**A New Approach for Collaborative Filtering based on Mining Frequent Itemsets**

Loc Nguyen

Huong Duong Company, Ho Chi Minh city, Vietnam

**Abstract**

Collaborative filtering (CF) is one of the most popular algorithms, for recommendation in cases, the items which are recommended to users, have been determined by relying on the outcomes done on surveying their communities. There are two main CF-approaches, which are memory-based and model-based.

We are, instead to propose a new CF model-based approach which has been implemented by basing on mining frequent itemsets technique. The database is mined to extract frequent itemsets. Hence items which have belonged to such frequent itemsets are ones, recommended to users. Because the mining task is to be performed at offline process-mode, which doesn’t give any negative effect on the recommendation task, as well our algorithm can also even better still by providing the recommendations with an immediate response.

We also prefer to propose another so-called Roller algorithm for improving process of mining frequent itemsets, when mining task is the heart of our CF approach. Simply because the Roller algorithms is implemented by assuming that “*The larger the support of an item is, the higher it’s likely that this item will occur in some frequent itemset, is*”. It’s a method, called the heuristic algorithm. Therefore the Roller algorithm model itself, upon doing the white-wash tasks, which is to roll a roller on a wall in such a way that would be capable of picking the frequent itemsets.

Moreover we also, together with it, are to provide the so-called enhanced techniques such as the followings: bit representations, bit matching as well bit mining in order to speeding up the algorithm processing with CF method. The basis of doing these techniques are to take advantages of bit-operations (*AND*, *NOT* functions) as well to be able to reduce the storages-capacities occupying spaces and to even giving a better chance of improvements to making the algorithms to be able to run faster.

**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 users are to considering such as products, services, 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 those meanings, there are two recommendations systems, found to be with a common trends: content-base filtering (CBF) and collaborative filtering (CF) [1] [5]:

* The CBF recommends an item to a user if such item has similarities in contents to other items that he like most in the past (and his rating for such item is high). Note that each item has contents which are their properties and so all items will compose a matrix, called the items content matrix.
* The CF on the other hands, recommends an items to user if his neighbors (mean the other users that are similar to him) are interested in such item. Notes that, user’s rating on any item does express his interest on that item. For that reason, all user’s ratings which carry out on the items will also composes a matrix, called the rating matrix.

Both of the above mentioned filtering (CBF & CF) do have their own strong, as well weak points. The CBF one is the one to focus on the item’s contents and user personality’s interests. And it is designed to recommend different items to different users. Each user therefore, can receive a unique recommendations; and this is also the strong point of CBF filtering. However CBF doesn't tend towards community like CF does. As the items that users may like “are hidden under” user community, CBF hasn’t been capable of discovering such implicit items. Because of this, it is acknowledged as a common weak point of CBF. Moreover, in case the number of users becomes larger at a huge volume, CBF may give a wrong prediction; else the accuracy of CF will get increased.

If there will be a huge contents associated with items, for instance and these items have had various properties then CBF in return, will consume even much more system resources, as well the processing time in order to analyze these items whereas, CF as a matter of fact, doesn’t pay any regard to the contents of the meant items. Instead the CF only works on the users’ ratings of the items and it is known as the strong point of this CF type. Because of that, CF wouldn’t be encountering with problems, such as how to analyze the richness in items’ contents. However this is also to reflecting the weak points of CF type as well, simply because CF can also do some unexpected recommendations in some situations, in which items are to be considered suitable to users, but they don’t relate to users’ profiles in fact. The problem then even turns into more serious trouble when having to facing with too many items which aren’t rated. It turns the rating matrix into the spare one which is to containing various missing values. 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, and then the next step would be the CF type, which is used to making predictions for recommendations. This mentioned technique will be positively useful to improve the predictions’ precision. But it does consuming more time when the first stage plays the role of the filtering step or pre-processing step while the content of items must be fully represented as a requirement. This technique is designed to requiring both, the items’ content matrix, and the rating matrix.
* 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 approach for the CF type. Note that to this CF type, there have been two common approaches, such as the memory-based and the model-based approaches. The model-based approach applies statistical and machine learning methods to mining the rating matrix. The result of this mining task is the above mentioned model.

Although the model-based approach doesn’t give result which is as precise as the combination approach, it can solve the problem of huge database and sparse matrix. Moreover it can responds user’s request immediately by making prediction on representative model though instant inference mechanism. So 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 [4], leading to get fastest speed and high quality of recommendation. In section *2* we propose an idea for the model-based CF algorithm based on mining frequent itemsets. The heuristic mining 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 vector, for instance, *u =* (*item 1 = 3, item 2 = 5, item 3 = 2*). It means that user *u* rated on *item 1, item 2* and *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 from rating matrix 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 online 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. Namely, it is impossible to estimate rating values for items *5* and *7*. Now please pay attention, these problems will be eliminated or solved by using the technique so-called *bit transformation*. Note that the terms “*bit*” and “*binary*” have the same meaning in this research.

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*} where *5*-most favorite and *1*-most dislike, then, the sample rating matrix is shown as what will be seen in table *1*.

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 sub-item *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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 1.** Rating matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | User 1 | User 2 | User 3 |
| Item\_1\_1 |  |  | *1* |
| Item\_1\_3 | *1* | *1* |  |
| Item\_2\_5 | *1* | *1* | *1* |
| Item\_3\_2 | *1* | *1* |  |
| Item\_3\_4 |  |  | *1* |
| Item\_4\_1 | *1* | *1* |  |

**Table 2.** Bit rating matrix

Each frequent itemset, that has been extracted from bit rating matrix and, it will carry a so-called bit form, *s* = (*item\_j1\_k1, item\_j2\_k2,…, item\_jm\_km*). Where, each element *item\_j\_k*, has been defined as bit sub-item. After that, rating vector is also to be transformed into bit rating vector *u* = (*item\_j1\_k1, item\_j2\_k2,…, item\_jn\_kn*). 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* = (*item 1 = 3, item 2 = 5, item 3 = 2*) is to be transformed into *u* = (*item\_1\_3, item\_2\_5, item\_3\_2*). While the bit frequent itemsets are *s1* = (*item\_1\_3, item\_2\_5, item\_3\_2, item\_4\_1*) and *s2* = (*item\_1\_1, item\_2\_5, item\_3\_4*). We also find that itemset *s1*, has been matched at the most, to *u* and so, the additional item *4* in *s1* is recommended to user with predictive value *1*.

Now the previous mentioned problems are 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 a rating vector or frequent itemset contains *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 such a rating vector or frequent itemset. 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 (bit *1*), its corresponding sub-item is rated with the value which is the position of such set bit. Following is an example of bit set:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *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* | | | | |

**Table 3.** Bit representation

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 *s1 =* (*item\_1\_3, item\_2\_5, item\_3\_2, item\_4\_1*) which is represented as *s1 =* (*00100 00001 01000 10000*). The frequent itemset *s2 =* (*item\_1\_1, item\_2\_5, item\_3\_4*) which is represented as *s2 =* (*10000 00001 00010 00000*). In order to match *s1* (or *s2*) with *u*, all we need is to do *AND* bit-operation between *s1* (or *s2*) and *u*.

