# A New Approach for Collaborative Filtering Based on Mining Frequent Itemsets

Phung Do[1], Vu Thanh Nguyen[1], and Tran Nam Dung[2]

[1] University of Information Technology, Ho Chi Minh City, Vietnam
{phungdtm,nguyenvt}@uit.edu.vn
[2] University of Natural Science, Ho Chi Minh City, Vietnam
trannamdung@yahoo.com

**Abstract.** As one of the most successful approaches to building recommender systems, collaborative filtering (*CF*) uses the known preferences of a group of users to make recommendations or predictions of the unknown preferences for other users. In this paper, we first propose a new CF model-based approach which has been implemented by basing on mining frequent itemsets technique with the assumption that "*The larger the support of an item is, the higher it's likely that this item will occur in some frequent itemset, is*". We then present the enhanced techniques such as the followings: bits representations, bits matching as well bits mining in order to speeding-up the algorithm processing with CF method.

**Keywords:** Collaborative Filtering, mining frequent itemsets, bit matching, bit mining.

## 1 Introduction

As one of the most successful approaches to building recommender systems, collaborative filtering (CF) uses the known preferences of a group of users to make recommendations or predictions of the unknown preferences for other users. There are three main categories of CF techniques: memory-based, model-based, and hybrid CF algorithms (that combine CF with other recommendation techniques). The memory-based CF methods are deployed into commercial systems such as http://www.amazon.com/ and Barnes and Noble, because they are easy-to-implement and highly effective [4, 5]. The main drawback of memory-based methods are the requirement of loading a large amount of in-line memory. Well-known memory-based CF techniques include neighbor-based CF (item-based/user-based CF algorithms with Pearson/vector cosine correlation) [6, 13] and item-based/user-based top-N recommendations [14]. The problem is serious when rating matrix becomes so huge in situation that there are extremely many persons using system. Computational resource is consumed much and system performance goes down; so system can't respond user require immediately. Model-based CF techniques use the pure rating data to estimate or learn a model to make predictions [6]. The model can be a data mining or machine learning algorithm. Well-known model-based CF techniques include Bayesian belief nets (BNs) CF models [6–8], clustering CF models [9, 10], latent semantic CF models [5], MDP (Markov decision process) - based

CF [11] and CF using dimensionality reduction techniques, for example, SVD, PCA [12]. Model-based CF achieved real-time response when inference speed much faster than calculated on the entire data in memory, but the time for building the model is slowly. In this paper, we proposed a model-based CF approach based on mining frequent itemsets and bit matching technique in order to speeding-up the algorithm processing with CF methods. In addition, the proposed approach is to increase the usefulness of recommendations to present those items that user interest by discovering the user's purchasing patterns. In section *2* we propose an idea for the model-based CF algorithm based on mining frequent itemsets. The heuristic algorithm is discussed carefully in the section *3*. We propose an abstract architecture along with a framework which assists them in implementing and evaluating CF algorithm in the section *4*. Section *5* is the evaluation. Section *6* is the conclusion. Note that terms such as "rating matrix", "dataset" and "database" have the same meaning in this paper.

## 2 A New CF Algorithm Based on Mining Frequent Itemsets

With the following given rating vectors, for instance, *u* = (*item 1 = 3, item 2 = 5, item 3* = *2*). It means that user *u* rated on *item 1, item 2, item 3*. Where, their values are *3*, *5* and *2*, respectively. Then the concept of creating this new CF algorithm, based on mining frequent itemsets is to consisting of two following processing steps:

- Modeling process: A set of frequent itemsets *S* is mined and it is performed in offline process mode.
- Recommendation process: whenever user *u* requires to get recommended items, a frequent itemset *s* is chosen from *S* so that *s* contains items *1*, *2* and *3*, for instance, *s* = (*item 1, item 2, item 3, item 5, item 7*). The additional items *5* and *7* are then recommended to user. This meant recommendation isn't like the modeling process; instead it's an on-line process.

