A New Approach for Collaborative Filtering based on Mining Frequent Itemsets

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

[1]University of Information Technology, Ho Chi Minh City, Vietnam
phungdtm@uit.edu.vn, 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

The recommendation system is a system which recommends to the users, all the items which are those among a large number of existing items in database. Items which are to point to anything that user are to considering such as products, books, news papers, etc. And there has been also an expectation that the recommended items will be likely the ones that the users would be like the most. Another words, such mentioned items are going to go along with the users' interests.

By that meaning, there are two recommendation systems, found to be with a common trend: content-base filtering (CBF) and collaborative filtering (CF) [2]:

- The CBF recommends an item to a user if such item is similar to other items that he like most in the past (and his rating for such item is high).
- The CF on the other hands, recommends an items to user if his neighbors (mean the other users that are identical to him) are interested in such item. It is also called memory-based CF.

Both of the above mentioned filtering (CBF & CF) do have their own strong, as well weak points. In order to alleviate this weakness of the CF type, there have been two techniques which could be helpful, used for improvements:

- The combinations of the CF and CBF types. This technique is breaking into two stages. First, it applies CBF to setting up a complete rating matrix, then the next step would be the CF type, which is used to making predictions for recommendations.
- Compressing the rating matrix into a representative model, which then is used to predict all the missing data for recommendations. This is a model-based for the CF type.

This paper focuses on model-based approach for CF based on mining frequent itemsets. It is the potential approach because it can take advantage of data mining. 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*. Section *4* is the evaluation. Section *5* 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, $s = (item\_j_1\_k_1, item\_j_2\_k_2, \ldots)$. 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 $u = (item\_j_1\_k_1, item\_j_2\_k_2, \ldots)$. 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 corres-

ponding 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:

- Step 1: 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.
- Step 2: 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.
- Step 3: If there is no item in descending ordering, the algorithm will be terminated. Otherwise:
  - 3.1. 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.
  - 3.2. 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.

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*) → *support*(s_1) = *2*.
- Picking *Item_3_2*: $s_1 = s_1$ *AND Item_3_2* = (*110*) *AND* (*110*) = (*110*) → *support*(s_1) = *2*.
- Picking *Item_4_1*: $s_1 = s_1$ *AND Item_4_1* = (*110*) *AND* (*110*) = (*110*) → *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 Evaluation

There are four measures to evaluate our CF algorithm: time (*T*), precise (*P*), recall (*R*) and usefulness (*U*). We use database *Movielens* [1] for evaluation. Database Movielens has two versions: version *100*K with *1.88* MB capacity including *100,000* ratings of *943* users on *1682* movies, version *1*M with *23.4* MB capacity including *1,000,209* ratings of *6040* users on *3900* movies. Our CF method is compared to the item-based/ user-based CF algorithms with simple/ Pearson/ vector cosine correlation.

4.1 Time Measure

The time measure *T* tells us how fast CF method responses to the requirement of system for recommendation task. This measure represents the speed of algorithm and so it is the time in seconds which algorithm runs over rating database (rating matrix). Let *D* be the rating matrix (rating database) and let $u_i \in D$ be the rating vector as each row in *D*. Let t_i be the time in seconds that the algorithm runs for row u_i. So the measure *T* is the sum of all t_i *(s)* when iterating over *D*.

$$T = \sum_i t_i$$

Our method is *35 – 317* times faster than other methods. The best thing is that it is almost independent from the size of database. The evaluation table indicates that our method takes 5 times more when dataset changes from *100K* to *1M* while other methods take *14 - 44* times more.