* If *s1 AND u = u* then *s1* matches with *u*.
* Otherwise *s1 AND u ≠ u* then *s1* doesn’t match with *u*.

When *s1* 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 = s1 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 *4th* cluster is set.

As a result, our algorithm will consist of three following 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* language 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\_itemsets* 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* language for our CF algorithm:

*B = bit\_transform*(*D*)

*S = mining\_frequent\_itemsets*(*B*)

*matched\_itemset = null*

*max\_count = -1*

*For each s ϵ 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, mining frequent itemsets from bit rating matrix, 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 *min\_sup* are removed from such descending ordering. The kept items are called the frequent items. Of course, the first item in this descending ordering gets maximum support.
* *Step 2*: The *ith* itemset (*i =* ) is initialized by the first item in this descending ordering. The support of *ith* itemset is initialized as the support of this first item. The current item now is the first item and it will be removed from descending ordering.
* *Step 3*: If there is no item in descending ordering, the algorithm will be terminated. Otherwise:
  + If the current item is the last one, in descending ordering, then all items in the *ith* itemset are removed from the descending ordering and the number *i* is increased by *1* (*i = i + 1*). Go back step *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: if it satisfies minimum support, the support of the *ith* itemset denoted *support*(*ith* *itemset*) is accumulated by current item; it is the count of total transactions that contains all items in both the *ith* itemset and current item. If *support*(*ith* *itemset*) is equal to or larger than minimum support, current item is added to the *ith* 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 *ith* itemset modeled as the roller. After each rolling (each iteration), such itemset gets thicker with more items.

Let *I* = (*i1, i2,…, im*) and *S* be a set of item and a set of frequent itemset, respectively. Let *C =* (*c1, c2,…, cn*) be the list of items whose supports meet minimum support (*min\_sup*) and are sorted according to descending ordering, *C ⊆ I*. Let *si* be the *ith* itemset. Let *c* be the current item. Let *filter\_minimum\_support*(*…*) be the function that filters items whose supports are greater than or equal to minimum support. 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 language for Roller algorithm programmed as function *mining\_frequent\_itemsets*:

*C = filter\_minimum\_support*(*I*)

*C = sort*(*C*)

*i = 1*

*While* (*C ≠ ∅*)

*c = first*(*C)*

*si =* {*c*}

*While* (*true*)

*If c = last*(*C*) *then*

*S = S* ∪ *si*

*C = C / S*

*break*

*Else*

*c = next*(*C*)

*b = bitset*(*si*) *AND bitset*(*c*)

*If count*(*b*) *≥ min\_sup then*

*si = si* ∪{*c*}

*End If*

*End If*

*End While*

*i = i +1*

*End While*

Although the Roller algorithm may ignore some frequent itemsets but it runs much faster than traditional mining frequent itemsets methods. Especially Roller 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, we have:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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* |

**Table 4.** Bit rating matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | User 1 | User 2 | User 3 |
| Item\_2\_5 | *1* | *1* | *1* |
| Item\_1\_3 | *1* | *1* | *0* |
| Item\_3\_2 | *1* | *1* | *0* |
| Item\_4\_1 | *1* | *1* | *0* |

**Table 5.** Frequent items given minimum support *2*

In step *1*, sub-items are sorted according to descending ordering of their supports and some sub-items not satisfying minimum support (*min\_sup*) are removed given the minimum support 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 *s1* is initialized as *Item\_2\_5* that is the first item in the ordering.

*s1 =* (*111*) and *support*(*s1*) = *count* (*111*) *= 3*

Where *support*(*…*) denotes the support of itemset and *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* and *Item\_4\_1* are picked in turn and all of them satisfy minimum support.

* Picking *Item\_1\_3*: *s1 = s1 ⋃* {*Item\_1\_3*} *=* (*111*) *AND* (*110*) *=* (*110*) *→ support*(*s1*) *= 2*.
* Picking *Item\_3\_2*: *s1 = s1 ⋃* {*Item\_3\_2*} *=* (*110*) *AND* (*110*) *=* (*110*) *→ support*(*s1*) *= 2*.
* Picking *Item\_4\_1*: *s1 = s1 ⋃* {*Item\_4\_1*} *=* (*110*) *AND* (*110*) *=* (*110*) *→ support*(*s1*) *= 2*.

Finally, the frequent itemset is *s1 =* (*110*) which include *Item\_2\_5, Item\_1\_3, Item\_3\_2* and *Item\_4\_1*. We recognize that the bit set of frequent itemset, named *s1* 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. The concept of pseudo-maximum is defined later. The improved Roller algorithm is described as below:

* *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 such descending ordering. The kept items are called the frequent items. Of course, the first item in this descending ordering gets maximum support.
* *Step 2*: The *ith* itemset (*i =* ) is initialized by the first item in this descending ordering. The support of *ith* itemset is initialized as the support of this first item. The current item now is the first item and it will be removed from descending ordering.
* *Step 3*: If there is no item in descending ordering, the algorithm will be terminated. Otherwise:
  + If the current item is the last one, in descending ordering, then all items in the *ith* itemset are removed from the descending ordering and the number *i* is increased by *1* (*i = i + 1*). Go back step *2*.
  + If the current item is NOT the last in descending ordering, we look an item up in the ordering so that such item can combine via *AND* bit-operation with *ith* 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.
* *Step 4*:
  + Checking the support of current item: if it satisfies minimum support, the support of the *ith* itemset denoted *support*(*ith* *itemset*) is accumulated by current item; it is the count of total transactions that contains all items in both the *ith* itemset and current item. If *support*(*ith* *itemset*) is equal to or larger than minimum support, current item is added to the *ith* itemset.
  + Go back step *3*.

Following is the pseudo-code like C language for improved Roller algorithm programmed as function *mining\_frequent\_itemsets*:

*C = filter\_minimum\_support*(*I*)

*C = sort(C)*

*i = 1*

*While (true)*

*c = first*(*C*)

*si =* {*c*}

*C = C /* {*c*}

*S = S* ∪ *si*

*While* (*true)*

*c = null*

*pseudo\_maximum = -1*

*For each item d ∈ C*

*temp = bitset*(*S*) *AND bitset*(*d*)

*If count*(*temp*) *≥ min\_sup &&*

*count(temp)> pseudo\_maximum then*

*c = d*

*pseudo\_maximum = count*(*temp*)

*End If*

*End For*

*If c = null then*

*break*

*Else*

*si = si* ∪{*c*}

*C = C /* {*c*}

*End If*

*End While*

*i = i + 1*

*End While*

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**

Database *Movielens* [2] including 100,000 ratings of 943 users on 1682 movies is used for evaluation. Database is divided into 5 folders, each folder includes training set over 80% whole database and testing set over 20% whole database. Training set and testing set in the same folder are disjoint sets.

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 four other methods: neighbor item-based method, neighbor user-based and SVD [5].