Although the modeling process does consume much more time than that of the recommendation one. But, it is executed in offline mode. Therefore it won't be causing any negative-time consuming impact on recommendation process. However, there would be a serious problem that could be raised, when the frequent itemset *s* = (*item 1, item 2, item 3, item 5, item 7*) didn't give any indication that which rating values items *1, 2* , and *3* have been assigned. It is obvious to know that items *1, 2* and *3* are rated by the values of *3, 5* and *2*, respectively in rating vector *u*. This means the rating vector *u* and the frequent itemset *s* don't match exactly. This eventually causes another hazard which is impossible when launching an attempt to compute predictive values, with missing ratings for rating vector. Now please pay attention, this problem will be eliminated or solved by using the technique so-called *bit transformation*. Note that the terms "*bit*" and "*binary*" have the same meaning.

For instance, a rating matrix, where its rows indicate users, its columns indicate items and each cell is the rating which user has given to item. With the ratings are in a range from *1* to *5* {*1…5*}, then, the sample rating matrix is shown as what will be seen in *Table 1.* The value 5 indicates the most preference.

Each item is "stretched" into *5* sub-items which are respective to *5* possible rating values {*1…5*}. Each sub-item is symbolized as *item_j_k* carrying two binary-states *1* and *0*, which indicates whether user rates on item *j* with concrete value *k*. For example, the bit *item_2_5* getting state *1* shows that user gave rating value *5* on item *2*. Now the rating matrix is transformed into bit rating matrix in which each cell is the rating of bit sub-item. Supposing that, empty cell has shown such cell get valued by *0*; it means that there is no one to give a rating on the cell yet. The bit rating matrix is shown in *Table 2*.

**Table 1.** Rating matrix table

| | Item 1 | Item 2 | Item 3 | Item 4 |
|---|---|---|---|---|
| User 1 | *3* | *5* | *2* | *1* |
| User 2 | *3* | *5* | *2* | *1* |
| User 3 | *1* | *5* | *4* | |

**Table 2.** Bit rating matrix table

| | User 1 | User 2 | User 3 |
|---|---|---|---|
| Item_1_1 | *0* | *0* | *1* |
| Item_1_3 | *1* | *1* | *0* |
| Item_2_5 | *1* | *1* | *1* |
| Item_3_2 | *1* | *1* | *0* |
| Item_3_4 | *0* | *0* | *1* |
| Item_4_1 | *1* | *1* | *0* |

Each frequent itemset, that has been extracted from bit rating matrix and, it will carry a so-called bit form, $s$ = (*item_j*$_1$*_k*$_1$*, item_j*$_2$*_k*$_2$*,…*). Where, each component *item_j_k*, has been defined as bit sub-item. After that, rating vector $u$ is also to be transformed into bit rating vector $u$ = (*item_j*$_1$*_k*$_1$*, item_j*$_2$*_k*$_2$*,…*). It's so easy to find that matching the twos, bit frequent itemset, to bit rating vector is completely simple. For instance, if using the shown previous examples; where, the rating vector $u$ = (*item1 = 3, item 2 = 5, item 3 = 2*) is to be transformed into $u$ = (*item_1_3, item_2_5, item_3_2*). While the frequent itemsets are $s_1$ = (*item_1_3, item_2_5, item_3_2, item_4_1*) and $s_2$ = (*item_1_1, item_2_5, item_3_4*). We also find that itemset $s_1$, has been matched at the most, to $u$ and so, the item *4* is recommended to user with predictive value *1*.

Now the previous mentioned problem is solved but our algorithm should be enhanced for a little more better. Suppose that the number of frequent itemsets is huge and even each itemset has also a lot of items. When we are to match the rating vector and frequent itemset, there will be a boom of combinations that may cause computer system collapsed or consumed an even a whole lot more of processing time. Therefore, we propose an enhancement method for matching purpose, based on the technique, called *bit matching*.

