Expert Systems with Applications 42 (2015) 4022–4028

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Reversed CF: A fast collaborative filtering algorithm using a *k*-nearest neighbor graph

Youngki Park [a,*], Sungchan Park [a], Woosung Jung [b], Sang-goo Lee [a]

[a] Room 418, Building 138, Seoul National University, Sillim-9-dong, Gwanak-gu, Seoul, Republic of Korea
[b] Room 206B, Building E8-10, Chungbuk National University, 12 Gaesin-dong, Heungdeok-gu, Cheongju-si, Chungcheongbuk-do, Republic of Korea

ARTICLE INFO

Article history:
Available online 10 January 2015

Keywords:
Reversed CF
Collaborative filtering
k-Nearest neighbor graph
Greedy filtering

ABSTRACT

User-based and item-based collaborative filtering (CF) methods are two of the most widely used techniques in recommender systems. While these algorithms are widely used in both industry and academia owing to their simplicity and acceptable level of accuracy, they require a considerable amount of time in finding *top-k* similar neighbors (items or users) to predict user preferences of unrated items. In this paper, we present Reversed CF (RCF), a rapid CF algorithm which utilizes a *k*-nearest neighbor (*k*-NN) graph. One main idea of this approach is to reverse the process of finding *k* neighbors; instead of finding *k* similar neighbors of unrated items, RCF finds the *k*-nearest neighbors of rated items. Not only does this algorithm perform fewer predictions while filtering out inaccurate results, but it also enables the use of fast *k*-NN graph construction algorithms. The experimental results show that our approach outperforms traditional user-based/item-based CF algorithms in terms of both preprocessing time and query processing time without sacrificing the level of accuracy.

1. Introduction

User-based and item-based collaborative filtering (CF) methods are two of the most widely used techniques in recommender systems. When a user requests a recommendation, the user-based CF algorithm, introduced by Herlocker, Konstan, Borchers, and Riedl (1999), predicts the users preferences for all of the unrated items based on similar users' preferences for those items. In a similar way, the item-based CF algorithm presented by Sarwar, Karypis, Konstan, and Riedl (2001) predicts the preferences of the user for all unrated items based on the user's preference levels for similar items.

Cremonesi, Koren, and Turrin (2010) and Lee, Song, Kahng, Lee, and Lee (2011) state that CF algorithms produce movie recommendations of a higher quality compared to baseline algorithms, which only recommend the most popular movies or highly rated movies. Although there have been proposed more efficient algorithms, such as those that use singular vector decomposition presented by Cremonesi et al. (2010) or a random walk proposed by Lee et al. (2011) and Lee, Park, Kahng, and Lee (2013), CF algorithms are still widely used in both industry and academia owing to their simplicity and acceptable levels of accuracy. For example, the Amazon and YouTube recommender systems, introduced by Linden, Smith, and York (2003) and Davidson et al. (2010) respectively, utilize CF-based algorithms. Additionally, many modified versions of CF algorithms such as the work of Lee, Park, Kahng, Lee, and Lee (2010b) and Park, Lee, and Lee (2011) are being proposed for the purpose of building context-aware recommender systems.

One of the main drawbacks of CF algorithms is that predictions are necessary for all unrated items. While such an approach facilitates evaluations of the accuracy of various algorithms using the root-mean-square error (RMSE), this method consumes a significant amount of recommendation time. Moreover, the pre-processing time is also long, especially for a user-based CF algorithm, as it has to calculate all of the similarity values between users.

In this paper, we present Reversed CF (RCF),[1] a fast CF algorithm using a *k*-nearest neighbor (*k*-NN) graph. One main idea of this approach is that it reverses the process of finding *k* neighbors. Not only does this algorithm perform fewer predictions while filtering out inaccurate results, but it also enables the use of fast *k*-NN graph construction algorithms. The contributions of our work can be summarized as follows:

* Corresponding author.
E-mail addresses: ypark@europa.snu.ac.kr (Y. Park), baksalchan@europa.snu.ac.kr (S. Park), wsjung@cbnu.ac.kr (W. Jung), sglee@europa.snu.ac.kr (S.-g. Lee).

[1] This paper is an extended version of the work of Park, Park, Lee, and Jung (2014a).

http://dx.doi.org/10.1016/j.eswa.2015.01.001
0957-4174/

- We present RCF, a fast CF algorithm which uses a k-NN graph. Because RCF performs fewer rating predictions, the recommendation time (query processing time) is significantly reduced compared to that of a user-based or an item-based CF algorithm.
- We apply a fast k-NN graph construction algorithm known as greedy filtering to reduce the RCF pre-processing time significantly. We also apply TF-IDF weighting to our dataset before executing the greedy filtering algorithm for further improvements.
- Through experiments with different parameter settings, we show that RCF outperforms traditional user-based/item-based CF algorithms in terms of both preprocessing time and recommendation time without sacrificing accuracy.

