *UBM* under Ciao, MovieLens 100 K, 1 M, 10 M datasets, respectively. As the size of a dataset gets larger, the effect of our data structures on performance improvement becomes greater. We also observe our approaches perform faster than *SVD* by a wide margin (see Table 1 in Section 3).

One may claim to use *NBMs* with pre-computed k-nearest neighbors, which can provide recommendation much faster than ours, taking at most 0.51 s even in MovieLens 10M dataset (see Table 1 in Section 3). However, as mentioned earlier, they inherently have a limitation of neglecting the latest ratings, which are quite important information to predict a target user's preference on items. To highlight such limitations, we conducted

Table 4
Characteristics of MovieLens datasets.

	ML-100k	ML-1M	ML-10M	Ciao
# of users	943	6,040	69,878	996
# of items	1,682	3,900	10,681	1,359
# of ratings	100,000	1,000,000	10,000,000	18,648

Table 5
Efficiency of our approach (in seconds).

Data		IBM_ours				UBM_ours			
		avg	std	min	max	avg	std	min	max
ML-100K	KNN	1.54	0.80	0.91	1.93	1.35	0.18	1.33	1.36
	WS	0.04	0.01	0.04	0.05	0.05	0.02	0.04	0.58
	Total	**1.58**	0.81	0.95	1.98	**1.40**	0.20	1.37	1.94
ML-1M	KNN	8.90	1.21	7.55	11.43	4.34	2.49	2.30	7.13
	WS	0.10	0.06	0.04	0.19	0.17	0.05	0.09	0.31
	Total	**9.00**	1.27	7.59	11.62	**4.51**	2.54	2.39	7.44
ML-10M	KNN	116.20	56.33	78.98	165.72	10.70	3.30	7.71	13.70
	WS	0.48	0.004	0.48	0.48.52	0.35	0.00	0.35	0.36
	Total	**116.68**	56.33	79.46	165.72	**11.05**	3.30	8.06	14.06
Ciao	KNN	0.75	0.18	0.62	0.84	0.67	0.19	0.53	0.72
	WS	0.003	0.002	0.002	0.005	0.003	0.001	0.003	0.004
	Total	**0.76**	0.18	0.62	0.85	**0.68**	0.19	0.53	0.73

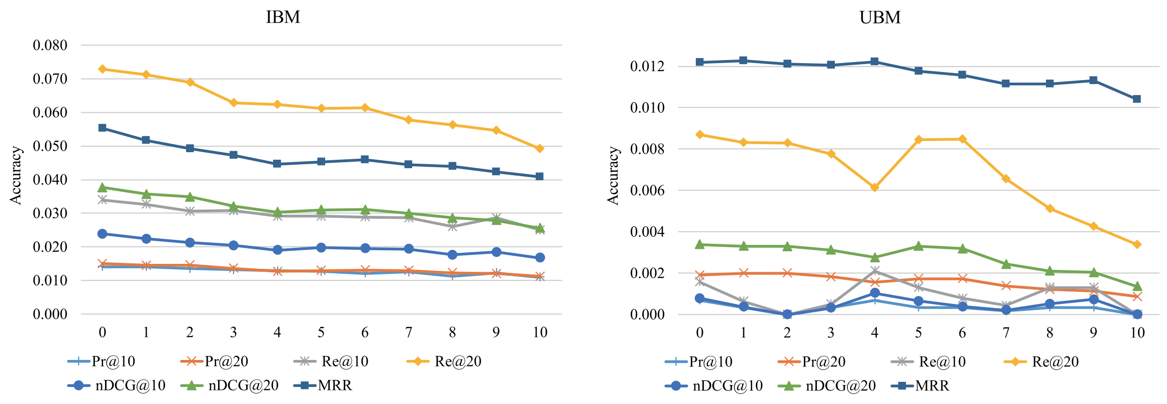


Fig. 3. Accuracy of each CF method according to their usage of the latest ratings.

another set of experiments comparing the accuracy of recommendation depending on whether it reflects the latest ratings or not. More specifically, we set the most recent 10% ratings as the test set (i.e., ground truth for recommendation) and the rest 90% of ratings as the training set. Then, among the ratings in the training set, we incrementally removed each user's the most recent 1% ratings. At each step we measured the recommendation accuracy provided by each method.