Table 4. Time evaluation (in seconds)

	100K	1M
Simple	0.0762	1.0953
Cosine	0.0773	3.4249
Pearson	0.0787	3.3119
Our method	0.0022	0.0108

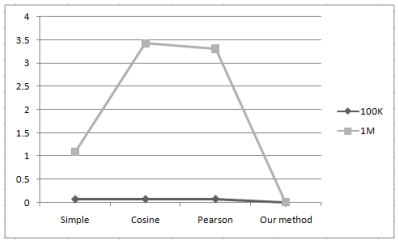

Fig. 1. Time evaluation (in seconds)

4.2 Precise Measure

Basically, the precise measure (P) is the inverse of normalized mean of error (E). In consequences, the error E is calculated as the mean of deviation. Let $u_i \in D$ and v_i be the actual rating vector and the rating vector recommended by our CF algorithm, respectively. Each u_i actually is a row in rating matrix. Each u_i has respective v_i. Suppose $u_i = (r_{i1}, r_{i2}, \ldots, r_{in})$ and $v_i = (r'_{i1}, r'_{i2}, \ldots, r'_{in})$, the basic idea is modeled by following formulation:

$$E = \frac{1}{|D|}\sum_{u_i \in D} error(u_i, v_i) = \frac{1}{|D|}\sum_{u_i \in D}\sqrt{\sum_j (r_{ij} - r'_{ij})^2}$$

Where $|D|$ is the size of database and $error(u_i, v_i)$ is the deviation between u_i and v_i.

We recognized that the error E is the ratio of total deviation between predictive rating values and actual rating values to the size of database, and known as the number of rows of rating matrix.

The rating matrix D is browsed row by row, each row u_i is a rating vector. The rating vector v_i is derived from u_i by removing randomly some values from u_i. For example, if $u_i = (1, 2, \mathbf{4}, \mathbf{5})$ then the derived $v_i = (1, 5, \mathbf{?}, \mathbf{?})$, the third component and fourth component are removed. Missing values (?) in vector v_i are predicted by CF method, for example $v_i = (1, 5, \mathbf{1}, \mathbf{1})$. After that the error is the deviation between u_i and v_i. Let k be the number of missing values, in this example, $k = 2$.

$$error(u_i, v_i) = \sqrt{\sum_{j=1}^{k} (r_{ik} - r'_{ik})^2} = \sqrt{(4-1)^2 + (5-1)^2} = 5$$

Suppose the range of rating value is $\{min \ldots max\}$, so min and max are the possible minimum and maximum of each rating. In this example, $min = 1$ and $max = 5$. The normalized error is computed as below:

$$normalized\_error(u_i, v_i) = \frac{\sqrt{\sum_{j=1}^{k} (r_{ik} - r'_{ik})^2}}{(max - \min)\sqrt{k}} = \frac{\sqrt{(4-1)^2 + (5-1)^2}}{(5-1)\sqrt{2}} = 0.88$$

The sum of errors is computed as the total error over database: $sum\ of\ error = \sum_{u_i \in D} error(u_i, v_i)$. The mean of error: $mean\_of\_error = \frac{1}{|D|}\sum_{u_i \in D} error(u_i, v_i)$. The normalized mean of error: $normalized\_mean\_of\_error = \frac{1}{|D|}\sum_{u_i \in D} normalized\_error(u_i, v_i)$ .It is written again:

$$normalized\_mean\_of\_error = \frac{1}{|D|(max - min)\sqrt{k}}\sum_{u_i \in D}\sqrt{\sum_{j=1}^{k} (r_{ik} - r'_{ik})^2}$$

So error E is the normalized mean of error. The less this error is, the more precise CF method is. So precise measure is the inverse of normalized mean of error in percentage:

Precise (P) = (*1 – normalized_mean_of_error*) * *100*

Although our method gets less precise measure, it runs faster and more stable than other methods. Its precision (*75.29% – 76.37%*) doesn't change when the size of

dataset increases. On the other hand, it runs *35 – 317* times faster than other methods but its precision is approximate to theirs. Moreover it can tolerate sparse matrix. Following is the line chart of precise evaluation:

Table 5. Precise evaluation (%)