The rest of this paper is structured as follows. In Section 2, we review user-based/item-based CF algorithms and their general optimization techniques. In Section 3, we present RCF, a fast collaborative filtering algorithm. In Section 4, we show experimental results comparing our approach to the traditional CF algorithms. Finally, we conclude the paper and present future research directions in Section 5.

2. Related work

Adomavicius and Tuzhilin (2005) classified existing recommender systems into six categories based on the types of recommendation approach (content-based filtering, collaborative filtering, and hybrid approach), and the types of recommendation techniques (heuristic-based approach and model-based approach) for the rating estimation. Although the traditional collaborative and heuristic-based approaches are outperformed by the different types of recommender systems, especially the model-based approaches such as one using matrix factorization introduced by Koren, Bell, and Volinsky (2009), singular vector decomposition presented by Cremonesi et al. (2010), or random walk proposed by Lee et al. (2011, 2013), in terms of prediction accuracy, Desrosiers and Karypis (2011) state that the traditional approaches are still widely used due to their simplicity, justifiability, efficiency and stability. For example, according to Linden et al. (2003) and Davidson et al. (2010), the Amazon and YouTube recommender systems exploit the collaborative and heuristic-based approaches. In this paper, we focus on the item-based CF and user-based CF algorithms, which are two of the most popular approaches among them.

User-based CF algorithms predict the preferences of all items unrated by the user based on similar user preferences for those items. According to Herlocker et al. (1999), the predicted rating for active user a for item i is defined as follows:

$$p_{a,i} = \overline{r_a} + \frac{\sum_{n \in N(a)} (r_{n,i} - \overline{r_n}) * sim(a,n)}{\sum_{n \in N(a)} |sim(a,n)|}, \tag{1}$$

where $N(a)$ denotes the set of k-nearest neighbors of a among the users that have rated item i; $r_{n,i}$ denotes the rating of item i by user n; $\overline{r_a}$ and $\overline{r_n}$ are the average ratings of user a and neighbor n, respectively; $sim(a,n)$ is the similarity between a and n. We use the Pearson correlation coefficient as the similarity measure for the user-based CF algorithm:

$$sim(a,n) = \frac{\sum_{i \in \mathcal{C}_i} (r_{a,i} - \overline{r_a})(r_{n,i} - \overline{r_n})}{\sqrt{\sum_{i \in \mathcal{C}_i} (r_{a,i} - \overline{r_a})^2} \sqrt{\sum_{i \in \mathcal{C}_i} (r_{n,i} - \overline{r_n})^2}}, \tag{2}$$

where $\mathcal{C}_i$ denotes the set of co-rated items.

Herlocker et al. (1999) and Desrosiers and Karypis (2011) state that there are common optimization techniques for the user-based CF algorithm, such as *significance weighting*, *variance weighting*, and *selecting neighborhoods*. The first two techniques are used to adjust the similarity values between users. If two users had fewer than 50 commonly rated items, significance weighting devalues the similarity between them by $(1 - \#\text{ commonly rated items}/50) * 100\%$; variance weighting decreases the influence of items with low variance, such as Titanic. The third technique selects only k neighbors when predicting the ratings of unrated items in that the use of less similar users may have a negative impact on the quality of recommendations.

Item-based CF algorithms predict the preferences of items unrated by the user based on the preference levels of similar items for the user. According to Sarwar et al. (2001), the predicted rating for active user a for unrated item i is defined as follows:

$$p_{a,i} = \frac{\sum_{n \in N(i)} r_{a,n} * sim(i,n)}{\sum_{n \in N(i)} |sim(i,n)|}, \tag{3}$$

where $N(i)$ denotes the set of k-nearest neighbors of i among the items that have been rated by active user a. In order to find $N(i)$ efficiently, the algorithm first constructs a l-nearest neighbor graph, which represents the l-nearest neighbor relationships between items. Then we can find the k number of neighbors based on this pre-computed l-NN graph instead of calculating the item-by-item similarity matrix when a recommendation is requested. In this equation, we use the adjusted cosine similarity as the similarity measure:

$$sim(i,n) = \frac{\sum_{u \in \mathcal{C}_u} (r_{u,i} - \overline{r_u})(r_{u,n} - \overline{r_u})}{\sqrt{\sum_{u \in \mathcal{C}_u} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{u \in \mathcal{C}_u} (r_{u,n} - \overline{r_u})^2}}, \tag{4}$$

where $\mathcal{C}_u$ denotes the set of co-raters. $\overline{r_u}$ is the average rating of user u.