## 2.1 Bit Representation and Bit Matching

Suppose there are *4* items and each item has *5* possible rating values, we use the bit set whose length is *4 * 5 = 20* bits (so-called *20*-length bit set) to represent rating vectors and frequent itemsets. The bit set is divided into many clusters or groups, for example, if each item has *5* possible rating values then each cluster has *5* bits. So each cluster represents a sub-item and the position of a bit in its cluster indicates the rating value of corresponding sub-item. If a cluster contains a bit which is set, its corresponding sub-item is rated with the value which is the position of such set bit. Following is an example of bit set:

**Table 3.** Bit representation

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Cluster 1 (item 1 = 3) | | | | | Cluster 2 (item 2 = 5) | | | | | Cluster 3 (item 3 = 2) | | | | | Cluster 4 | | | | |

For example, rating vector $u$ = (*item1 = 3, item 2 = 5, item 3 = 2*) is transformed into $u$ = (*item_1_3, item_2_5, item_3_2*) which is represented as $u$ = (*00100 00001 01000 00000*) having four clusters. The frequent itemset $s_1$ = (*item_1_3, item_2_5, item_3_2, item_4_1*) which is represented as $s_1$ = (*00100 00001 01000 10000*). The frequent itemset $s_2$ = (*item_1_1, item_2_5, item_3_4*) which is represented as $s_2$ = (*10000 00001 00010 00000*). In order to match $s_1$ (or $s_2$) with $u$, all we need is to do AND bit-operation between $s_1$ (or $s_2$) and $u$.

- If $s_1$ *AND* $u = u$ then $s_1$ matches with $u$
- If $s_1$ *AND* $u \neq u$ then $s_1$ doesn't match with $u$

When $s_1$ get matched with $u$, we do *AND – NOT* operation, as to extract items which are recommended to users. Suppose, the recommended item is denoted *r_item*:

*r_item* = $s_1$ *AND* (*NOT u*) = (*00000 00000 00000 10000*)

From this bit set, it is easy to recognize that item *4* is recommended with predict value is *1* because the first bit of $4^{th}$ cluster is set.

As a result, our algorithm will consist of 3 steps:

- Step 1: Rating matrix is transformed into bit rating matrix.
- Step 2: Bit rating matrix is mined, as well as to extract frequent itemsets.
- Step 3: Rating vector and frequent itemsets are represented as bit sets. Bit matching operations are performed in order to find out the appropriate frequent itemset which is matched with rating vector. Basing on such frequent itemset, it is possible to determine which items are recommended. Moreover missing values of recommended items can be also predicted.

## 2.2 Pseudo-code Like C for New CF Algorithm

Let *D, B, S* be rating matrix, bit rating matrix and the set of frequent itemsets, respectively. Let *matched_itemset* and *r_item* be matched itemset and recommended item, respectively. Let *bitset*(…), *count*(…) be functions that transforms item into bit set and counts the number of bit *1* (s) in bit set. Let *bit_transform* be the function which transforms rating matrix into bit rating matrix. Let *mining_frequent_itemset* be the mining function which extracts frequent itemsets from bit rating matrix (see *sections 3.1, 3.2*). Following is the pseudo-code like C for our CF algorithm:

```
B = bit_transform(D)
S = mining_frequent_itemset(B)
matched_itemset = null
max_count = -1
For each s ∈ S
   bs = bitset(u) AND bitset(s)
   If bs = bitset(u) && count(bs) > max_count then
      matched_itemset = s
      max_count = count(bs)
   End If
End For
r_item = bitset(matched_itemset) AND (NOT bitset(u))
```

The second step is the most important, since it's very often, ones are to asking that. Such as, there is a question: "How frequent itemsets are extracted from rating matrix". This question is, then answered in the next section about mining frequent itemsets.

## 3 Mining Frequent Itemsets

Our mining frequent itemsets method is based on the assumption: "*The larger the support of an item is, the higher it's likely that, this item occurs in some itemset*". In other words, items with the high support tend to combine together so as to form a frequent itemset. So our method is the heuristic algorithm so-called *Roller* algorithm. The basic idea is similar to that of a white-wash task. Suppose you imagine that there is a wall and there is the dataset (namely, rating matrix) containing all items. Such dataset is modeled as this wall. On the wall, all items are shown in a descending ordering of their supports; it means that the higher frequent item is followed by the lower frequent item. Moreover, we have a roller and we roll it on the wall, from item to item, with respect to the descending ordering. If an item is found, satisfied at a minimum support (*min_sup*), it is, then added to the frequent itemset and the rolling task is continued to keep moving on, until there is no item that meets minimum support. The next time, all items in this frequent itemset are removed from the meant wall and the next rolling task will be performed to find out new frequent itemset.