Fig. 3 depicts the results: the *y*-axis indicates the accuracy; the numbers in the *x*-axis indicate the percentages of the removed latest ratings. Although some exceptions are made by *UBM*, the overall accuracy of the *NBMs* tends to decrease as more recent ratings could not be exploited to the recommendation. In other words, neglecting the latest ratings may cause the quality of recommendation to degrade. Considering the running time of pre-computing similarities among all users or all items, which is already shown on Table 1 (denoted as *IBM_pre* and *UBM_pre*), it is also infeasible to repeat such pre-computations whenever new information is added to a dataset. In contrast, *NBMs* with our data structures can exploit the latest ratings as well as run fast enough, not requiring any offline computations. In conclusion, our approach enables *NBMs* to achieve reasonable tradeoff between accuracy and efficiency in recommendation.

Table 6
Accuracy of CF methods for cold-start users.

		UBM	IBM	SVD
MovieLens 100K	Pr@10	0.003	**0.033**	0.013
	Pr@20	0.005	**0.028**	0.014
	Re@10	0.004	**0.050**	0.034
	Re@20	0.011	**0.080**	0.073
	nDCG@10	0.005	**0.044**	0.025
	nDCG@20	0.008	**0.054**	0.039
	MRR	0.026	**0.098**	0.058

5.3. Recommendation in sparser settings

For recommender systems, it is important to evaluate the accuracy and efficiency of CF methods for more extreme situations where data is sparser. Here, we design two scenarios: recommen-

Table 7
Execution time of CF methods for cold-start users.

	IBM				UBM				SVD			
	avg	std	min	max	avg	std	min	max	avg	std	min	max
KNN	11.02	0.52	8.43	12.01	7.64	0.04	7.58	7.82	4.81	0.00	4.81	4.81
WS	0.002	0.001	0.001	0.002	0.003	0.001	0.003	0.004				
Total	**11.03**	0.52	8.43	12.01	**7.64**	0.04	7.58	7.82	**4.81**	0.00	4.81	4.81

	IBM_ours				UBM_ours			
	avg	std	min	max	avg	std	min	max
KNN	1.85	1.02	0.94	2.12	1.32	0.01	1.29	1.34
WS	0.003	0.002	0.001	0.004	0.003	0.002	0.001	0.004
Total	**1.86**	1.02	0.94	2.13	**1.32**	0.01	1.29	1.34

Table 8
Execution time of CF methods for variations of MovieLens 100K dataset.

MovidLens 100K 99.7%	IBM				UBM				SVD			
	avg	std	min	max	avg	std	min	max	avg	std	min	max
KNN	8.18	1.12	7.91	8.53	7.20	0.13	7.10	7.36	0.34	0.00	0.34	0.34
WS	0.003	0.002	0.001	0.004	0.003	0.001	0.003	0.004				
Total	**8.18**	1.12	7.91	8.53	**7.20**	0.13	7.10	7.36	**0.34**	0.00	0.34	0.34

MovidLens 100K 99.7%	IBM_ours				UBM_ours			
	avg	std	min	max	avg	std	min	max
KNN	0.22	0.09	0.21	0.24	0.08	0.00	0.07	0.08
WS	0.003	0.001	0.002	0.004	0.003	0.001	0.001	0.004
Total	**0.22**	0.09	0.21	0.25	**0.08**	0.00	0.07	0.08

MovidLens 100K 99.5%	IBM				UBM				SVD			
	avg	std	min	max	avg	std	min	max	avg	std	min	max
KNN	8.59	1.43	7.82	9.12	7.34	0.11	7.23	7.46	0.57	0.00	0.57	0.57
WS	0.003	0.001	0.002	0.004	0.004	0.002	0.001	0.005				
Total	**8.59**	1.43	7.82	9.12	**7.34**	0.11	7.23	7.47	**0.57**	0.00	0.57	0.57