Cremonesi et al. (2010) and Lee et al. (2011) indicate that CF algorithms produce movie recommendations of a high quality compared to baseline algorithms, which only recommend the most popular movies or highly rated movies. Although more efficient algorithms have been proposed, CF algorithms are still widely used in industry and academia due to their simplicity and acceptable levels of accuracy. However, there are several barriers preventing the realization of rapid recommendations when using existing approaches. First, it is necessary to find different neighbors depending on the active users. Specifically, the user-based CF algorithm finds the k-nearest users from among all users who have rated a certain unrated item i when predicting the rating of i, whereas the item-based CF algorithm finds the k-nearest items from among all items that have been rated by active user u. Second, it is necessary to predict all of the unrated items when a recommendation is requested by a user. This procedure is somewhat inefficient in that according to Cremonesi et al. (2010) and Park, Yang, Song, Lee, and Lee (2013b), we usually need only the top-N recommendation results in real-world scenarios. While CF algorithms would provide rapid recommendations if there were not too many recommendation requests in a short time frame, it would not be easy for commercial recommender systems in which numerous recommendations are being requested by numerous users to provide real-time recommendations. Although there have been a few approaches to reduce the recommendation time, such as the work of Sarwar et al. (2001), Das, Datar, Garg, and Rajaram (2007), and Birtolo and Ronca (2013), either the performance gain is not significant or the approaches are not based on user-based/item-based collaborative filtering.

3. Fast collaborative filtering

Our approach consists of two main steps: first, we approximately construct a k'-nearest neighbor graph (k'-NN graph) as a preprocessing step based on our previous work of Park, Park, Lee, and Jung (2013a, 2014b) (Section 3.1). Here, we usually set k' such that $I \gg k' > k$. Second, we find the k neighbors of unrated items based on the k'-NN graph. Then we recommend items to users using the k neighbors and our revised version of the non-normalized cosine neighborhood (Section 3.2).

3.1. Nearest neighbor graph construction

The construction of a k'-NN graph is a task which involves finding the k' nodes most similar to each node. Although other tasks, such as *k-NN search* presented by Datar, Immorlica, Indyk, and Mirrokni (2004) and Gan, Feng, Fang, and Ng (2012), *reverse k-NN search* proposed by Achtert et al. (2006), *similarity join* introduced by Lee, Park, Shim, and Lee (2010a) and *top-k similarity join* presented by Xiao, Wang, Lin, and Shang (2009) and Kim and Shim (2012), can be used for recommender systems, we use the k'-NN graph because it is one of the most appropriate data structure for our algorithm. One of the easiest ways to construct a k'-NN graph is to calculate the similarities between all of the nodes and extract the nodes most similar to each node. In spite of its simplicity, this brute-force approach requires quadratic time complexity, which is burdensome when used in conjunction with large amounts of data. An alternative way to cope with this problem is to use inverted indices given the fact that item-by-user matrices are usually very sparse. However, according to Park et al. (2014b), this approach is also not appropriate for handling large amounts of high-dimensional data.

Our main idea is to construct an approximate k'-NN graph based on *greedy filtering* presented by Park et al. (2013a, 2014b) in order to speed up this process. It is known that greedy filtering outperforms other k'-NN graph construction algorithms, such as *NN-Descent* proposed by Dong, Moses, and Li (2011) or *kNN-Overlap* presented by Chen, Fang, and Saad (2009), for high-dimensional sparse datasets. If there is no decline in the quality of recommendations when we use approximate graphs, we do not have to spend much time on building an exact k'-NN graph. The accuracy of the k'-NN graph is defined as follows:

$$Accuracy = \frac{\#correct\ k' - nearest\ neighbors}{\#nodes \cdot k'} \tag{5}$$

Fig. 1 shows an example of how greedy filtering constructs an approximate k'-NN graph. In this figure, there are five items and ten users; the values in the matrix indicate the ratings, each corresponding to its item and user. The main idea of greedy filtering is to filter item pairs whose "large value dimensions" (the shaded portions in the figure) do not overlap at all. In this figure, i_1 and i_2 share a common large value dimension. Hence, we calculate the similarity between i_1 and i_2. In contrast, i_2 and i_4 do not have a common large value dimension; accordingly, we do not calculate the similarity between i_2 and i_4. Park et al. (2014b) describe in detail the manner in which large value dimensions are selected for each item. In an actual implementation of this method, we use adjacency lists instead of adjacency matrices. The empirical time complexity of greedy filtering is $O(|\mathcal{I}|)$, where $\mathcal{I}$ is a set of items.

Users

Items	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
i_1	0.52		0.37	0.31				0.33	0.23	
i_2	0.73	0.55	0.1		0.37			0.05		
i_3	0.25	0.4				0.27	0.29			0.1
i_4	0.25	0.27	0.3	0.35	0.8					
i_5			0.48	0.32	0.37		0.34			0.2

Fig. 1. Example of greedy filtering.

