**A Novel Collaborative Filtering Algorithm by Bit Mining Frequent Itemsets**

Loc Nguyen

Sunflower Soft Company, Ho Chi Minh city, Vietnam

**Abstract**

Collaborative filtering (CF) is a popular technique in recommendation study. Concretely, items which are recommended to user are determined by surveying her/his communities. There are two main CF approaches, which are memory-based and model-based. I propose a new CF model-based algorithm which is implemented by basing on mining frequent itemsets from rating database. Hence items which belong to such frequent itemsets are recommended to user. Because the mining task is performed at offline process-mode, which doesn’t give any negative effect on the recommendation task, my CF algorithm can provide the recommendations with an immediate response.

I also propose another so-called Roller algorithm for improving the process of mining frequent itemsets when mining task is the heart of my CF approach. Because Roller algorithm 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*”, it is heuristic algorithm. Therefore Roller algorithm models upon doing white-wash tasks, which rolls a roller on a wall in such a way that is capable of picking frequent itemsets.

Moreover I provide enhanced techniques such as bit representation, bit matching and bit mining in order to speed up recommendation process. These techniques take advantages of bitwise operations (*AND*, *NOT*) so as to reduce storage space and make algorithms run faster.

**Keywords**: collaborative filtering, mining frequent itemsets, bit matching, bit mining.

**1. Introduction**

Recommendation system recommends users favorite items among a large number of existing items in database. Items refer to anything such as products, services, books, and newspapers. It is expected that users are most interested in recommended items. There are two common trends: content-base filtering (CBF) and collaborative filtering (CF) in building up a recommendations system as follows (Su & Khoshgoftaar, 2009, pp. 3-13) (Ricci, Rokach, Shapira, & Kantor, 2011, pp. 73-139):

* The CBF recommends an item to user if such item has similarities in contents to other items that he like most in the past (his rating for such item is high). Note that each item has contents which are its properties and so all items will compose a matrix, called *item content matrix*.
* The CF, on the other hand, recommends an item to user if his neighbors which are other users similar to him are interested in such item. User’s rating on any item expresses his interest on that item. All ratings on items will also compose a matrix, called *rating matrix*.

Both CBF and CF have their own strong and weak points. CBF focuses on item contents and user interests. It is designed to recommend different items to different users. Each user can receive a unique recommendation and so this is a strong point of CBF. However CBF doesn’t tend towards community like CF does. If items that user may like “are hidden under” his community, CBF is not able to discover such implicit items and so this is a weak point of CBF. Moreover, if the number of users is huge, CBF may give a wrong prediction whereas CF will get an increase in accuracy.

If there are huge contents associated with items, CBF will consume much more system resources and processing time in analyzing these items whereas CF does not encounter the problem of richness in item contents. CF only works on the user ratings of items instead, which is known as a strong point of CF. However, this also implies weak point of CF because CF can make unexpected recommendations in some situations, in which items are considered suitable to users but they do not relate to user profiles in fact. When many items are not rated, rating matrix is turned into *spare matrix* which contains various missing values. The weakness of CF becomes more serious in case of spare matrix. There are two typical approaches to alleviate this weakness, as follows:

* The first one combines CBF and CF, called *combination approach*. In typical scenario, the technique has two stages. Firstly, it applies CBF into completing spare matrix and secondly, it applies CF into making prediction for recommendation. The technique improves accuracy of prediction but it consumes much more time and resources because the first stage plays the role of pre-processing step and there is requirement of both item content matrix and rating matrix.
* The second one compresses spare matrix into a *representative model* which then is used to make recommendations. This is *model-based approach* for CF. Note that CF has two common approaches such as memory-based and model-based. The model-based approach often applies statistical and machine learning methods into constructing representative model.