MovidLens 100K 99.5%	IBM_ours				UBM_ours			
	avg	std	min	max	avg	std	min	max
KNN	0.25	0.13	0.18	0.30	0.09	0.00	0.08	0.09
WS	0.002	0.001	0.001	0.002	0.003	0.002	0.001	0.004
Total	**0.25**	0.13	0.18	0.30	**0.09**	0.01	0.08	0.09

MovidLens 100K 99.3%	IBM				UBM				SVD			
	avg	std	min	max	avg	std	min	max	avg	std	min	max
KNN	9.42	1.32	9.13	9.78	7.52	0.16	7.34	7.66	1.16	0.00	1.16	1.16
WS	0.003	0.002	0.002	0.005	0.003	0.001	0.003	0.004				
Total	**9.42**	1.32	9.13	9.79	**7.52**	0.16	7.34	7.66	**1.16**	0.00	1.16	1.16

MovidLens 100K 99.3%	IBM_ours				UBM_ours			
	avg	std	min	max	avg	std	min	max
KNN	0.32	0.10	0.27	0.34	0.23	0.01	0.21	0.24
WS	0.003	0.001	0.003	0.005	0.002	0.002	0.001	0.003
Total	**0.32**	0.10	0.27	0.35	**0.23**	0.01	0.21	0.24

Table 9
Percentage of time decrease in *KNN* step.

	99.3	**99.5**	**99.7**
IBM_ours	**79.22%**	**83.76%**	**85.71%**
IBM	11.96%	19.71%	23.55%
UBM_ours	**82.90%**	**93.30%**	**94%**
UBM	2.84%	5.16%	6.97%

Table 10
Accuracy of CF methods equipped with the *zero-injection* [11] approach.

		UBM	IBM	SVD
MovieLens 100K	Pr@10	0.084	0.125	**0.126**
	Pr@20	0.089	**0.104**	0.101
	Re@10	0.170	0.250	**0.251**
	Re@20	0.332	**0.381**	0.367
	nDCG@10	0.106	0.219	**0.220**
	nDCG@20	0.166	**0.260**	0.256
	MRR	0.130	**0.344**	0.342

dation for *cold-start users* [4,16,22] and recommendation under extremely sparse datasets. First, the cold-start users are those users who just have joined (i.e., rarely used) the store and thus have few ratings (in this paper, we define them as those who have rated less than 5 items). Due to the lack of their prior ratings, which are used to capture their preferences on items, recommender systems may have a difficulty in providing good recommendations to them. *Accurate* as well as *efficient* recommendation is critical since it can stimulate the users' appetite to spend and eventually make them keep using the store steadily. In this experiment, we simulated the

Table 11
Execution time of CF methods equipped with the *zero-injection* [11] approach.

		IBM				UBM				SVD			
		avg	std	min	max	avg	std	min	max	avg	std	min	max
100K	DI	0.46											
	KNN	13.18	1.38	11.86	13.52	7.96	0.17	7.84	8.16	29.60	0.00	29.60	29.60
	WS	0.04	0.02	0.32	0.44	0.04	0.01	0.34	0.43				
	Total	**13.68**	1.86	12.64	14.42	**8.46**	0.64	8.64	9.05	**30.06**	0.46	30.06	30.06

		IBM_ours				UBM_ours			
		avg	std	min	max	avg	std	min	max
100K	DI	0.46							
	KNN	1.97	1.02	1.44	2.84	1.68	0.04	1.65	1.73
	WS	0.04	0.01	0.04	0.05	0.051	0.02	0.04	0.58
	Total	**2.47**	1.49	1.94	3.35	**2.19**	0.52	2.15	2.77

cold-start user setting by following [4,22]. Then, we evaluated the accuracy and execution time of each method under this setting.