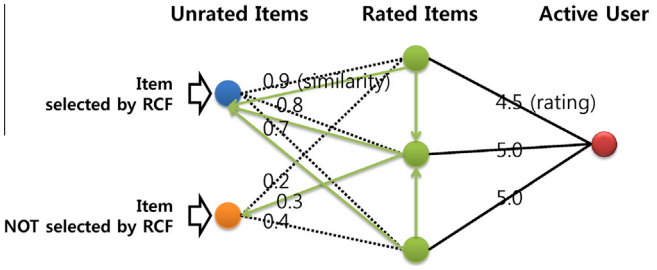


Fig. 2. Example of Reversed CF.

According to Park et al. (2014b), this algorithm performs much better when we apply the TF-IDF weighting scheme and this process does not decrease the quality of recommendations significantly. Thus we adjust the values in the input matrix based on the TF-IDF weighting scheme:

$$M'_{i,j} = \left(0.5 + \frac{0.5 \cdot M_{i,j}}{max\{M_{i,k} : u_k \in \mathcal{U}\}}\right) * log\left(\frac{|\mathcal{U}|}{F(u_j)}\right), \tag{6}$$

where $M_{i,j}$ denotes the original value of the matrix corresponding to the ith item and the jth user; $\mathcal{U}$ denotes a set of users; $F(u_j)$ denotes the number of items that have values corresponding to the users u_j.

For example, suppose that we use the cosine similarity with the TF-IDF weighting scheme and that there are two items i_1 and i_2 in a dataset. In such a case, greedy filtering would calculate their level of similarity if they are highly rated by at least one certain inactive user. Otherwise, it would filter out those item pairs.

3.2. Fast recommendation algorithm

Recall that the two main drawbacks of user-based or item-based CF algorithms are that they have to find different neighbors depending on active users and that they have to predict all of the unrated items. Our novel algorithm, RCF, solves these problems. Let $B[i]$ be a k-NN list of item i, which was already calculated in Section 3.1. Let $\mathcal{I}_u$ and $\mathcal{I}_r$ be sets of unrated items and rated items of an active user a, respectively. Then RCF works as follows:

1. For every item $i \in \mathcal{I}_u$, prepare an empty set $S[i]$.
2. For every item $i \in \mathcal{I}_r$ and every item such that $j \in \mathcal{I}_u$ and $j \in B[i]$, add i to $S[j]$.
3. For every item $i \in \mathcal{I}_u$, if $|S[i]| > k$, then delete all except for the most similar k_1 items from $S[i]$.
4. Then, predict the ratings for all items i such that $i \in \mathcal{I}_u$ and $|S[i]| = k$,

$$p_{a,i} = \overline{r}_i + \sum_{n \in S[i]} (r_{a,n} - \overline{r}_n) * sim(i,n) \tag{7}$$

Here, $sim(i,n)$ is the cosine similarity between i and n. It is defined as follows:

$$sim(i,n) = \frac{\sum_{c \in \mathcal{C}_u} r_{c,i} \cdot r_{c,n}}{\sqrt{\sum_{c \in \mathcal{C}_u} (r_{c,i})^2} \sqrt{\sum_{c \in \mathcal{C}_u} (r_{c,n})^2}} \tag{8}$$

Table 1
Summary of the recommendation algorithms.

Algorithm	Phase	Task
UserCF	Preprocessing	Similarity matrix construction Significance weighting
	Recommendation	Selecting k users that have rated the item All rating predictions
ItemCF	Preprocessing	l-NN graph construction ($l \gg k$)
	Recommendation	Selecting k items that have been rated by an active user All rating predictions
RCF	Preprocessing	k'-NN graph construction ($k' > k$)
	Recommendation	Selecting k items Fewer rating predictions
RCF + TFIDF + GF	Preprocessing	k'-NN graph construction using GF ($k' > k$)
	Recommendation	Selecting k items Fewer rating predictions

If an unrated item does not have the list of k_1 number of items, RCF does not predict its rating.

Fig. 2 shows an illustrative example of our approach. In this example, we set k and k' to 2 and 2, respectively. Thus we find 2-nearest neighbors for each rated item. An edge from a rated item i to an unrated item j indicates that j is one of the 2-nearest neighbors of i. The first unrated item has three incoming edges and the similarities between this item and the rated items are 0.9, 0.8 and 0.7 respectively. Because k is 2 in this example, we discard the edge labeled with 0.7, and predict the ratings of the item based on the remaining edges. On the other hand, the second unrated item has just one incoming edge so that we do not predict the rating of the item.

The intuition behind this algorithm is that if one of the nearest neighbors of rated item i is unrated item j, there would be a high probability that one of the nearest neighbors of unrated item j is rated item i. This is why the proposed algorithm is termed Reversed CF. One of the main characteristics of RCF is that it does not predict the preferences of all unrated items of a user. This approach does not sacrifice the level of recommendation quality for two reasons. First, if the rating of an unrated item is predicted by RCF, RCF and the item-based CF algorithm select the same neighbors for predicting the item in many cases. Second, if the rating of an unrated item is not predicted by RCF, the average similarity value of the k-nearest neighbors of the unrated item is usually lower than that of another item predicted by RCF, which is the case when it is difficult for the item-based CF algorithm to predict accurate ratings. In Section 4, we will discuss this in more detail.