Although model-based approach does not give such a precise result as combination approach does, it can solve the problem of huge database and sparse matrix. Moreover it can respond user’s request immediately by making prediction on representative model through instant inference mechanism. In general, there are four common approaches for model-based CF such as clustering model, latent model, Markov decision process model and matrix factorization model. Clustering CF (Ungar & Foster, 1998) is based on the assumption that users in the same group have the same interest; so they rate items similarly. There are two clustering techniques: using clustering algorithms (k-means, k-centroids, etc.) and using Bayesian classifier. Authors (Miyahara & Pazzani, 2000) propose the Simple Bayesian Classifier for CF. Suppose rating values range in the integer interval {1, 2, 3, 4, 5}, there is a set of 5 respective classes {*c*1, *c*2, *c*3, *c*4, *c*5}. The Simple Bayesian Classifier uses Naive Bayesian classification method (Miyahara & Pazzani, 2000, p. 4) to determine which class a given user belongs to. Author (Langseth, 2009) assumes that there is a linear mapping from the latent space of users and items to the numerical rating scale. Such mapping which conforms the full joint distribution over all ratings constructs a Bayesian network. Parameters of joint distribution are learned from training data, which are used for predicting active users’ ratings. According to (Campos, Fernández-Luna, Huete, & Rueda-Morales, 2010), the hybrid recommender model is the Bayesian network that includes three main kinds of nodes such as feature nodes, item nodes, and user nodes. Each feature node represents an attribute of item. Active users’ ratings are dependent on these nodes.

Given a set of user *U* = {*u*1*, u*2*,…, um*} and a set of items *I* = {*i*1*, i*2*,…, in*}, observation (*u, i*) where and is considered as the co-occurrence of user and item. A latent class variable *c* is associated with each co-occurrence (*u, i*). We have a set of latent class variables *C* = {*c*1*, c*2*,…, ck*}. The mapping *c*: *U* x *I* → *C* is called latent class model or aspect model (Hofmann, 2004, pp. 91-92)[6]. The problem which needs to be solved is how to specify the latent class model. Authors (Hofmann, 2004, p. 95) propose Expectation Maximization (EM) algorithm to determine which latent variable is the most suitable to be associated to (*u, i*).

Sparse rating matrix has insignificant rows or columns. Dimensionality reduction aims to get rid of such redundant rows or columns so as to keep principle rows/columns. As a result, the dimension of rating matrix is reduced as much as possible. Because this approach pays attention to analyzing matrix, it can be called matrix factorization approach. There are two well-known dimensionality algorithms such as singular value decomposition (SVD) and principle component analysis (PCA). Given rating matrix *M*, the SVD approach decomposes *M* into three partial matrices *U*, *Σ*, *V* so that *M* is product of them, *M* = *UΣVT* where the superscript *T* denotes matrix transposition operation (Percy, 2009, p. 1). *U* and *V* are called user feature matrix and item feature matrix, respectively. *Σ* is diagonal matrix of eigenvalues. In study of CF, SVD is approximated by two ways (Percy, 2009, pp. 2-5):

* Only *r* largest eigenvalues are used to compose matrix *Σ* and so matrices *U*, *Σ*, *V* become smaller.
* The objective function is established based on the sum of squared errors between existing ratings and predictive ratings. Matrices *U*, *Σ*, *V* are parameters of such objective function. The gradient descent approach is used to adjust these parameters so as to minimize such objective function. This method is proposed by authors Simon Funk and Genevieve Gorrell. It is also called incremental SVD.

According to (Shani, Heckerman, & Brafman, 2005, p. 1266), recommendation can be considered as a sequential process including many stages. At each stage a list of items which is determined based on the last user’s rating is recommended to user. So recommendation task is the best *action* that recommender system must do at concrete stage so as to satisfy user’s interest. The recommendation becomes the process of making decision so as to choose the best action. Authors (Shani, Heckerman, & Brafman, 2005, p. 1266) propose Markov decision process (MDP) to perform recommendation task. MDP is represented as four-tuple model <*S*, *A*, *R*, *T*> where *S*, *A*, *R*,and *T* are a set of states, a set of actions, reward function, and transition probability density, respectively (Shani, Heckerman, & Brafman, 2005, pp. 1270-1271). The essence of making decision process is to find out the optimal policy with regard to such four-tuple model. Note that a policy is defined as the function that assigns an action to pre-assumption state.