Our algorithm includes four following steps:

- <u>Step 1</u>: Computing the supports of all items and arranging these items on the wall, according to the descending ordering of their supports. Note that all items whose supports don't meet minimum support are removed from this descending ordering. The kept items are called the frequent items.
- <u>Step 2</u>: The $i^{th}$ itemset is initialized by the first item in this descending ordering. The support of $i^{th}$ itemset is initialized as the support of this first item. The current item now is the first item and it is removed from descending ordering.
- <u>Step 3</u>: If there is no item in descending ordering, the algorithm will be terminated. Otherwise:
  - <u>3.1</u>. If the current item is the last one, in descending ordering, then all items in the $i^{th}$ itemset are removed from the descending ordering and the number $i$ is increased by $1$ ($i = i + 1$). Go to step 2.
  - <u>3.2</u>. If the current item is NOT the last in descending ordering, then, the next item is picked and so the current item now is the next item.

- Step 4: Checking the support of current item:
  - The support of current item satisfies *min_sup*: the support of the $i^{th}$ itemset *support*($i^{th}$ *itemset*) is accumulated by current item; it is the count of total transactions that contains all items in both the $i^{th}$ itemset and current item. If *support*($i^{th}$ *itemset*) is equal to or larger than *min_sup*, this item is added to the $i^{th}$ itemset.
  - Go back step *3*.

It is easy to recognize that step *3* and *4* are similar to that of a white-wash task which "rolls" the $i^{th}$ itemset modeled as the roller. After each rolling (each iteration), such itemset get thicker with more items.

Let $I = (i_1, i_2,..., i_m)$ and $S$ be a set of item and a set of frequent itemset, respectively. Let $O = (o_1, o_2,..., o_n)$ be the list of items whose supports are sorted according to descending ordering, $O \subseteq I$. Let $s_i$ be the $i^{th}$ itemset. Let $c$ be the current item. Let *support*(…) be the function calculating the support of item or itemset. Let *sort*(…), *first*(…), *next*(…), *last*(…) be sorting, getting first item, getting next item, getting last item functions, respectively. Following is the pseudo-code like C for Roller algorithm (function *mining_frequent_itemset*):

```
O = sort(I)
i = 1
While (true)
   c = first(O)
   $s_i$ = $s_i$ ∪ {c}
   O = O / {c}
   If O = ∅ then return S
   While (true)
      If c = last(O) then
         S = S ∪ $s_i$
         O = O / S
         i = i +1
         break
      Else
         c = next(O, c)
         If support(c) < min_sup continue
         b = bitset(S) AND bitset(c)
         If count(b) ≥ min_sup then
            $s_i$ = $s_i$ ∪ {c}
         End If
      End If
   End While
End While
```

Although the Roller algorithm may ignore some frequent itemsets but it runs much faster than traditional mining frequent itemsets methods. Especially our algorithm can be enhanced by using a so-called technique of bit mining.

## 3.1 Bit Mining

When rating matrix (dataset) is transformed into bit rating matrix, item and itemset become cluster (sub-item) and bit set (see *section 2*). The support of item or itemset are the number of bits whose values are 1 (*s*) in bit set. Given bit rating matrix as above table (*see table 2*). In step *1*, sub-items are sorted according to descending ordering and some sub-items not satisfying *min_sup* are removed given the *min_sup* is *2*. Now sub-items are represented as bit cluster: *Item_2_5* = (*111*), *Item_1_3* = (*110*), *Item_3_2* = (*110*), *Item_4_1* = (*110*).

In step *2*, the first itemset $s_1$ is initialized as *Item_2_5*

$s_1$ = (*111*) and *support*($s_1$) = *count* (111) = *3*

Where *count* (…) indicates the number of bits whose values are *1* (s) in bit set (…).