Tables 6 and 7 report the results of accuracy and execution time, respectively[5]. In Table 6, we note that *UBM* (or *IBM*) represents both conventional *UBM* and *UBM_ours* (or *IBM* and *IBM_ours*) since they provide the same recommendation results with each other. We observe that *IBM* provides the highest accuracy. In Table 7, as expected, *IBM_ours* and *UBM_ours* outperform the other ones in terms of efficiency. Focusing on the *KNN* step, we observe that all the methods take a little less time to find the *k*-nearest neighbors compared to the time required in normal (i.e., non-cold-start) settings, due to the reduced number of ratings and the associated computations. In terms of the percentage of the time decrease, *IBM_ours* and *UBM_ours* take 25.6% and 33.3% less time, respectively, while conventional *IBM* and *UBM* take only 10.4% and 1.2% less time, respectively. Since conventional *NBMs* originally consider many ratings which are actually unnecessary into the similarity computations, they are not so influenced by the decrease (or increase) in the number of ratings. However, our approaches are quite influenced since we utilize only the necessary ratings. Therefore, our approaches accelerate *NBMs* to provide recommendation more effectively in sparser settings, which appears more strongly in the next experiments.

Second, we generate datasets that are extremely sparse: three variations of the MovieLens 100K dataset with overall sparsity 99.3%, 99.5%, and 99.7%. We have already introduced how we generate the data and conduct experiments in Section 3 and also have reported the results in terms of accuracy. Here, we report the execution time on Table 8. Again, all the methods take less time to find *k*-nearest neighbors, compared to the time obtained with the original MovieLens 100K dataset. Remarkably, the percentage of the time decrease is much more significant in our proposed *NBMs* than in conventional *NBMs*, which are summarized in Table 9 (e.g., *UBM_ours* takes 94% less time while conventional *UBM* takes only 6.97% less time). This again confirms that our data structures enhance the efficiency of *NBMs*: As the dataset gets sparser, the efficiency gain becomes higher.

5.4. Incorporated with data imputation approach

Recently, a wide variety of approaches for improving the accuracy of top-*N* recommendation have been extensively studied. Some of them are acquiring additional information from a trust network [18,27,29] or crowdsourcing [30,31], and some others are performing data imputation [32]. Among them, the data imputation approaches, which *infer* the ratings of users on *unrated* items by using rated items and impute the inferred ratings to a user-item rating matrix, have been widely used because they do not require any additional information from other sources (e.g., trust networks and crowdsourcing). They are also known effective enough to improve the accuracies of popular CF methods by two to five orders of magnitude on average [11].

In this experiment, we aimed to confirm that our approach can be well incorporated with the data imputation approaches. To this end, we set experimental environments and the approach for data imputation by following [11], a state-of-the-art work in data imputation approaches. We applied their imputation approach to *NBMs* and then found the top-*N* recommended items with using our data structures. We measured the accuracy and the time taken within all those procedures. We compare the results with those of conventional *NBMs* and *SVD* incorporated with the same data imputation approach.

Table 10 and 11 report the results of accuracy and execution time, respectively. In Table 10, *IBM* indicates the highest accuracy. In Table 11, *DI* denotes the time to infer missing ratings and to impute them to a dataset. Here, contrary to the results in the previous section, both conventional *NBMs* and our *NBMs* take a little more time in the *KNN* step compared to those without imputed ratings, due to the increased number of ratings and the associated computations. Specifically, *IBM_ours* and *UBM_ours* take 38.4% and 24.4% more time, respectively, while conventional *IBM* and *UBM* take only 11.2% and 2.8% more time, respectively. The percentage of the time increase is higher in ours than existing ones because conventional *NBMs* originally consider many ratings (i.e., actually unnecessary) into the similarity computations while our approaches utilize only the essential ratings. Nonetheless, *IBM_ours* and *UBM_ours* perform faster than the other methods, which confirms that our approach orthogonally improves the efficiency of *NBMs* incorporated with the state-of-the-art data imputation approach. Even though we do not show here, our approach can be also well incorporated to any other approach borrowing additional information such as trust networks or crowdsourcing.