There are 7 metrics used in this evaluation: *MAE, MSE, precision, recall, F1, ARHR* [3] and *time*. Note that time metric is calculated in seconds. MAE and MSE are predictive accuracy metrics that measure how close predicted value is to rating value. The less MAE and MSE are, the high accuracy is. Precision, recall and F1 are quality metrics that measure the quality of recommendation list – how much the recommendation list reflects user’s preferences. ARHR is also quality metric that indicates how well recommendation list is matched to user’s rating list according to rating ordering. The large quality metric is, the better algorithm is. Following table is the evaluation result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Our method** | **Item-based** | **User-based** | **SVD** |
| MAE | *0.8529* | *0.5222* | *0.9319* | *0.5363* |
| MSE | *1.6304* | *0.6675* | *2.1664* | *1.1734* |
| Precision | *0.1058* | *0.0245* | *0.0014* | *0.0041* |
| Recall | *0.0404* | *0.0092* | *0.0005* | *0.0015* |
| F1 | *0.0576* | *0.0131* | *0.0008* | *0.0021* |
| ARHR | *0.0341* | *0.0040* | *0.0003* | *0.0012* |
| Time | *0.0050* | *8.5607* | *7.5558* | *0.0207* |

**Table 6.** Evaluation result

Our method is more effective than other methods when getting high quality via metrics such as precision, recall, F1 and ARHR. Its accuracy is slightly lower than item-based, user-based methods and SVD via metrics such as MAE and MSE. The best thing is that it runs much faster than other methods.

**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 quality of recommendation and consuming less time and resources. Its sole drawback is that it may ignore some interesting patterns because of heuristic assumption. However this drawback is alleviated by taking advantage of enhancement techniques such as bit mining and 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 mining method in this paper so that we have an open and objective viewpoint about the domain of recommendation.

**Acknowledgment**

This research is the place to acknowledge Ms. Phung, Do Thi Minh – University of Information Technology, Vietnam National University and Mr. Dong, Vu Ngoc who gave me valuable comments and advices. These comments help me to improve this research.

**Reference**

1. [Su, Khoshgoftaar 2009]. Xiaoyuan Su and Taghi M. Khoshgoftaar. 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.
2. [Movielens dataset 2011]. Home page is http://www.movielens.org. Download dataset from http://www.grouplens.org/node/12.
3. [Herlocker, Konstan, Terveen, Riedl 2004]. Jonathan. L. Herlocker, Joseph. A. Konstan, Loren G. Terveen, and John. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems, vol. 22, no. 1, pp. 5–53, 2004.
4. [Han, Kamber 2006]. Jiawei Han and Michelline Kamber. Data Mining: Concepts and Techniques. Second Edition. © 2006 by Elsevier Inc.
5. [Ricci, Rokach, Shapira, Paul 2011]. Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Recommender Systems Handbook. ISBN: 978-0-387-85819-7, e-ISBN: 978-0-387-85820-3, DOI: 10.1007/978-0-387-85820-3. Springer New York Dordrecht Heidelberg London, Springer Science+Business Media Copyright, LLC 2011.
6. [Ungar, Foster 1998]. L. H. Ungar and D. P. Foster. Clustering Methods for Collaborative Filtering. Proceedings of the Workshop on Recommendation Systems, AAAI Press, 1998.
7. [Breese, Heckerman, Kadie 1998]. John S. Breese, David Heckerman and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Technical Report MSR-TR-1998. Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmon, WA 98052.
8. [Hofmann 2004]. Thomas Hofmann. Latent Semantic Models for Collaborative Filtering. ACM Transactionson Information Systems, Vol.22, No.1, January 2004, Pages 89-115.
9. [Shani, Heckerman, Brafman 2005]. G. Shani, D. Heckerman, and R. I. Brafman. An MDP-based Recommender System. Journal of Machine Learning Research, vol. 6, pp. 1265-1295, 2005.

**Một cách tiếp cận mới cho lọc cộng tác dựa trên khai thác tập phổ biến**

Nguyễn Phước Lộc

Công ty TNHH MTV Lập trình Hướng Dương, Tp. Hồ Chí Minh city, Việt Nam

**Tóm tắt**

Lọc cộng tác (collaborative filtering – CF) là một trong những giải thuật khuyến nghị phổ biến, mỗi mặt hàng giới thiệu cho một người dùng dựa vào kết quả khảo sát các đánh giá từ cộng đồng nhiều người dùng. Có hai cách tiếp cận cho CF: nạp ức (memory-based) và mô thức (model-based).

Chúng tôi đề xuất một giải thuật CF mô thức mới dựa trên khai thác tập phổ biến từ cơ sở dữ liệu. Theo đó, những mặt hàng trong tập phổ biến sẽ được khuyến nghị cho người dùng. Tác vụ khai khoáng tập phổ biến được thực hiện ngoại tuyến nên không ảnh hướng đến tác vụ khuyến nghị, vì vậy giải thuật đề xuất có khả năng đáp ứng tức thời yêu cầu khuyến nghị.

Do trọng tâm của giải thuật CF mô thức mới là khai thác tập phổ biến, chúng tôi cũng đề xuất một giải thuật khai khác tập phổ biến mới được gọi là Roller nhằm tăng hiệu quả của tiến trình khai thác tập phổ biến. Giải thuật Roller dựa trên giả định “*độ hỗ trợ* (*support*) *của một mặt hàng càng lớn thì khả năng xuất hiện mặt hàng này trong một tập phổ biến nào đó càng cao*”. Đây là giải thuật thông minh (heuristic), mô phỏng thao tác lăn sơn tường, theo đó, tập phổ biến được khởi gán và mở rộng qua động tác lăn sơn tường.

Ngoài ra, chúng tôi cũng đề xuất những kỹ thuật tăng cường như biểu diễn nhị phân (bit representation), so khớp nhị phân (bit matching) cũng như khai thác nhị phân (bit mining) nhằm gia tăng tốc độ thuật toán. Điểm cơ bản của các kỹ thuật này là giảm không gian lưu trữ và thời gian xử lý bằng toán tử nhị phân như *AND, NOT*.

**Từ khóa**: lọc cộng tác, khai thác tập phổ biến, so khớp nhị phân, khai thác nhị phân

**1. Giới thiệu**

Hệ thống khuyến nghị (recommendation system) là hệ thống giới thiệu những mặt hàng mà người dùng có khả năng yêu thích nhất trong một cơ sở dữ liệu lớn gồm rất nhiều mặt hàng. Mặt hàng có thể là sản phẩm, dịch vụ, sách, tạp chí và tất cả những gì mà người dùng với vai trò khách hàng có thể mua. Những mặt hàng được giới thiệu đó thể hiện sở thích (interests) hay mối quan tâm của người dùng.

Hệ thống khuyến nghị được xây dựng theo hai hướng: lọc nội dung (content-based filtering – CBF) và lọc cộng tác (collaborative filtering – CF) [1]:

* CBF giới thiệu một mặt hàng với người dùng nếu mặt hàng này tương tự về nội dung với những mặt hàng mà họ đã yêu thích với ngữ nghĩa rằng mặt hàng yêu thích là mặt hàng có đánh giá cao. Nội dung của một mặt hàng là những thuộc tính của nó và tập nội dung tất cả các mặt hàng tạo thành ma trận, gọi là ma trận nội dung (content matrix).
* Với tiếp cận khác, CF giới thiệu một mặt hàng với người dùng nếu những láng giềng của họ cũng thích mặt hàng đó với định nghĩa láng giềng là những người giống nhau về sở thích. Đánh giá của một người dùng lên một mặt hàng thể hiện sở thích của họ đối với mặt hàng đó. Tất cả đánh giá người dùng trên các mặt hàng tạo thành ma trận, gọi là ma trận đánh giá (rating matrix).