In this paper, I aims to propose a new approach for model-based CF in which users’ purchase patterns are mined and modeled as frequent itemsets which, in turn, are used to make recommendations. It is the efficient approach because it takes advantage of data mining (Han & Kamber, 2006, pp. 227-250), leading to get fastest speed and high quality of recommendation. Although there are many researches focusing on model-based CF (Su & Khoshgoftaar, 2009), the method to mine association rules for CF, proposed by authors (Shyu, Haruechaiyasak, Chen, & Zhao, 2005), is nearest to the CF algorithm in this paper. The authors (Shyu, Haruechaiyasak, Chen, & Zhao, 2005) apply shortest path algorithm of graph theory into finding distances between web pages with note that these distances are derived from use access sequences. Association rules are mined based on such distances, then applied into CF.

The core of the proposed CF is bit mining to discover frequent itemsets, mentioned in section 3. It is required to do a survey on bit mining. The terms “*bit*” and “*binary*” have the same meaning in this paper. Authors (Dong & Han, 2006) propose so-called *BitTableFI* algorithm. This algorithm uses data structure *BitTable* to horizontally and vertically compress database for generating candidate itemsets and counting their supports in quick. Each row in BitTable, corresponding with a transaction in purchase database, is a string of bits and each bit (0 or 1) indicates whether or not an item is purchased.

Authors (Song, Yang, & Xu, 2008) propose so-called *Index-BitTableFI* algorithm which takes advantages of BitTable. According to (Song, Yang, & Xu, 2008, p. 508), given a BitTable, index array and the corresponding computing method are proposed. By computing the subsume index, itemsets in accordance with representative item can be identified quickly through using breadth-ﬁrst search at one time (Song, Yang, & Xu, 2008, p. 509). Then, for the resulting itemsets generated by the index array, depth-ﬁrst search strategy is used to generate all other frequent itemsets (Song, Yang, & Xu, 2008, p. 510). Concept of index array is the core of Index-BitTableFI algorithm. This algorithm runs significantly faster than traditional BitTableFI algorithm.

Authors (León, Suárez, & Feregrino-Uribe, 2010) propose so-called *AMFI* algorithm based on breadth first search through equivalence classes combined with compressed vertical binary representation of dataset (León, Suárez, & Feregrino-Uribe, 2010, p. 486). The binary representation and equivalence classes make the algorithm run faster. AMFI is not close to the bit mining in this paper because the essence of AMFI is to take advantages of breadth first search and equivalence classes.

Authors (Raja & Raj, 2014) propose so-called *CPT-fi* algorithm following Apriori property but such algorithm clusters similar transactions into one and forms a *compact bit-table* so as to reduce memory consumption and frequency of checking itemsets. Compact bit-table is the most significant aspect of CPT-fi algorithm (Raja & Raj, 2014, p. 74).

Authors (Király, Gyenesei, & Abonyi, 2014) propose an Apriori-like algorithm but they take advantages of bit-table representation of dataset. Given two matrices of frequent itemsets, the algorithm takes Hadamard product of these matrices in order to generate candidate frequent itemsets (Király, Gyenesei, & Abonyi, 2014, p. 4).

Authors (Li, Ho, Shan, & Lee, 2006) propose so-called *MFI-TransSW* algorithm to mine frequent itemsets over online data streams with a *transaction-sensitive* *sliding window*. The idea of sliding window is significant. The *w*-length transaction-sensitive sliding window denoted *TransSW* includes *w* transactions. Given item *X* is represented by a *w*-length bit set. If *X* is in the *ith* transaction, the *ith* of bit set of *X* is set to be 1 (Li, Ho, Shan, & Lee, 2006, p. 2674). The MFI-TransSW has three phrase such as window initialization phrases, window sliding phrase, and frequent itemsets generation phrase (Li, Ho, Shan, & Lee, 2006, p. 2674). In the window initialization phrase, TransSW is created from coming transactions given pre-defined length *w*. The window sliding phrase is activated when TransSW is full. At that time, new coming transaction is added to TransSW and the oldest transaction is removed based on bitwise left shift operation. Finally, in the frequent itemsets generation phrase, MFI-TransSW algorithm uses bitwise operations to generate candidate itemsets and find frequent itemsets according to Apriori property (Li, Ho, Shan, & Lee, 2006, p. 2675).