6. Related work

This section briefly reviews existing approaches dealing with the efficiency issue in CF algorithms. They can be roughly classified into two categories: (a) the approaches dealing with the efficiency in *NBMs* and (b) those dealing with the efficiency in *LFMs*. Among them, we focus on category (a), rather than category (b), since category (a) is more closely related to our work.

Liang et al. [33] exploited *locality sensitive hashing* (LSH) as a tool to efficiently find similar users or items. However, since LSH *approximates* the *k*-nearest neighbors, their approach has a loss of recommendation accuracy. Moreover, their method requires the target user's additional profile information such as demographic information. On the other hand, our proposed approach neither

[5] We note that the results in Tables 6 and 10 are not in consequence of our contribution.

causes a loss of recommendation accuracy nor requires such extra information. Hwang et al. [20] improved the execution time of *NBM*s by using the notion of *category experts*, which is defined as the top-*k* users in giving ratings in a certain category [34]. The authors selected a few users as experts in each category and used their ratings, rather than ordinary neighbors' ones. They provided promising running time because the experts could be easily identified by just counting the number of ratings given by each user. However, their recommendation is difficult to be personalized because the category experts are always the same regardless of the target user, which consequently results in the loss of accuracy. Liu et al. [35] proposed an evolutionary collaborative filtering algorithm that updates a previously built model incrementally with the new rating data, under the *NBM* framework. In other words, their idea is to compute incrementally the user-by-user similarities or item-by-item similarities (i.e., model) whenever the batch of new rating data arrives, rather than re-building the model from the scratch (i.e., using both old and new data). In their experiments, they reported about 50 min to update the model with using 33K ratings, which is, however, infeasible when applied to online recommendation.

Most of the approaches dealing with the efficiency of *LFM*s aim at performing matrix factorization in parallel using GPUs or a distributed computing framework such as Spark. Li et al. [36] and Wang et al. [37] demonstrated how they use GPUs to accelerate the process of computing matrix factorization models. Winlaw et al. [38] and Xie et al. [39] introduced the parallel implementation of the *alternating least squares* (ALS) algorithm which is a common solution to matrix factorization problems, based on the Spark platform. He et al. [21] proposed an incremental matrix factorization algorithm aiming at online recommendation.

7. Conclusions

In recommender systems, conventional *NBM*s have tradeoff between recommendation accuracy and performance. Also, *LFM*s suffer from low accuracy when a dataset is sparse. Motivated by those points, we have proposed an approach for *NBM*s to efficiently retrieve only those users and items necessary in finding the nearest neighbors. To realize it, we have introduced two data structures that can be easily updated in real time when any possible change happens as well as making our *NBM*s provide exactly the same recommendation results with conventional *NBM*s, providing tens of times improved efficiency. Through our extensive experiments, we have demonstrated that our approach successfully solves the performance bottleneck that conventional *NBM*s suffer from, achieving up to 72.9 times faster execution time. We finally have evaluated the accuracy and efficiency of our approach from the viewpoints of recommendation in sparser settings and also incorporation with existing data imputation approaches.

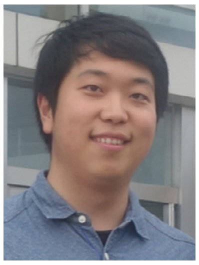
Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIP; Ministry of Science, ICT & Future Planning) (No. NRF-2017R1A2B3004581).