Cả hai giải thuật CBF và CF đều có ưu nhược điểm riêng. CBF tập trung vào nội dung mặt hàng và sở thích cá nhân người dùng nên mục tiêu của nó là khuyến nghị những mặt hàng khác nhau cho những người dùng khác nhau. Điểm mạnh của CBF là khả năng đưa ra những khuyến nghị đặc thù nhưng CBF không hướng đến cộng đồng như CF. Khi đó những mặt hàng mà người dùng có thể thích bị ẩn trong cộng đồng, không thể phát hiện được bằng CBF. Đây là nhược điểm của CBF, một nhược điểm khác, CBF có thể dự đoán sai sở thích người dùng nếu số lượng người dùng quá lớn.

Nếu nội dung mỗi mặt hàng quá lớn với nhiều thuộc tính, giải thuật CBF tiêu thụ nhiều tài nguyên máy tính và do đó thời gian xử lý sẽ chậm; ngược lại tốc độ không thành vấn đề đối với CF trong trường hợp này do CF không tập trung vào các thuộc tính mặt hàng. Thay vào đó, CF chỉ quan tâm đến đánh giá của người dùng trên mỗi mặt hàng và đây chính là điểm mạnh của CF. Tuy nhiên, mặc dù không đối mặt với vấn đề dữ liệu nội dung các mặt hàng quá lớn, nhược điểm của CF cũng nảy sinh từ đây, bởi vì CF sẽ đưa ra những khuyến nghị ngoài mong đợi trong trường hợp mà một số mặt hàng có thể phù hợp với người dùng nhưng họ chưa đánh giá chúng. Đây là vấn đề dữ liệu thưa, nghĩa là ma trận đánh giá bị thiếu nhiều giá trị hay nói cách khác có quá nhiều mặt hàng chưa được đánh giá. Hai kỹ thuật hữu ích được đề xuất để khắc phục nhược điểm này của CF:

* Kết hợp CBF và CF: kỹ thuật này gồm hai giai đoạn, giai đoạn đầu áp dụng CBF để hoàn chỉnh ma trận đánh giá, giai đoạn sau áp dụng CF để thực hiện khuyến nghị. Kỹ thuật này hữu ích khi cải thiện độ chính xác của khuyến nghị nhưng tốn nhiều thời gian cho giai đoạn đầu đóng vai trò như bước tiền xử lý. Kỹ thuật dùng cả ma trận đánh giá và ma trận nội dung
* Mô hình hóa ma trận đánh giá như một mô hình suy diễn, ước lượng giá trị bị thiếu và thực hiện khuyến nghị dựa trên mô hình suy diễn này. Giải thuật CF theo cách tiếp cận mô hình này được gọi là CF mô thức (model-based CF), còn một cách tiếp cận khác là CF nạp ức (memory-based CF). CF mô thức áp dụng các phương pháp thống kê và máy học để khai thác ma trận đánh giá.

Mặc dù cách tiếp cận mô thức không cho kết quả chính xác như cách tiếp cận kết hợp nhưng nó giải quyết vấn đề dữ liệu khổng lồ và ma trận thưa; hơn nữa nó có thể đáp ứng yêu cầu người dùng một cách tức thời qua cơ chế suy diễn nhanh nhạy. Vì vậy, bài báo này tập trung vào cách tiếp cận mô thức dựa trên khai khoáng tập phổ biến. Đây là hướng tiếp cận đầy tiềm năng vì tận dụng kỹ thuật khai khoáng [4], dẫn đến tốc độ nhanh và chất lượng khuyến nghị cao. Ý tưởng chính giải thuật được đề cập trong phần 2. Chi tiết giải thuật khai khoáng thông minh được mô tả chi tiết trong phần 3. Phần 4 là đánh giá và phần 5 là kết luận. Lưu ý trong bài báo các thuật ngữ như ma trận đánh giá (rating matrix), cơ sở dữ liệu đánh giá (rating database), tập huấn luyện (training set) có cùng nghĩa.

**2. Một giải thuật lọc cộng tác mới dựa trên khai thác tập phổ biến**

Giả sử với vector đánh giá *u* = {*item 1 = 3, item 2 = 5, item 3 = 2*}, nghĩa là người dùng *u* đánh giá các mặt hàng *item 1, item 2, item 3* lần lượt với các giá trị *3, 5, 2*. Giải thuật CF đề xuất, dựa trên khai thác tập phổ biến, gồm hai tiến trình sau:

* Tiến trình mô hình hóa: hoạt động ngoại tuyến với mục đích khai thác danh sách các tập phổ biến từ ma trận đánh giá.
* Tiến trình khuyến nghị: bất cứ khi nào người dùng *u* yêu cầu những mặt hàng yêu thích, một tập phổ biến *s* được rút từ *S* sao cho *s* chứa các mặt hàng *item 1, item 2, item 3*; ví dụ *s* = {*item 1, item 2, item 3, item 5, item 7*}. Các mặt hàng thêm vào là 5 và 7 sẽ được khuyến nghị cho người dùng. Không giống với tiến trình mô hình hóa, tiến trình khuyến nghị hoạt động trực tuyến đáp ứng tức thời yêu cầu khuyến nghị.

Tiến trình mô hình hóa tốn nhiều thời gian hơn tiến trình khuyến nghị nhưng do hoạt động ngoại tuyến nên không ảnh hưởng đến tiến trình khuyến nghị. Tuy nhiên vấn đề quan trọng phát sinh khi tập phổ biến *s* = {*item 1, item 2, item 3, item 5, item 7*} không thể hiện giá trị đánh giá của các mặt hành *item 1, item 2, item 3* vốn là 3, 5, 2 tương ứng trong vector đánh giá *u*. Điều này có nghĩa là vector đánh giá *u* và tập phổ biến *s* không khớp hoàn toàn. Một khó khăn khác xuất hiện khi không thể ước lượng các giá trị bị thiếu, ví dụ: không thể ước lượng giá trị đánh giá cho các mặt hàng *5* và *7*. Các vấn đề này được giải quyết bằng kỹ thuật *biến đổi nhị phân* (binary transformation). Lưu ý, thuật ngữ *bit* và *nhị phân* cùng ý nghĩa trong bài báo này.

Giả sử với ma trận đánh giá có dòng đại diện người dùng, cột đại diện mặt hàng, mỗi ô là giá trị đánh giá của một người dùng lên một mặt hàng. Các đánh giá dao động trong khoảng {*1 … 5*} với *5* biểu thị sự yêu thích nhiều nhất. Bảng biểu *1* thể hiện một ma trận đánh giá mẫu.