4. Experiments

4.1. Experimental setup

4.1.1. Dataset and algorithms

We use the MovieLens dataset[2] for comparisons: there are 1,000,209 ratings, 3952 movies, and 6040 users; each user rates at least 20 number of items; the rating scale ranges from 1 to 5 in which higher ratings indicate greater preference. We considered four types of algorithms for a comparison: UserCF implements the work by Herlocker et al. (1999); ItemCF implements the work by Sarwar et al. (2001); RCF implements only the fast recommendation algorithm presented in Section 3.2; RCF + TFIDF + GF implements the fast recommendation algorithm presented in both Sections 3.1 and 3.2.

[2] <http://grouplens.org/datasets/movielens/>.

We set the default parameters k, k', and l to 10, 20, and 300, respectively. Table 1 summarizes the above mentioned recommendation algorithms and their related parameters.

4.1.2. Quality evaluation

We follow the testing methodology of a recommender system introduced by Cremonesi et al. (2010). We divide the ratings into two groups. One group of data consisting of 986,206 ratings (98.6% of ratings) is used for our training set, and the other group of data consisting of 14,003 ratings (1.4% of the ratings) is used for the probe set. The test set consists of all of the five-star ratings (1661 ratings) of 3719 unpopular movies (99.65% of the movies) in the probe set. Then, for each rating of movie m rated by user u in the test set, we randomly select 1000 movies unrated by u and recommend the top-N movies from among the 1001 movies (the 1000 items selected in addition to m); if we recommend m, we refer to this as a hit. Finally, we measure the degree of recall using the following equation:

$$recall = \frac{\#\ hits}{|test\ set|}. \tag{9}$$

4.1.3. Performance evaluation

We measure both the preprocessing time and the recommendation time for each algorithm. In UserCF, the preprocessing time is the overall time needed to construct the user-by-user similarity matrix plus the time for significance weighting. For ItemCF, we measure the l-nearest neighbor (l-NN) graph construction time as the preprocessing time. We use the inverted index-based method to calculate the l-NN graph, as it is one of the fastest algorithms for constructing an exact nearest neighbor graph. Similarly, the preprocessing time of RCF consists of only the time needed to construct the k'-NN graph; we construct this graph using inverted indices. The preprocessing time of RCF + TFIDF + GF is identical to that of RCF, except it uses greedy filtering to construct the k'-NN graph. The recommendation time is the total time to produce top-N recommendations for all 6040 users, because according to Davidson et al. (2010), it is common to precompute all of the recommendation results in commercial systems.

4.2. Overall comparison

Fig. 3 shows the recall of the above mentioned algorithms while varying the number of recommended items. In this result, RCF outperforms both UserCF and ItemCF, which means that fewer rating predictions yield better results. When we apply the TF-IDF weighting scheme and use the approximate k'-NN graph with 80% accuracy instead of an exact k'-NN graph, the recall is decreased slightly, though this method still outperforms UserCF and ItemCF.

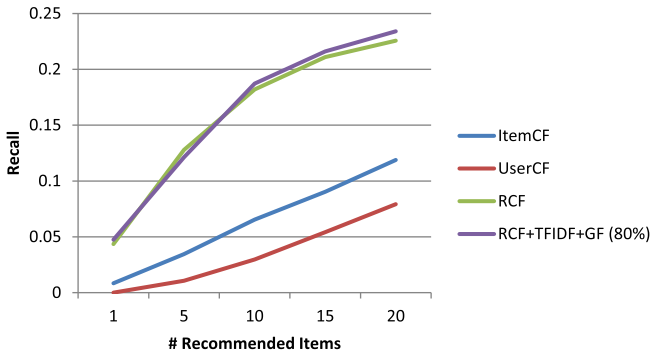


Fig. 3. Comparison of all algorithms (recall).

Table 2
Comparison of prediction accuracy with two different item sets.

Item set	Avg. sim.	Avg. MAE	Avg. RMSE
Items selected by RCF	0.1954	0.6475	0.8361
Items NOT selected by RCF	0.1300	0.7353	0.9300

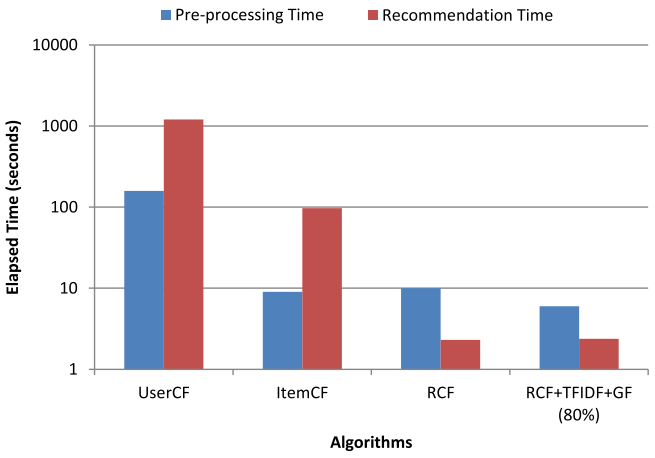


Fig. 4. Comparison of all algorithms (elapsed time).