	100K	1M
Simple	79.19%	80.19%
Cosine	78.42%	79.88%
Pearson	77.85%	79.76%
Our method	75.29%	76.37%

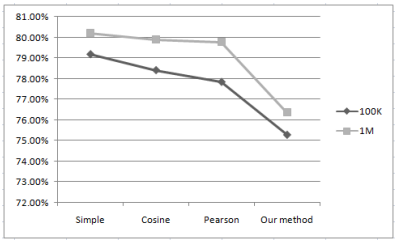

Fig. 2. Precise evaluation (%)

4.3 Recall Measure

The recall measure is the ratio of the number of recommended items to the number of total potential items. For example, there are *100* items in dataset which can recommend to users but algorithm only gives *90* items to users. The number total potential items is 100 and the number of recommended items is *90*, so the recommend recall is *90%.*

Recommend Recall = (*Recommended items / Potential items*) * *100*

Comment that all memory-based CF (s) get 100% recommend recall because they computed directly on dataset and don't build up any model and there is no loss of items. Following is the line chart of recommend recall evaluation:

Table 6. Recommend recall evaluation

	100K	1M
Simple	100%	100%
Cosine	100%	100%
Pearson	100%	100%
Our method	97.18%	99.91%

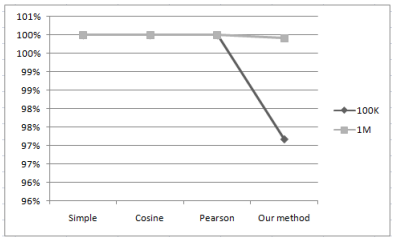

Fig. 3. Recommend recall evaluation

4.4 Usefulness Measure

The precise measure is based on statistical data, it doesn't reflect how much user really like recommended items and so it is very rigid. There is in case that some items are given to user and they are predicted with high consistency and high precision but they are not interesting items for user. The usefulness indicator measures the level of user interest on such recommended items. In other words, it tells us how useful such recommended items are and so this measure is the most important one. Usefulness measure (*U*) is the ratio of the number of interesting items to the number of best items. Best item is defined as the item which algorithm recommends to user and its rating value is highest. Interesting item is the item on which user rates with highest value.

Usefulness (*U*) = (*number of interesting items / number of best items*) * *100*

For example, algorithm gives user $item_1$, $item_3$, $item_5$ whose values are *1, 5, 5*, respectively and user rates on $item_2$, $item_3$, $item_4$ with values *2, 5, 3,* respectively. Best

items are $item_3$ and $item_5$. Interesting item is $item_3$. So the usefulness measure is *1 / 2 = 50%*. Our method gets less precise measure but it gains a highest usefulness. It is much better than other methods when we consider both the usefulness and the cost of time consuming (*T* measure). The reason is that our method discovers the patterns of user ratings while other methods try to predict missing values. So all recommended items conform to user interests. Following is the line chart of usefulness evaluation:

Table 7. Usefulness evaluation (%)

	100K	1M
Simple	10.64%	10%
Cosine	10.64%	10%
Pearson	10.64%	10%
Our method	22.73%	24.86%

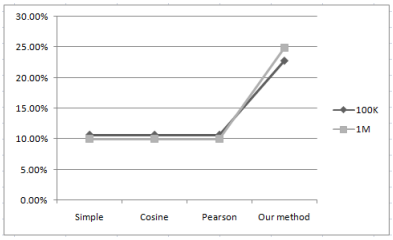

Fig. 4. Usefulness evaluation (%)

5 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. Su, X., Khoshgoftaar, T. M. *A Survey of Collaborative Filtering Techniques*. Hindawi Publishing Corporation, Advances in Artificial Intelligence, Volume 2009, Article ID 421425, 19 pages, doi:10.1155/2009/421425.
3. Jiawei Han, Michelline Kamber. *Data Mining: Concepts and Techniques*. Second Edition. © 2006 by Elsevier Inc.