Mỗi mặt hàng được “tách” thành *5* mặt hàng con (sub-items) tương ứng với *5* giá trị đánh giá khả dĩ {*1 … 5*}. Mỗi mặt hàng con *item\_j\_k* có hai trạng thái nhị phân *1* và *0* cho biết người dùng có hay không đánh giá mặt hàng *j* với giá trị *k*. Ví dụ, trường hợp *item\_2\_5* có trạng thái *1* cho biết người dùng đã đánh giá mặt hàng *2* với giá trị *5*. Bây giờ ma trận đánh giá biến đổi thành ma trận đánh giá nhị phân (bit rating matrix), trong đó mỗi ô là một đánh giá nhị phân đối với một mặt hàng con (sub-item). Quy ước ô rỗng là ô nhận trạng thái *0*; nghĩa là không có người dùng đánh giá trên ô đó. Ma trận đánh giá nhị phân thể hiện trong bảng biểu 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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* |  |

**Bảng 1.** Ma trận đánh giá

|  |  |  |  |
| --- | --- | --- | --- |
|  | User 1 | User 2 | User 3 |
| Item\_1\_1 |  |  | *1* |
| Item\_1\_3 | *1* | *1* |  |
| Item\_2\_5 | *1* | *1* | *1* |
| Item\_3\_2 | *1* | *1* |  |
| Item\_3\_4 |  |  | *1* |
| Item\_4\_1 | *1* | *1* |  |

**Table 2.** Ma trận đánh giá nhị phân

Mỗi tập phổ biến *s* trích xuất từ ma trận đánh giá nhị phân sẽ ở dạng nhị phân, *s =* (*item\_j1\_k1, item\_j2\_k2, …, item\_jn\_kn*), trong đó mỗi phần tử *item\_j\_k* là một mặt hàng con nhị phân (*bit sub-item*). Sau đó mỗi vector đánh giá được biến đổi thành vector đánh giá nhị phân (*bit rating vector*) *u* = (*item\_j1\_k1, item\_j2\_k2,…, item\_jn\_kn*). Việc so khớp giữ tập phổ biến nhị phân và vector đánh giá nhị phân trở nên dễ dàng. Với ví dụ trên, vector đánh giá *u* = (*item 1 = 3, item 2 = 5, item 3 = 2*) được chuyển thành vector đánh giá nhị phân *u =* (*item\_1\_3, item\_2\_5, item\_3\_2*), giả sử có hai tập phổ biến nhị phân *s1 =* (*item\_1\_3, item\_2\_5, item\_3\_2, item\_4\_1*) and *s2 =* (*item\_1\_1, item\_2\_5, item\_3\_4*). Chúng ta nhận thấy *s1* khớp với *u* nhất, do đó mặt hàng *4*, mặt hàng khác biệt trong *s1* sẽ được khuyến nghị cho người dùng với đánh giá ước lượng là *1*.

Vậy, các vấn đề trên đã được giải quyết nhưng giải thuật chúng tôi cần được cải tiến. Nếu số lượng tập phổ biến lớn và mỗi tập phổ biến có nhiều mặt hàng, sẽ có sự bùng nổ tổ hợp làm giảm hiệu năng máy tính khi so khớp vector đánh giá và tập phổ biến. Do đó chúng tôi đề xuất một phương pháp cải tiến việc so khớp, được gọi là kỹ thuật so khớp nhị phân (*bit matching*).

**2.1. Biểu diễn nhị phân và so khớp nhị phân**

Giả sử vector đánh giá hay tập phổ biến có *4* mặt hàng và mỗi mặt hàng có *5* giá trị đánh giá, chúng ta dùng một tập bit (bit set) dài *4 \* 5* = *20* bit để biểu diễn vector đánh giá này. Tập bit được chia thành những cụm (cluster) hay nhóm (group), ví dụ, nếu mặt hàng có *5* giá trị đánh giá thì mỗi cụm sẽ có *5* bit. Như vậy mỗi cụm đại diện một mặt hàng con (sub-item) và vị trí của một bit trong một cụm cho biết giá trị đánh giá của mặt hàng con tương ứng. Nếu cụm có một bit được bật, mặt hàng con tương ứng được đánh giá với giá trị là vị trí của bit được bật đó trong cụm. Sau đây là một ví dụ của tập bit.

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

**Bảng 3**. Biểu diễn bit

Ví dụ, vector đánh giá *u* = (*item 1 = 3, item 2 = 5, item 3 = 2*) được chuyển thành *u =* (*item\_1\_3, item\_2\_5, item\_3\_2*) biểu diễn dưới dạng nhị phân *u =* (*00100 00001 01000 00000*) gồm *4* cụm. Tập phổ biến *s1 =* (*item\_1\_3, item\_2\_5, item\_3\_2, item\_4\_1*) có dạng nhị phân là *s1 =* (*00100 00001 01000 10000*). Tập phổ biến *s2 =* (*item\_1\_1, item\_2\_5, item\_3\_4*)có dạng nhị phân là *s2 =* (*10000 00001 00010 00000*). Để so khớp *s1* (hay *s2*) với *u*, ta dùng toán tử nhị phân AND:

* Nếu *s1 AND u = u* thì *s1* khớp với *u*
* Ngược lại, nếu *s1 AND u u* thì *s1* không khớp *u*

Nếu *s1* khớp với *u* thì toán tử nhị phân *AND-NOT* được dùng để rút trích mặt hàng cần khuyến nghị với người dùng. Đặt *r\_item* là mặt hàng cần khuyến nghị, ta có:

*r\_item* = *s1 AND* (*NOT u*) = (*00000 00000 00000 10000*)

Tập bit của *r\_item* cho thấy bit đầu tiên của cụm thứ tư được bật, do đó mặt hàng *4* được khuyến nghị với giá trị ước lượng là *1*. Nhìn chung, giải thuật đề xuất gồm 3 bước:

* *Bước 1*: Biến đổi ma trận đánh giá thành ma trận đánh giá nhị phân.
* *Bước 2*: Rút trích những tập phổ biến từ ma trận đánh giá nhị phân bằng kỹ thuật khai khoáng.
* *Bước 3*:Vector người dùng và tập phổ biến biểu diễn dưới dạng tập bit. Toán tử so khớp bit được sử dùng để tìm ra tập phổ biến khớp với vector người dùng, đó là tập phổ biến thích hợp. Các mặt hàng cần khuyến nghị cho người dùng sẽ được rút trích từ tập phổ biến thích hợp này, hơn nữa, các giá trị bị thiếu cũng được ước lượng.

**2.2. Mã giả tựa C cho giải thuật đề xuất**

Đặt *D, B* và *S* lần lượt là ma trận đánh giá, ma trận đánh giá nhị phân và danh sách các tập phổ biến. Giả sử *matched\_itemset* và *r\_item* lần lượt là tập phổ biến thích hợp (đã được khớp) và mặt hàng khuyến nghị. Đặt *bitset*(…) là hàm biến đổi mặt hàng thành tập bit và *count*(…) là hàm đếm số bit *1* trong tập bit. Đặt *bit\_transform* là hàm biến đổi ma trận mặt hàng thành ma trận mặt hàng nhị phân và *mining\_frequent\_itemset* là hàm khai thác tập phổ biến từ ma trận đánh giá nhị phân (xem mục *3.1, 3.2*). Sau đây là mã giả tựa *C* cho giải thuật đề xuất:

*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*))

Bước thứ hai – khai khoáng tập phổ biến từ ma trận đánh giá nhị phân, quan trọng nhất; do đó phần tiếp theo sẽ đề cập đến phương pháp khai thác tập phổ biến.