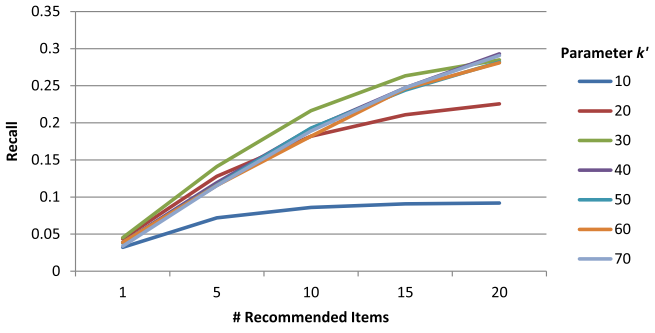


Fig. 5. Recall of RCF variants with different k' parameters.

There are two main reasons why RCF outperforms ItemCF despite the fact that RCF simulates ItemCF. First, ItemCF usually predicts the ratings of unrated items based on fewer similar items. Second, rating predictions based on less similar items are less accurate than those based on similar items. Table 2 provides evidence of these assertions. First, we divided the items into two subsets, where one subset contains unrated items whose ratings are predicted by RCF and the other subset contains other unrated items. Then, for each subset, we measured the average similarity of selected neighbors, MAE, and RMSE after executing ItemCF. The average similarity value supports the first assertion, and MAE and RMSE support the second assertion. Note MAE and RMSE of user u are defined as follows:

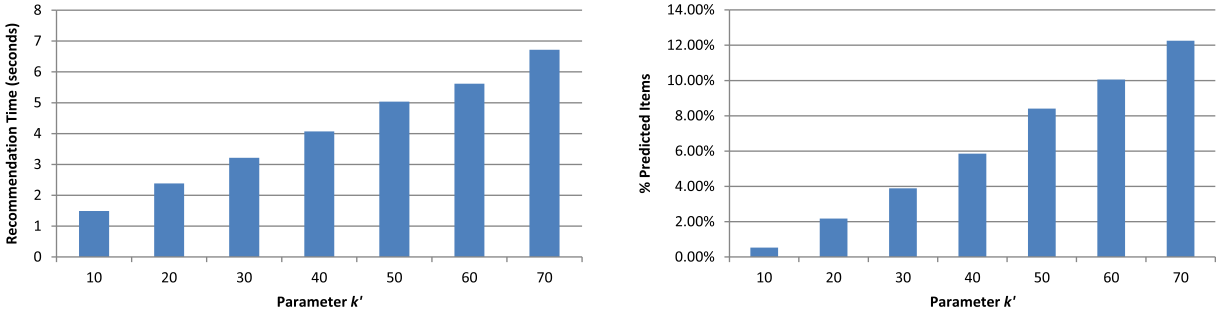
$$MAE(u) = \frac{\sum_{i \in \mathcal{I}_S} |p_{u,i} - \hat{p}_{u,i}|}{|\mathcal{I}_S|} \tag{10}$$

$$RMSE(u) = \sqrt{\frac{\sum_{i \in \mathcal{I}_S} (p_{u,i} - \hat{p}_{u,i})^2}{|\mathcal{I}_S|}}, \tag{11}$$

where $\hat{p}_{u,i}$ denotes the predicted rating of item i by u, $p_{u,i}$ denotes its corresponding actual rating, and $\mathcal{I}_S$ denotes an item set, which can be a set of either items selected by RCF or items not selected by RCF.

One limitation of RCF is that the algorithm cannot recommend many items if the parameter k' is not large enough. Because of this limitation, as shown in Fig. 3, the recall of RCF and the RCF variant does not increase significantly when N is large enough. Although Sarwar et al. (2001) and Park et al. (2013b) state that we usually need only a small number of recommendations in real-world scenarios, if there is a need for a very large number of recommendations, the performance of RCF would be similar to that of ItemCF.