References

[1] H.-S. Chiang, T.-C. Huang, User-adapted travel planning system for personalized schedule recommendation, Inf. Fusion 21 (2015) 3–17.
[2] J. Ha, S.-H. Kwon, S.-W. Kim, C. Faloutsos, S. Park, Top-n recommendation through belief propagation, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ACM, 2012, pp. 2343–2346.
[3] X. Su, T.M. Khoshgoftaar, A survey of collaborative filtering techniques, Adv. Artif. intell. 2009 (2009) 4.
[4] J. Lee, D. Lee, Y.-C. Lee, W.-S. Hwang, S.-W. Kim, Improving the accuracy of top-n recommendation using a preference model, Inf. Sci. 348 (2016) 290–304.
[5] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, Z. Chen, Scalable collaborative filtering using cluster-based smoothing, in: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2005, pp. 114–121.
[6] J.S. Breese, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.
[7] G. Linden, B. Smith, J. York, Amazon. com recommendations: item-to-item collaborative filtering, IEEE Internet Comput. 7 (1) (2003) 76–80.
[8] M. Deshpande, G. Karypis, Item-based top-n recommendation algorithms, ACM Trans. Inf. Syst. (TOIS) 22 (1) (2004) 143–177.
[9] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th International Conference on World Wide Web, ACM, 2001, pp. 285–295.
[10] Y. Koren, R. Bell, C. Volinsky, et al., Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.
[11] W.-S. Hwang, J. Parc, S.-W. Kim, J. Lee, D. Lee, Told you i didn't like it: exploiting uninteresting items for effective collaborative filtering, in: Proceedings of the IEEE 32nd International Conference on Data Engineering, IEEE, 2016, pp. 349–360.
[12] P. Cremonesi, Y. Koren, R. Turrin, Performance of recommender algorithms on top-n recommendation tasks, in: Proceedings of the fourth ACM Conference on Recommender Systems, ACM, 2010, pp. 39–46.
[13] L. Zheng, A survey and critique of deep learning on recommender systems, 2016. http://bdsc.lab.uic.edu/docs/survey-critique-deep.pdf.
[14] S. Li, J. Kawale, Y. Fu, Deep collaborative filtering via marginalized denoising auto-encoder, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 811–820.
[15] B. Kanagal, A. Ahmed, S. Pandey, V. Josifovski, J. Yuan, L. Garcia-Pueyo, Supercharging recommender systems using taxonomies for learning user purchase behavior, Proc. VLDB Endow. 5 (10) (2012) 956–967.
[16] L.H. Son, Dealing with the new user cold-start problem in recommender systems: a comparative review, Inf. Syst. 58 (2016) 87–104.
[17] B. Li, Q. Yang, X. Xue, Transfer learning for collaborative filtering via a rating-matrix generative model, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 617–624.
[18] W.-S. Hwang, S.-Y. Li, S.-W. Kim, H.J. Choi, On exploiting trustors in trust-based recommendation, J. Internet Technol. 16 (4) (2015) 755–765.
[19] F. Zhang, N.J. Yuan, D. Lian, X. Xie, W.-Y. Ma, Collaborative knowledge base embedding for recommender systems, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 353–362.
[20] W.-S. Hwang, H.-J. Lee, S.-W. Kim, Y. Won, M.-s. Lee, Efficient recommendation methods using category experts for a large dataset, Inf. Fusion 28 (2016) 75–82.
[21] X. He, H. Zhang, M.-Y. Kan, T.-S. Chua, Fast matrix factorization for online recommendation with implicit feedback, in: Proceedings of SIGIR, vol. 16, 2016.
[22] H. Ma, I. King, M.R. Lyu, Effective missing data prediction for collaborative filtering, in: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2007, pp. 39–46.
[23] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 426–434.
[24] M.R. McLaughlin, J.L. Herlocker, A collaborative filtering algorithm and evaluation metric that accurately model the user experience, in: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2004, pp. 329–336.
[25] G. Chowdhury, Introduction to Modern Information Retrieval, Facet publishing, 2010.
[26] F.M. Harper, J.A. Konstan, The movielens datasets: history and context, ACM Trans. Interact. Intell. Syst. (TiiS) 5 (4) (2016) 19.
[27] J. Tang, H. Gao, H. Liu, mTrust: discerning multi-faceted trust in a connected world, in: Proceedings of the fifth ACM International Conference on Web Search and Data Mining, ACM, 2012, pp. 93–102.
[28] A. Tuzhilin, Towards the next generation of recommender systems, in: Proceedings of the 1st International Conference on E-Business Intelligence (ICEBI2010),, Atlantis Press, 2010.
[29] W.-S. Hwang, S. Li, S.-W. Kim, H.J. Choi, Exploiting trustors as well as trustees in trust-based recommendation, in: Proceedings of the 22nd ACM international Conference on Conference on Information & Knowledge Management, ACM, 2013, pp. 1893–1896.
[30] S. Chang, F.M. Harper, L. He, L.G. Terveen, Crowdlens: experimenting with crowd-powered recommendation and explanation, in: Proceedings of the Tenth International AAAI Conference on Web and Social Media, 2016.
[31] J. Lee, M. Jang, D. Lee, W.-S. Hwang, J. Hong, S.-W. Kim, Alleviating the sparsity in collaborative filtering using crowdsourcing, in: Proceedings of the Workshop on Crowdsourcing and Human Computation for Recommender Systems (CrowdRec), vol. 5, 2013.