**3. Khai thác tập phổ biến**

Chúng tôi đề xuất phương pháp khai thác tập phổ biến dựa trên giả định “*độ hỗ trợ* (support) *của một mặt hàng càng lớn thì khả năng xuất hiện mặt hàng này trong một tập phổ biến nào đó càng cao*”. Nói cách khác, những mặt hàng có độ hỗ trợ lớn có xu hướng kết hợp với nhau để tạo thành tập phổ biến. Giải thuật chúng tôi, tên gọi *Roller*, là một giải thuật thông minh (heuristic). Ý tưởng chính giống thao tác lăn sơn tường. Hãy tưởng tượng, dữ liệu huấn luyện, cụ thể là ma trận đánh giá, được mô phỏng như một bức tường. Các mặt hàng được bố trí trên mặt tường theo thứ tự giảm dần độ hỗ trợ, nghĩa là, mặt hàng phổ biến thấp theo sau mặt hàng phổ biến cao. Sau đó, chúng ta có một cây lăn tường và lăn nó trên bức tường, quét qua các mặt hàng theo thứ tự giảm dần độ hỗ trợ. Nếu một mặt hàng thỏa ngưỡng hỗ trợ dưới (minimum support – *min\_sup*) thì nó được thêm vào tập phổ biến và thao tác lăn này tiếp tục cho đến khi không còn mặt hàng nào thỏa ngưỡng hỗ trợ dưới. Lần tiếp theo, tất cả mặt hàng trong tập phổ biến trước đó sẽ bị gỡ khỏi bức tường và thao tác lăn tường mới lại tiếp tục để tìm tập phổ biến khác. Giải thuật gồm bốn bước sau:

* *Bước 1*: tính độ hỗ trợ tất cả các mặt hàng và sắp xếp chúng trên bức tường theo một danh sách ứng với độ hỗ trợ giảm dần, tiêu chuẩn loại trừ là những mặt hàng không thỏa ngưỡng hỗ trợ dưới *min\_sup*. Nói cách khác, bức tường chỉ gồm các mặt hàng thỏa ngưỡng hỗ trợ dưới, được gọi là mặt hàng phổ biến (frequent items) và mặt hàng đầu tiên trong danh sách là mặt hàng có độ hỗ trợ lớn nhất.
* *Bước 2*: Tập phổ biến thứ *i* (i = ) được khởi tạo là mặt hàng đầu tiên trong danh sách và có độ hỗ trợ cũng bằng độ hỗ trợ của mặt hàng đầu tiên. Mặt hàng đầu tiên này trở thành mặt hàng hiện hành và sẽ được gỡ khỏi danh sách.
* *Bước 3*: Nếu không còn mặt hàng nào trong danh sách, giải thuật sẽ dừng. Ngược lại:
  + Nếu mặt hàng hiện hành là mặt hàng cuối cùng trong danh sách thì tất cả mặt hàng trong tập phổ biến thứ *i* sẽ bị gỡ khỏi danh sách và *i* được tăng lên *1* (*i = i + 1*). Trở lại bước 2.
  + Nếu mặt hàng hiện hành KHÔNG phải mặt hàng cuối cùng trong danh sách thì mặt hàng kế tiếp được xét và trở thành mặt hàng hiện hành.
* *Bước 4*:
  + Kiểm tra độ hỗ trợ của mặt hàng hiện hành, nếu thỏa ngưỡng hỗ trợ dưới thì độ hỗ trợ của tập phổ biến thứ *i*, ký hiệu *support*(*itemset i*) được cộng tích lũy với mặt hàng hiện hành, chính là tổng tất cả giao dịch chứa cả tập phổ biến thứ *i* và mặt hàng hiện hành. Nếu *support*(*itemset i*) lớn hơn hoặc bằng ngưỡng hỗ trợ dưới thì mặt hàng hiện hành được thêm vào tập phổ biến thứ *i*.
  + Trở lại bước 3.

Bước *3* và *4* tương tự thao tác sơn lăn tường với cây lăn là tập phổ biến thứ *i*. Mỗi lần lặp là một lần lăn, sau mỗi lần lăn, tập phổ biến sẽ dày lên vì đính càng nhiều mặt hàng.

Gọi *I =* {*i1, i2, …, im*} là danh sách các mặt hàng và *S* là danh sách các tập phổ biến. Đặt *C =* {*c1, c2, …, cn*} là danh sách các mặt hàng có độ hỗ trợ thỏa ngưỡng hỗ trợ dưới và sắp theo thứ tự giảm dần, *C I*. Đặt *si* là tập phổ biến thứ *i*. Đặt *c* là mặt hàng hiện hành. Đặt *filter\_minimum\_support*(…) là hàm lọc ra những mặt hàng có độ hỗ trợ lớn hơn hoặc bằng ngưỡng hỗ trợ dưới. Đặt *sort*(…), *first*(…), *next*(…), *last*(…) lần lượt là các hàm sắp xếp, lấy phần tử đầu tiên, lấy phần tử cuối cùng từ một danh sách các mặt hàng. Sau đây là mã giả tựa *C* cho giải thuật Roller được lập trình như hàm *mining\_frequent\_itemsets*.

*C = filter\_minimum\_support*(*I*)

*C = sort*(*C*)

*i = 1*

*While* (*C ≠ ∅*)

*c = first*(*C*)

*si =* {*c*}

*While* (*true*)

*If c = last*(*C*) *then*

*S = S si*

*C = C / S*

*break*

*Else*

*c = next*(*C*)

*b = bitset*(*si*) *AND bitset*(*c*)

*If count*(*b*) *≥ min\_sup then*

*si = si* {*c*}

*End If*

*End If*

*End While*

*i = i +1*

*End While*

Mặc dù giải thuật Roller có thể bỏ sót vài tập phổ biến nhưng nó chạy nhanh hơn giải thuật khai thác tập phổ biến truyền thống. Mục tiếp theo sẽ trình bày cách cải tiến giải thuật Roller bằng kỹ thuật khai khoáng nhị phân.