Fig. 4 shows the pre-processing time and recommendation time of the above mentioned algorithms on a log scale: (1) UserCF is the slowest algorithm among these four algorithms. As a preprocessing step, this algorithm constructs a user-by-user similarity matrix and applies the significance weighting to the similarity matrix, which takes quadratic time complexity in total. It also consumes a considerable amount of recommendation time when a query is requested, because for each unrated item, it selects k users who have rated the item and predicts the rating of the item. (2) ItemCF is faster than UserCF in that it does not need to calculate a similarity matrix or complete the significance weighting step. Instead, it constructs a l-NN graph in which l greatly exceeds k. Although l is a large constant, we reduce the time to construct the graph using inverted index join, which is one of the fastest algorithms among all exact k-NN graph construction algorithms. (3) While the preprocessing time of RCF is similar to that of ItemCF, this algorithm significantly outperforms ItemCF in terms of recommendation time for two reasons. First, it does not take much time to select the neighbors of each unrated item in that it only checks the set size of each unrated item and then deletes all except for the most similar k items. Second, it calculates fewer item ratings, which dramatically decreases the recommendation time. (4) RCF + TFIDF + GF is the fastest algorithm among these four algorithms. While its recommendation time is similar to that of RCF, it outperforms RCF in terms of



(a) Elapsed time

(b) Percentage of rating predictions

Fig. 6. Effect of different k' parameters.

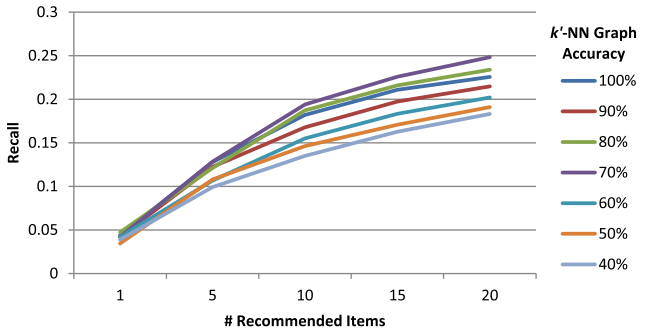


Fig. 7. Recall of RCF variants with different k'-NN graph accuracy levels.

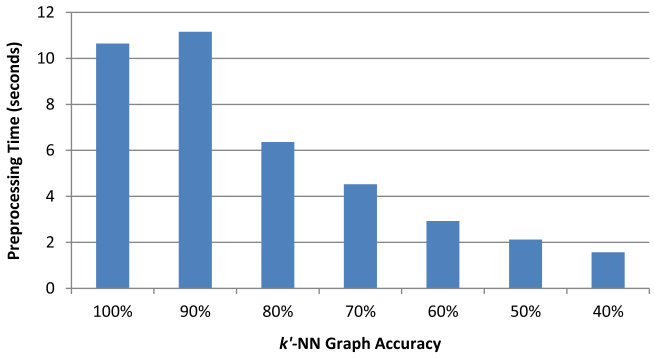


Fig. 8. Elapsed time of RCF variances with different graph accuracy levels.

preprocessing time, as it constructs an approximate k'-NN graph by means of greedy filtering. In Section 4.3, we demonstrate even faster recommendations by changing the greedy filtering parameters.

4.3. Effects of parameter changes

We identified several important factors that affect the quality and performance of the algorithms: the parameters k, k', l, and the k'-NN graph accuracy. Because the parameters k and l were analyzed in the work of Herlocker et al. (1999) and Sarwar et al. (2001), we only analyze k' and the k'-NN graph accuracy in this paper. Fig. 5 shows the recall of RCF variants with different parameter k', varying the number of recommended items. Note k' is directly related to the number of rating predictions performed by RCF. When we increase the parameter from 10 to 20 or from 20 to 30, the recommendation quality is improved because we can consider more items for the top-N recommendation. However, when we increase the parameters from 30 to 70, the recommendation quality is not improved for the reasons given in the previous subsection. Fig. 6(a) and (b) show that the parameter k' is also related to the recommendation time and the percentage of rating predictions, respectively. Because we can improve the execution time by setting k' to a low value, it would be desirable to set k' to 20 or 30.

Similarly, Figs. 7 and 8 show the recall and pre-processing time of RCF variants with different graph accuracy levels, varying the number of recommended items. There are two interesting findings in these figures: first, the recall is the highest when k'-NN graph accuracy is 70%. We can see this result because we cannot guarantee that we will always prefer the items more similar to the preferred items. Similar results are shown in the work of Chen et al. (2009), where an approximate k-NN graph is used for fast agglomerative clustering. Second, the elapsed time of RCF is the highest when k'-NN graph accuracy is 90%, because we use inverted index join instead of greedy filtering when we construct the exact k'-NN graph. Generally, however, the quality of recommendations slightly drops off when we decrease the graph accuracy, whereas the pre-processing time is significantly reduced. We can infer that these RCF variants perform even better in terms of preprocessing time as the number of nodes or dimensions scales up due to the scalability gained when using greedy filtering.