In step *3* and *4*, sub-items (clusters) such as *Item_1_3, Item_3_2, Item_4_1* are picked in turn and all of them satisfy *min_sup*.

- Picking *Item_1_3*: $s_1$ = $s_1$ *AND Item_1_3*=(*111*) *AND* (*110*) = (*110*) $\rightarrow$ *support*($s_1$) = *2*.
- Picking *Item_3_2*: $s_1$ = $s_1$ *AND Item_3_2* = (*110*) *AND* (*110*) = (*110*) $\rightarrow$ *support*($s_1$) = *2*.
- Picking *Item_4_1*: $s_1$ = $s_1$ *AND Item_4_1* = (*110*) *AND* (*110*) = (*110*) $\rightarrow$ *support*($s_1$) = *2*.

Finally, the frequent itemset is $s_1$ = (*110*) which include *Item_2_5, Item_1_3, Item_3_2, Item_4_1*. We recognize that the bit set of frequent itemset, named $s_1$ is accumulated by frequent item after each iteration. This make algorithm runs faster. The cost of counting bit set and performing bit operations isn't significant.

## 3.2 The Improvement of Roller Algorithm

Roller algorithm may lose some frequent itemsets because there is a case in that some frequent items don't have so high a support (they are not excellent items) and they are in the last of descending ordering. So they don't have many chances to join to frequent itemsets. However they really contribute themselves into some frequent itemset because they can combine together to build up frequent itemset, but they don't make the support of such itemset decreased much. It is difficult to discover their usefulness. In order to overcome this drawback, the Roller algorithm is modified so that such useful items are not ignored.

So in step *3*, instead of choosing the next item as the current item, we can look up an item whose support is *pseudo-maximum* and choose such item as the current item. In step *3.2*, if the current item is NOT the last in descending ordering, we look up the item which is combined (AND operation) with $i^{th}$ itemset so as to form the new itemset whose support is maximum. Such item as being the so-called *pseudo-maximum support item* is chosen as the current item.

The improved Roller algorithm take slightly more time than normal Roller algorithm for looking up *pseudo-maximum support item* in step *3* but it can discover more frequent itemsets. So its accuracy is higher than normal Roller algorithm.

## 4 Algorithm Implementation in General Architecture

The new CF is implemented and evaluated according to the general architecture interpreted by UML language has 4 basic interfaces and classes: *Algorithm*, *KBase*, *Dataset* and *Evaluator*. Such interfaces and classes are considered as the software-engineering standard for CF algorithm. Researcher will conform to such standard when they apply this framework into writing a new algorithm.

- Interface *Algorithm* represents abstract algorithm. The main task that researchers does is to realize this interface according to their goals when they invent a new algorithm. In most cases, they implement directly two classes *MemoryBasedCF* and *ModelBasedCF* which are derived from *Algorithm*. *MemoryBasedCF* and *ModelBasedCF* represent memory-based *CF* algorithm and model-based *CF* algorithm, respectively.
- Interface *KBase* represents knowledge base which associates with a model-based algorithm *ModelBasedCF*. The structure of *KBase* is very flexible and it depends on ideas and purposes of algorithm.
- Class *Dataset* is composed of a rating matrix and personal profiles. Each row of rating matrix is represented by class *RatingVector*. Personal profile is represented by class *Profile*. Framework is responsible for manipulating *Dataset*.
- Class *Evaluator* is used by framework in order to evaluate algorithm according to criterions so-called *measures* such as time, precision and recall. Such measures are defined inside *Evaluator*. *Evaluator* reads and feeds dataset on algorithm *Algorithm*. Finally, it evaluates such algorithm by calculating *Measures* based on result of executing algorithm.