Authors (Bashir, Jan, & Baig, 2009) build up lexicographic tree of items in transaction database. Itemset generation is done according to lexicographic order and bit-vector representation with attention that transaction database is transformed into bit table (Bashir, Jan, & Baig, 2009, p. 8). In a vertical bit-vector representation, if item *i* is in transaction *j*, the bit *j* of bit-vector *i* is set to be 1. Moreover, the authors (Bashir, Jan, & Baig, 2009, pp. 9-10) propose a so-called *Projected-Bit-Regions* (PBR) to gain efficient projection of bit-vectors for calculating supports of itemsets. PBR and lexicographic tree are significant aspects of the research, which leads to achieve high speed in mining frequent itemsets.

When compared with other researches on bit mining, the proposed heuristic mining algorithms is the simplest one in which there is no requirement of complex structure and complicated computation. In other words, its strong point is based on its simplicity, in which what we need to do includes rolling bit sets with bit operations. As a result, it gains high speed although it can lose some frequent itemsets. Note that bit mining serves CF application mentioned in section 2 and only optimal frequent itemset is used to generate recommendation list. So, it is not requisite for discovering all frequent itemsets in the most accurate way.

This paper consists of five sections. In section 2, I propose a new 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**

Given rating vector *u* =(*item* 1 = 3, *item* 2 = 5, *item* 3 = 2) indicates that user *u* rated on *item* 1, *item* 2and *item* 3 with values 3,5 and 2, respectively. The proposed CF algorithm, based on mining frequent itemsets, consists of two following processes:

* *Modeling process* is performed in offline mode, in which a set of frequent itemsets *S* is mined from rating matrix.
* *Recommendation process*: whenever user *u* requires 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 process is performed in online mode.

Although the modeling process consumes much more time than the recommendation one does, it is executed in offline mode and so it does not cause any negative time-consuming impact on recommendation process. However, a serious problem is raised when the frequent itemset *s* = (*item* 1, *item* 2, *item* 3, *item* 5, *item* 7) didn’t indicate rating values assigned to items 1, 2, and 3. It is known that items 1, 2 and 3 are rated by values 3, 5 and 2, respectively in rating vector *u*. It means that rating vector *u* and frequent itemset *s* don’t match exactly. This causes another hazard to predicting missing ratings. Concretely, it is impossible to estimate rating values for items 5 and 7. These problems will be solved by using a so-called *bit transformation* technique.

For instance, a rating matrix, where its rows indicate users, its columns indicate items and each cell is the rating which a user gives to an item. Suppose the ratings range in integer interval {1, 2, 3, 4, 5} where 5-most favorite and 1-most dislike, the sample rating matrix is shown in table 1.

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

**Table 1.** Rating matrix

Each item is “stretched” into 5 sub-items which are respective to 5 possible rating values {1, 2, 3, 4, 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. If a cell gets state 0, there is no one giving a rating on such cell yet. The bit rating matrix is shown in table 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | *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\_5 | 1 | 1 | 0 |

**Table 2.** Bit rating matrix

The frequent itemset, which is extracted from bit rating matrix, carries a so-called bit form *s* = (*item\_j*1*\_k*1, *item\_j*2*\_k*2,…, *item\_jm\_km*). Each element *item\_j\_k*, is defined as bit sub-item. Rating vector is also transformed into bit rating vector *u* = (*item\_j*1*\_k*1, *item\_j*2*\_k*2,…, *item\_jn\_kn*). Thus, it is completely simple to match the bit frequent itemset *s* with the bit rating vector *u*. For instance, the rating vector *u* = (*item* 1 = 3, *item* 2 = 5, *item* 3 = 2) is transformed into *u* = (*item*\_1\_3, *item*\_2\_5, *item*\_3\_2) whereas the bit frequent itemsets are *s*1 = (*item*\_1\_3, *item*\_2\_5, *item*\_3\_2, *item*\_4\_5) and *s*2 = (*item*\_1\_1, *item*\_2\_5, *item*\_3\_4). We find that itemset *s*1 is matched most with *u* and so, the additional item 4 in *s*1 is recommended to user with predictive value 5.