[32] Y. Ren, G. Li, J. Zhang, W. Zhou, Adam: adaptive-maximum imputation for neighborhood-based collaborative filtering, in: Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2013, IEEE, 2013, pp. 628–635.
[33] H. Liang, H. Du, Q. Wang, et al., Real-time collaborative filtering recommender systems, in: Proceedings of the Twelfth Australasian Data Mining Conference, Australian Computer Society Inc., 2014.
[34] W.-S. Hwang, H.-J. Lee, S.-W. Kim, M. Lee, On using category experts for improving the performance and accuracy in recommender systems, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ACM, 2012, pp. 2355–2358.
[35] N.N. Liu, M. Zhao, E. Xiang, Q. Yang, Online evolutionary collaborative filtering, in: Proceedings of the Fourth ACM Conference on Recommender Systems, ACM, 2010, pp. 95–102.
[36] F. Li, S. Zhang, Y. Ye, X. Han, GPUMF: a GPU-enpowered collaborative filtering algorithm through matrix factorization, in: Proceedings of the International Conference on Service Science (ICSS), 2015, IEEE, 2015, pp. 88–92.
[37] Z. Wang, Y. Liu, S. Chiu, An efficient parallel collaborative filtering algorithm on multi-GPU platform, J. Supercomput. 72 (6) (2016) 2080–2094.
[38] M. Winlaw, M.B. Hynes, A. Caterini, H. De Sterck, Algorithmic acceleration of parallel ALS for collaborative filtering: speeding up distributed big data recommendation in spark, in: Proceedings of the IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015, IEEE, 2015, pp. 682–691.
[39] L. Xie, W. Zhou, Y. Li, Application of improved recommendation system based on spark platform in big data analysis, Cybern. Inf. Technol. 16 (6) (2016) 245–255.



Dong-Kyu Chae received the B.S. degree in computer science from Hanyang University, Seoul, Korea, in 2012. Since 2012, he has been pursuing the Ph.D. degree in computer and software engineering at Hanyang University. His current research interests include data mining, social network analysis, and deep learning.

Sang-Chul Lee received the B.S., M.S., and Ph.D. degrees in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2005, 2007, and 2012, respectively. He worked as a postdoctoral researcher at Carnegie Mellon University from 2013 to 2015. Currently, he is a senior engineer at Hyundai Heavy Industries. His research interests include data mining, information retrieval, and recommender systems.

Si-Yong Lee received the B.S. and M.S degrees in Electronics and Computer Engineering from Hanyang University, Seoul, Korea, in 2009 and 2011, respectively. Currently, he is working at SureSoft Technologies Inc. His current research interests include recommender systems and software engineering.

Sang-Wook Kim received a B.S. degree in Computer Engineering from Seoul National University at 1989, and earned M.S. and Ph.D. degrees in Computer Science from KAIST at 1991 and 1994. From 1995 to 2003, he served as an Associate Professor of Division of Computer, Information, and Communications Engineering at Kangwon National University. In 2003, he joined Hanyang University, where he currently is a Professor at Department of Computer Science and Engineering. His research interests include databases, data mining, social network analysis, and recommendation. From 2009 to 2010, Dr. Kim visited Computer Science Department at CMU as a Visiting Professor. From 1999 to 2000, he also worked with IBM Watson Research Center as a Post-Doc. He is nowáan AssociateáEditor of Information Sciences and is a member of ACM and IEEE.