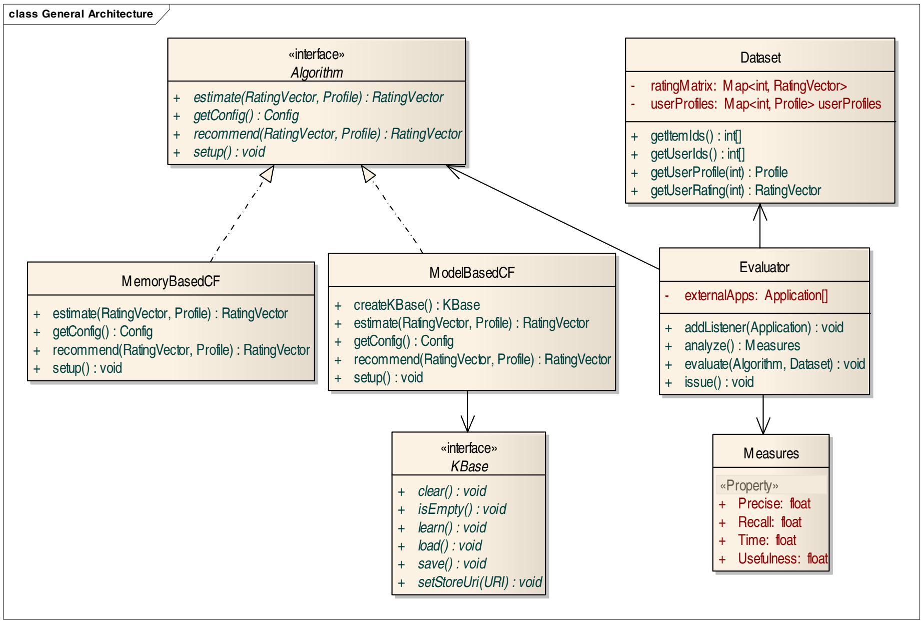
class General Architecture
«interface» Algorithm
+ estimate(RatingVector, Profile) : RatingVector
+ getConfig() : Config
+ recommend(RatingVector, Profile) : RatingVector
+ setup() : void
Dataset
- ratingMatrix: Map<int, RatingVector>
- userProfiles: Map<int, Profile> userProfiles
+ getItemIds() : int[]
+ getUserIds() : int[]
+ getUserProfile(int) : Profile
+ getUserRating(int) : RatingVector
MemoryBasedCF
+ estimate(RatingVector, Profile) : RatingVector
+ getConfig() : Config
+ recommend(RatingVector, Profile) : RatingVector
+ setup() : void
ModelBasedCF
+ createKBase() : KBase
+ estimate(RatingVector, Profile) : RatingVector
+ getConfig() : Config
+ recommend(RatingVector, Profile) : RatingVector
+ setup() : void
Evaluator
- externalApps: Application[]
+ addListener(Application) : void
+ analyze() : Measures
+ evaluate(Algorithm, Dataset) : void
+ issue() : void
«interface» KBase
+ clear() : void
+ isEmpty() : void
+ learn() : void
+ load() : void
+ save() : void
+ setStoreUri(URI) : void
Measures
«Property»
+ Precise: float
+ Recall: float
+ Time: float
+ Usefulness: float

**Fig. 1.** General architecture for CF algorithms

The CF algorithm in this paper is realized as following steps:

- Interface *Algorithm* is impemented according to the goal and schema of this CF
- Maximum frequent itemsets are represented as *KBase*. So Roller algorithm which finds frequent itemsets is implemented as method *KBase::learn*().
- This CF is executed by the *Evaluator* in such framework
- The evaluation measures on this CF are computed by such framework after this execution.

## 5 Evaluation

Database *Movielens* [1] including *100,000* ratings of *943* users on *1682* movies is used for evaluation. Database is divided into 5 folders, each folder includes one training set over 80% whole database and one testing set over 20% whole database. Training set and testing set in the same folder are disjoint sets.