**3.1. Khai khoáng nhị phân**

Khi ma trận đánh giá biến đổi thành ma trận đánh giá nhị phân, mặt hàng và tập phổ biến trở thành cụm (mặt hàng con) và tập bit (xem mục 2). Độ hỗ trợ của mặt hàng và tập phổ biến là số lượng bit *1* trong tập bit. Bảng biểu sau mô tả ma trận đánh giá nhị phân.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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* |

**Bảng 4**. Ma trận đánh giá nhị phân

|  |  |  |  |
| --- | --- | --- | --- |
|  | User 1 | User 2 | User 3 |
| Item\_2\_5 | *1* | *1* | *1* |
| Item\_1\_3 | *1* | *1* | *0* |
| Item\_3\_2 | *1* | *1* | *0* |
| Item\_4\_1 | *1* | *1* | *0* |

**Bảng 5**. Mặt hàng phổ biến thỏa ngưỡng hỗ trợ dưới (là 2)

Bước *1* sắp xếp các mặt hàng con theo độ hỗ trợ giảm dần và loại bỏ mặt hàng nào không thỏa ngưỡng hỗ trợ dưới. Giả sử với ngưỡng hỗ trợ là 2, ta có các mặt hàng con (cụm): *Item\_2\_5 =* (*111*)*, Item\_1\_3 =* (*110*)*, Item\_3\_2 =* (*110*)*, Item\_4\_1 =* (*110*).

Bước *2* khởi tạo tập phổ biến *s1* với mặt hàng con đầu tiên *Item\_2\_5* trong danh sách và có độ hỗ trợ là *3*.

*s1 =* (*1111*) *và support*(*s1*) *= count*(*111*) *= 3*

Lưu ý, *support*(…) biểu diễn độ hỗ trợ của tập phổ biến và *count*(…) đếm số bit *1* trong tập bit.

Bước *3* và bước *4* lần lượt duyệt các mặt hàng con *Item\_1\_3, Item\_3\_2, Item\_4\_1*.

* Duyệt *Item\_1\_3*: *s1 = s1 ⋃* {*Item\_1\_3*} *=* (*111*) *AND* (*110*) *=* (*110*) *→ support*(*s1*) *= 2*.
* Duyệt *Item\_3\_2*: *s1 = s1 ⋃* {*Item\_3\_2*} *=* (*110*) *AND* (*110*) *=* (*110*) *→ support*(*s1*) *= 2*.
* Duyệt *Item\_4\_1*: s1 = *s1 ⋃* {*Item\_4\_1*} *=* (*110*) *AND* (*110*) *=* (*110*) *→ support*(*s1*) *= 2*.

Tập phổ biến sau cùng *s1 =* (*100*) bao gồm các mặt hàng *Item\_2\_5, Item\_1\_3, Item\_3\_2, Item\_4\_1*. Tập bit của tập phổ biến được tích lũy bởi từng mặt hàng qua mỗi lần lặp. Điều này làm thuật toán chạy nhanh hơn và chi phí cho toán tử nhị phân cùng đếm số bit *1* là không đáng kể.

**3.2. Cải tiến giải thuật Roller**

Giải thuật Roller có thể bỏ sót một số tập phổ biến trong trường hợp những mặt hàng không có độ hỗ trợ cao và chúng ở cuối danh sách. Mặc dù chúng không có nhiều cơ hội gia nhập vào một tập phổ biến nhưng chúng thật sự có khả năng kết thành tập phổ biến hoặc đóng góp vào một tập phổ biến nào đó, hay nói cách khác, chúng không làm giảm đi nhiều độ hỗ trợ của một tập phổ biến nếu như thêm chúng vào tập phổ biến đó. Tuy nhiên rất khó phát hiện các mặt hàng dạng này cùng với tính hữu ích của chúng. Để khắc phục nhược điểm này, giải thuật Roller cần cải tiến để không bỏ sót những mặt hàng như thế.

Theo đó, trong bước 3, thay vì chọn mặt hàng kế tiếp là mặt hàng hiện hành, chúng ta sẽ chọn mặt hàng có độ hỗ trợ *giả cực đại* (pseudo-maximum) làm mặt hàng hiện hành. Khái niệm *giả cực đại* định nghĩa sau. Giải thuật Roller cải tiến như sau:

* *Bước 1*: tính độ hỗ trợ tất cả các mặt hàng và sắp xếp chúng trên bức tường theo một danh sách ứng với độ hỗ trợ giảm dần, tiêu chuẩn loại trừ là những mặt hàng không thỏa ngưỡng hỗ trợ dưới *min\_sup*. Nói cách khác, bức tường chỉ gồm các mặt hàng thỏa ngưỡng hỗ trợ dưới, được gọi là mặt hàng phổ biến (frequent items) và mặt hàng đầu tiên trong danh sách là mặt hàng có độ hỗ trợ lớn nhất.
* *Bước 2*: Tập phổ biến thứ *i* (i = ) được khởi tạo là mặt hàng đầu tiên trong danh sách và có độ hỗ trợ cũng bằng độ hỗ trợ của mặt hàng đầu tiên. Mặt hàng đầu tiên này trở thành mặt hàng hiện hành và sẽ được gỡ khỏi danh sách.
* *Bước 3*: Nếu không còn mặt hàng nào trong danh sách, giải thuật sẽ dừng. Ngược lại:
  + Nếu mặt hàng hiện hành là mặt hàng cuối cùng trong danh sách thì tất cả mặt hàng trong tập phổ biến thứ *i* sẽ bị gỡ khỏi danh sách và *i* được tăng lên *1* (*i = i + 1*). Trở lại bước 2.
  + Nếu mặt hàng hiện hành KHÔNG phải mặt hàng cuối cùng trong danh sách thì mặt hàng hiện hành là mặt hàng khi kết hợp với tập phổ biến thứ *i* sẽ tạo thành một tập phổ biến mới có độ hỗ trợ lớn nhất. Mặt hàng này được gọi là mặt hàng có độ hỗ *trợ giả cực đại* (pseudo-maximum).
* Bước 4:
  + Kiểm tra độ hỗ trợ của mặt hàng hiện hành, nếu thỏa ngưỡng hỗ trợ dưới thì độ hỗ trợ của tập phổ biến thứ *i*, ký hiệu *support*(*itemset i*) được cộng tích lũy với mặt hàng hiện hành, chính là tổng tất cả giao dịch chứa cả tập phổ biến thứ i và mặt hàng hiện hành. Nếu *support*(*itemset i*) lớn hơn hoặc bằng ngưỡng hỗ trợ dưới thì mặt hàng hiện hành được thêm vào tập phổ biến thứ *i*.
  + Trở lại bước 3.

Sau đây là mã giả tựa C cho giải thuật Roller cải tiến được lập trình như hàm *mining\_frequent\_itemsets*.

*C = filter\_minimum\_support*(*I*)

*C = sort*(*C*)

*i = 1*

*While* (*true*)

*c = first*(*C*)

*si =* {*c*}

*C = C /* {*c*}

*S = S si*

*While* (*true*)

*c = null*

*pseudo\_maximum = -1*

*For each item d C*

*temp = bitset*(*S*) *AND bitset*(*d*)

*If count*(*temp*) *≥ min\_sup && count*(*temp*)*> pseudo\_maximum then*

*c = d*

*pseudo\_maximum = count*(*temp*)

*End If*

*End For*

*If c = null then*

*break*

*Else*

*si = si* {*c*}

*C = C /* {*c*}

*End If*

*End While*

*i = i + 1*

*End While*

Giải thuật Roller cải tiến chậm hơn một chút so với Roller thông thường do phải tìm những mặt hàng giả cực đại trong bước *3* nhưng phát hiện nhiều tập phổ biến hơn, vì thế đạt được độ chính xác cao hơn.