5. Conclusions

This paper presents RCF, a fast CF algorithm which utilizes a k'-NN graph. Not only does this algorithm perform fewer predictions while filtering out inaccurate results, but it also supports the rapid retrieval of similar users. The experimental results show that our approach outperforms traditional user-based/item-based CF algorithms in terms of both preprocessing time and query processing time without sacrificing the level of accuracy when we set k and k' to 10 and 20, respectively. While much of the recent work, such as Birtolo and Ronca (2013) and Lee et al. (2013), focuses on improving the recommendation quality, the main aim of our approach is to reduce the elapsed time required for recommendation.

The limitations of our approach are twofold: first, RCF is not appropriate for the case where we have to predict the ratings for all of the unrated items. In future work, we would like to present a novel algorithm for coping with this problem. Second, the performance of greedy filtering significantly depends on the dataset so that the algorithm could be slower than inverted index join in the worst case. Thus we are currently developing a novel k'-NN graph construction algorithm that guarantees high level of quality and performance.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIP) (No. 20110030812). The work of Woosung Jung was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2012R1A1A1043769).

References

Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., & Renz, M. (2006). Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proceedings of the 2006 ACM SIGMOD international conference on management of data* (pp. 515–526). ACM.

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*, 734–749.

Birtolo, C., & Ronca, D. (2013). Advances in clustering collaborative filtering by means of fuzzy c-means and trust. *Expert Systems with Applications, 40*, 6997–7009.

Chen, J., Fang, H.-r., & Saad, Y. (2009). Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection. *The Journal of Machine Learning Research, 10*, 1989–2012.

Cremonesi, P., Koren, Y., & Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on recommender systems* (pp. 39–46). ACM.

Das, A. S., Datar, M., Garg, A., & Rajaram, S. (2007). Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th international conference on world wide web* (pp. 271–280). ACM.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on computational geometry* (pp. 253–262). ACM.

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., et al. (2010). The youtube video recommendation system. In *Proceedings of the fourth ACM conference on recommender systems* (pp. 293–296). ACM.

Desrosiers, C., & Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook* (pp. 107–144). Springer.

Dong, W., Moses, C., & Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on world wide web* (pp. 577–586). ACM.

Gan, J., Feng, J., Fang, Q., & Ng, W. (2012). Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data* (pp. 541–552). ACM.

Herlocker, J. L., Konstan, J. A., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 230–237). ACM.

Kim, Y., & Shim, K. (2012). Parallel top-k similarity join algorithms using mapreduce. In *IEEE 28th international conference on data engineering (ICDE), 2012* (pp. 510–521). IEEE.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer, 42*, 30–37.

Lee, S., Park, S., Kahng, M., & Lee, S.-g. (2013). Pathrank: Ranking nodes on a heterogeneous graph for flexible hybrid recommender systems. *Expert Systems with Applications, 40*, 684–697.

Lee, D., Park, S. E., Kahng, M., Lee, S., & Lee, S.-g. (2010b). Exploiting contextual information from event logs for personalized recommendation. In *Computer and information science 2010* (pp. 121–139). Springer.

Lee, D., Park, J., Shim, J., & Lee, S.-g. (2010a). An efficient similarity join algorithm with cosine similarity predicate. In *Database and Expert Systems Applications* (pp. 422–436). Springer.

Lee, S., Song, S.-i., Kahng, M., Lee, D., & Lee, S.-g. (2011). Random walk based entity ranking on graph for multidimensional recommendation. In *Proceedings of the fifth ACM conference on recommender systems* (pp. 93–100). ACM.

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing, 7*, 76–80.

Park, Y., Yang, B., Song, S., Lee, S., & Lee, S.- g. (2013b). Context-aware recommendation in smart cars. In *Proceedings of the 6th biennial workshop on digital signal processing for in-vehicle systems* (pp. 64–67).

Park, Y., Park, S., Lee, S.- g., & Jung, W. (2014a). Fast collaborative filtering with a k-nearest neighbor graph. In *Proceedings of the international conference on big data and smart computing* (pp. 92–95).

Park, S. E., Lee, S., & Lee, S.-g. (2011). Session-based collaborative filtering for predicting the next song. In *Proceedings of first ACIS/JNU international conference on computers, networks, systems and industrial engineering* (pp. 353–358). IEEE.

Park, Y., Park, S., Lee, S.-g., & Jung, W. (2013a). Scalable k-nearest neighbor graph construction based on greedy filtering. In *Proceedings of the 22nd international conference on world wide web companion* (pp. 227–228). International World Wide Web Conferences Steering Committee.

Park, Y., Park, S., Lee, S.-g., & Jung, W. (2014b). Greedy filtering: A scalable algorithm for k-nearest neighbor graph construction. In *Database systems for advanced applications* (pp. 327–341). Springer.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on world wide web* (pp. 285–295). ACM.

Xiao, C., Wang, W., Lin, X., & Shang, H. (2009). Top-k set similarity joins. In *IEEE 25th international conference on data engineering, 2009. ICDE'09* (pp. 916–927). IEEE.