**Table 4.** Evaluation result

| | **Our method** | **Cosine** | **Pearson** | **Simple** |
|---|---|---|---|---|
| MAE | 0.21459 | 0.37278 | 0.45747 | 0.33391 |
| MSE | 0.10285 | 0.23887 | 0.31175 | 0.24616 |
| RMSE | 0.32041 | 0.48485 | 0.51458 | 0.47705 |
| Precision | 0.10554 | 0.00064 | 0.00046 | 0.00077 |
| Recall | 0.04042 | 0.00024 | 0.00018 | 0.00028 |
| F1 | 0.05758 | 0.00035 | 0.00026 | 0.0004 |
| Spearman | 0.12540 | -1 | 0 | 0 |
| Time | 0.00491 | 1.2117 | 2.10073 | 0.11006 |

The system setting includes: Processor Pentium(R) Dual-Core CPU E5700 @ 3.00GHz, RAM 2GB, Available RAM 1GB, Microsoft Windows 7 Ultimate 2009 32-bit, Java 7 HotSpot (TM) Client VM. Our CF method is compared to three other methods: *simple method* – simplest memory-based CF algorithm, c*osine method* – memory-based CF algorithm in which the cosine measure is used, *Pearson method* – memory-based CF algorithm in which the Pearson measure is used and Green Fall method – model-based CF using mining frequent itemsets technique.

There are 8 metrics used in this evaluation: *MAE, MSE, RMSE, recall, precision, F1, Spearman correlation* [4] and *time*. Note that all metrics except time metric are normalized in range [0, 1] and time metric is calculated in seconds. Table 4 is the evaluation result.

Our method is much better than cosine, Pearson, simple methods in all aspects: less error ratio via *MAE*, *MSE* and *RMSE* metrics; more accuracy via recall, precision and

F1 metrics; higher correlation via Spearman metric. The best thing is that it runs much faster than other methods.

## 6 Conclusion

Our CF approach is different from other model-based CF methods when trying to discover user interests. The mining technique is important for extracting frequent itemsets considered as patterns of user interests. However traditional mining algorithms consume much more time and resources. So we proposed a new mining method, a so-called Roller algorithm. Based on evaluation measures, Roller is proved as reliable algorithm with high performance, fast speed, high usefulness and consuming less time and resources. Its sole drawback is that it may ignore some user patterns because of heuristic assumption. However this drawback is alleviated by taking advantage of enhancement techniques such as bit mining, the concept of *pseudo-maximum support*.

In the future, we will propose another new model-based CF method which uses Bayesian network in inferring user interests. Such method based on statistical mechanism will be compared to the method in this paper so that we have an open and objective viewpoint about mining technique and statistical technique.

## References

1. Movielens dataset 2011. Home page is `http://www.movielens.org`, Download dataset from `http://www.grouplens.org/node/12`
2. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 2nd edn. Elsevier Inc. (2006)
3. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems 22(1), 5–53 (2004)
4. Linden, G., Smith, B., York, J.: Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Computing 7(1), 76–80 (2003)
5. Hofmann, T.: Latent semantic models for collaborative filtering. ACM Transactions on Information Systems 22(1), 89–115 (2004)
6. Breese, J., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, UAI 1998 (1998)
7. Miyahara, K., Pazzani, M.J.: Collaborative filtering withthe simple Bayesian classifier. In: Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence, pp. 679–689 (2000)
8. Su, X., Khoshgoftaar, T.M.: Collaborative filtering for multi-class data using belief nets algorithms. In: Proceedings of the International Conference on Tools with Artificial Intelligence, ICTAI 2006, pp. 497–504 (2006)
9. Ungar, L.H., Foster, D.P.: Clustering methods for collaborative filtering. In: Proceedings of the Workshop on Recommendation Systems. AAAI Press (1998)

10. Chee, S.H.S., Han, J., Wang, K.: RecTree: an efficient collaborative filtering method. In: Proceedings of the 3rd International Conference on Data Warehousingand Knowledge Discovery, pp. 141–151 (2001)
11. Shani, G., Heckerman, D., Brafman, R.I.: An MDP-based recommender system. Journal of Machine Learning Research 6, 1265–1295 (2005)
12. Billsus, D., Pazzani, M.: Learning collaborative information filters. In: Proceedings of the 15th International Conference on Machine Learning, ICML 1998 (1998)
13. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.: Analysis of recommendation algorithms for E-commerce. In: Proceedings of the ACM E-Commerce, Minneapolis, Minn, USA, pp. 158–167 (2000)
14. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, WWW 2001, pp. 285–295 (May 2001)