**4. Đánh giá**

Chúng tôi sử dụng cơ sở dữ liệu Movielen [1] gồm 100.000 đánh giá của 943 người dùng trên 1682 mặt hàng. Cơ sở dữ liệu được chia thành 5 nhóm (folder), mỗi nhóm chứa 80% tập huấn luyện (training set) và 20% tập kiểm thử (testing set). Tập huấn luyện độc lập tập kiểm thử trong cùng một bộ.

Thiết lập hệ thống phần cứng và phần mềm bao gồm: bộ xử lý Dual-Core CPU E5700 @ 3.00GHz, RAM 2GB, RAM còn trống 1GB, Microsoft Windows 7 Ultimate 2009 32-bit, Java 7 HotSpot (TM) Client VM. Giải thuật đề xuất được so sánh với bốn giải thuật khác: Green Fall là giải thuật khuyến nghị dựa trên khai thác tập phổ biến, láng giềng gần nhất dựa trên mặt hàng (item-based), láng giềng gần nhất dựa trên người dùng (user-based) và SVD [5].

Đánh giá sử dụng 7 độ đo: *MAE, MSE, RMSE, độ chuẩn xác, độ bao phủ, F1, ARHR* [11] and *thời gian xử lý*. Thời gian xử lý được tính bằng giây. MAE và MSE đo lường độ chính xác giữa giá trị dự đoán (ước lượng) và giá trị đánh giá thực tế. MAE và MSE càng nhỏ thì độ chính xác càng cao. Độ chuẩn xác, độ bao phủ, F1 đo lường chất lượng khuyến nghị, nghĩa là đo lường mức độ yêu thích của người dùng đối với những mặt hàng được khuyến nghị, nói cách khác, những mặt hàng mà hệ thống khuyến nghị cho người dùng phản ánh bao nhiêu phần trăm sở thích và mối quan tâm của người dùng. ARHR cũng là độ đo chất lượng khuyến nghị theo một khía cạnh khác – đo lường mức độ phù hợp giữa thứ tự các mặt hàng được khuyến nghị với thứ tự các mặt hàng mà người dùng đánh giá. Các độ đo chất lượng càng lớn, giải thuật càng tốt. Sau đây là kết quả đánh giá:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Our method** | **Item-based** | **User-based** | **SVD** |
| MAE | *0.8529* | *0.5222* | *0.9319* | *0.5363* |
| MSE | *1.6304* | *0.6675* | *2.1664* | *1.1734* |
| Độ chuẩn xác | *0.1058* | *0.0245* | *0.0014* | *0.0041* |
| Độ bao phủ | *0.0404* | *0.0092* | *0.0005* | *0.0015* |
| F1 | *0.0576* | *0.0131* | *0.0008* | *0.0021* |
| ARHR | *0.0341* | *0.0040* | *0.0003* | *0.0012* |
| Thời gian xử lý | *0.0050* | *8.5607* | *7.5558* | *0.0207* |

**Bảng 5.1**. Kết quả đánh giá

Giải thuật chúng tôi hiệu quả hơn nhiều so với các giải thuật khác qua các độ đo chất lượng như *độ chuẩn xác, độ bao phủ, F1, ARHR* mặc dù độ chính xác thấp hơn item-based, user-based, SVD qua các độ đo MAE, MSE. Điểm xuất sắc nhất là giải thuật chạy rất nhanh gấp 4 lần giải thuật chạy nhanh nhất SVD.

**5. Kết luận**

Giải thuật chúng tôi khác với những giải thuật CF mô thức khác qua việc phát hiện sở thích người dùng. Tập phổ biến khai thác từ cơ sở dữ liệu được xem như mẫu quan tâm hay sở thích người dùng. Chúng tôi sử dụng một kỹ thuật khai khoáng mới, gọi là Roller, để trích xuất tập phổ biến để thay thế phương pháp khai khoáng truyền thống vốn tiêu tốn nhiều thời gian và tài nguyên máy tính. Qua đánh giá, giải thuật Roller được chứng minh ưu việt với hiệu năng cao, tốc độ nhanh, độ chuẩn xác và độ bao phủ đều cao, tiêu tốn ít tài nguyên máy tính. Nhược điểm duy nhất là Roller có thể bỏ sót vài mẫu sở thích người dùng do giả định thông minh (heuristic). Tuy nhiên kỹ thuật khai khoáng nhị phân và khái niệm độ hỗ trợ *giả cực đại* đã khắc phục nhược điểm trên.

Chúng tôi dự định phát triển một giải thuật CF mô thức mới dựa trên suy diễn mạng Bayesian để phát hiện sở thích người dùng. Đây là phương pháp thống kê sẽ được so sánh với phương pháp khai khoáng trong nghiên cứu này để chúng ta có một góc nhìn mới về lĩnh vực khuyến nghị.

**Lời cảm ơn**

Cảm ơn bà Đỗ Thị Minh Phụng – Trường Đại học Công nghệ Thông tin, Đại học Quốc Gia Thành phố Hồ Chí Minh, Việt Nam và ông Vũ Ngọc Đồng đã giúp tôi hoàn thiện và cải tiến nghiên cứu này bằng những góp ý và lời khuyên bổ ích.

**Reference**

1. [Su, Khoshgoftaar 2009]. Xiaoyuan Su and Taghi M. Khoshgoftaar. 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.
2. [Movielens dataset 2011]. Home page is http://www.movielens.org. Download dataset from http://www.grouplens.org/node/12.
3. [Herlocker, Konstan, Terveen, Riedl 2004]. Jonathan. L. Herlocker, Joseph. A. Konstan, Loren G. Terveen, and John. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems, vol. 22, no. 1, pp. 5–53, 2004.
4. [Han, Kamber 2006]. Jiawei Han and Michelline Kamber. Data Mining: Concepts and Techniques. Second Edition. © 2006 by Elsevier Inc.
5. [Ricci, Rokach, Shapira, Paul 2011]. Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Recommender Systems Handbook. ISBN: 978-0-387-85819-7, e-ISBN: 978-0-387-85820-3, DOI: 10.1007/978-0-387-85820-3. Springer New York Dordrecht Heidelberg London, Springer Science+Business Media Copyright, LLC 2011.
6. [Ungar, Foster 1998]. L. H. Ungar and D. P. Foster. Clustering Methods for Collaborative Filtering. Proceedings of the Workshop on Recommendation Systems, AAAI Press, 1998.
7. [Breese, Heckerman, Kadie 1998]. John S. Breese, David Heckerman and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Technical Report MSR-TR-1998. Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmon, WA 98052.
8. [Hofmann 2004]. Thomas Hofmann. Latent Semantic Models for Collaborative Filtering. ACM Transactionson Information Systems, Vol.22, No.1, January 2004, Pages 89-115.
9. [Shani, Heckerman, Brafman 2005]. G. Shani, D. Heckerman, and R. I. Brafman. An MDP-based Recommender System. Journal of Machine Learning Research, vol. 6, pp. 1265-1295, 2005.