Now the aforementioned problems are solved but my algorithm should be enhanced. Suppose that the number of frequent itemsets is huge and each itemset has also a lot of items. When we match rating vector with frequent itemsets, there will be a boom of combinations that may cause computer system to be collapsed or consume much processing time. Therefore, I propose an enhancement method for matching purpose, based on *bit matching* technique.

**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, I 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. Table 3 shows 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* =(*item* 1 = 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\_5) is represented as *s*1 =(00100 00001 01000 00001). The frequent itemset *s*2 =(*item*\_1\_1, *item*\_2\_5, *item*\_3\_4) is represented as *s*2 =(10000 00001 00010 00000). In order to match *s*1 (or *s*2) with *u*, we need 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*.
* Otherwise (*s*1 *AND u*)≠ *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 00001)

From this bit set, it is easy to recognize that item 4 is recommended with predict value is 5 because the fifth bit of 4*th* cluster is set.

As a result, my algorithm will consist of three following steps:

* *Step* 1: Rating matrix is transformed into bit rating matrix.
* *Step* 2: Bit rating matrix is mined 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 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*(*…*) and *count*(*…*) be functions that transform item into bit set and counts the number of bit 1 (s) in bit set, respectively. 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 subsections 3.1, 3.2). Following is the pseudo-code like C language for my CF algorithm with note that rating matrix *D* and recommended item *r\_item* are input and output of algorithm, respectively. Steps 1 and 2 are specified at code lines 1-2. Step 3 is specified at lines 3-14. Lines 3-12 express how to determine optimal frequent itemset denoted *matched\_itemset*. Lines 13-14 express how to extract recommended items from *matched\_itemset*.

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

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

(03) *matched\_itemset = null*

(04) *max\_count =* –1

(05) *For each s S*

(06) *bs = bitset*(*u*) *AND bitset*(*s*)

(07) *If bs = bitset*(*u*) *&&*

(08) *count*(*bs*) *> max\_count then*

(09) *matched\_itemset = s*

(10) *max\_count = count*(*bs*)

(11) *End If*

(12) *End For*

(13) *r\_item = bitset*(*matched\_itemset*) *AND*

(14)(*NOT bitset*(*u*))

The second step, mining frequent itemsets from bit rating matrix, is the most important since it is the core of the proposed CF algorithm. Therefore the question “how to extract frequent itemsets from rating matrix” will be answered in section 3.

**3. Mining frequent itemsets**

My mining frequent itemsets method is based on the assumption: “*The larger the support of an item is, the higher it’s likely that such item occurs in some itemset*”. In other words, items with high support tend to combine together so as to form a frequent itemset. So my method is the heuristic algorithm called Roller algorithm. The basic idea is originated from white-wash task. Suppose you imagine that there is a wall and there is the dataset (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 will be added to the frequent itemset and the rolling task is continued until there is no item that meets minimum support. In the next time, all items in this frequent itemset are removed from the wall and the next rolling task will be performed to find out new frequent itemset.

My 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:
  + The support of the *ith* itemset denoted *support*(*ith* *itemset*) is accumulated by the 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.

Steps 3 and 4 are similar to 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* = (*i*1, *i*2,…, *im*) and *S* be a set of items and a set of frequent itemsets, respectively. Let *C =* (*c*1, *c*2,…, *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* with note that the set of items *I* and the set of frequent itemsets *S* are input and output of such function, respectively.

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

(02) *C = sort*(*C*)

(03) *i =* 1

(04) *While* (*C ≠ ∅*)

(05)  *c = first*(*C)*

(06)  *si =* {*c*}

(07)  *While* (*true*)

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

(09)  *S = S* ∪{*si*}

(10)  *C = C / S*

(11)  *break*

(12)  *Else*

(13)  *c = next*(*C*)

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

(15)  *If count*(*b*)≥ *min\_sup then*

(16)  *si = si* ∪{*c*}

(17)  *End If*

(18)  *End If*

(19) *End While*

(20)

(21) *i = i +*1

(22) *End While*

According to step 1, all frequent items are sorted at code lines 1-2. The *ith* itemset is initialized at lines 5-6, according to step 2. Lines 8-11 express that if current item *c* is the last one then, the *ith* itemset is completed and its items are removed from *C*, according to step 3. Line 13 expresses that the next item is picked and it becomes the current item, according to step 3. Line 14 aims to calculate the support of the *ith* itemset accumulated by the current item, according to step 4. Given lines 15-16, the current item is added to the *ith* itemset if it satisfies minimum support.

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 *bit mining* technique.

**3.1. Bit mining**

When rating matrix is transformed into bit rating matrix, item and itemset become cluster (sub-item) and bit set, respectively. The support of item or itemset is the number of bits whose values are 1 (s) in bit set. Given minimum support is 2, table 4 shows frequent items extracted from bit rating matrix shown in table 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | *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\_5 | 1 | 1 | 0 |

**Table 4.** 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\_5 =(110).

In step 2, the first itemset *s*1 is initialized as *Item*\_2\_5 that is the first item in the ordering.

*s*1 =(111) and *support*(*s*1) = *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\_5 are picked in turn and all of them satisfy minimum support.

* Picking *Item*\_1\_3: *s*1 = *s1* ⋃ {*Item*\_1\_3} =(111) *AND* (110) = (110) → *support*(*s*1) = 2.
* Picking *Item*\_3\_2: *s*1 = *s1* ⋃{*Item*\_3\_2} =(110) *AND* (110) = (110) → *support*(*s*1) = 2.
* Picking *Item*\_4\_5: *s*1 = *s1* ⋃{*Item*\_4\_5} =(110) *AND* (110) = (110) → *support*(*s*1) = 2.

Finally, the frequent itemset is *s*1 =(110) which includes *Item*\_2\_5, *Item*\_1\_3, *Item*\_3\_2and *Item*\_4\_5. We recognize that the bit set of frequent itemset, named *s*1 is accumulated by frequent item after each iteration. This makes algorithm run faster. The cost of counting bit set and performing bit operations isn’t significant.

**3.2. Improvement of Roller algorithm**

Roller algorithm may lose some frequent itemsets in case that some frequent items don’t have so high support (they are not excellent items) and they are in the last of descending ordering. So they don’t have many chances to join frequent itemsets. However they really contribute themselves into some frequent itemset because they can combine together to build up a 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 Roller algorithm is improved by modifying steps 3 and 4 as follows:

* *Step* 1 is the same to that of normal Roller algorithm.
* *Step* 2 is the same to that of normal Roller algorithm.
* *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 and satisfies minimum support. Such item called *pseudo-maximum support* item is chosen as the current item if it exists. If it does not exist, the current item is null.
* *Step* 4: The current item is added to the *ith* itemset if it is not null. Go back step 3.

Following is the pseudo-code like C language for improved Roller algorithm programmed as function *mining\_frequent\_itemsets* with note that the set of items *I* and the set of frequent itemsets *S* are input and output of such function, respectively. Code lines 9-19 find out pseudo-maximum support item according to step 3. Code line 24 adds such item to the *ith* itemset.

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

(02) *C = sort*(*C*)

(03) *i =* 1

(04) *While* (*C ≠ ∅*)

(05) *c = first*(*C*)

(06) *si =* {*c*}

(07) *C = C /* {*c*}

(08) *While* (*true)*

(09) *c = null*

(10) *pseudo\_maximum = –*1

(11) *For each item d ∈ C*

(12) *temp = bitset*(*si*) *AND bitset*(*d*)

(13) *If count*(*temp*) *≥ min\_sup &&*

(14) *count*(*temp*) *> pseudo\_maximum then*

(15)

(16) *c = d*

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

(18) *End If*

(19) *End For*

(20)

(21) *If c = null then*

(22) *break*

(23) *Else*

(24) *si = si* ∪{*c*}

(25) *End If*

(26) *End While*

(27) *S = S* ∪{*si*}

(28) *C = C / S*

(29)

(30) *i = i +* 1

(31) *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 and Discussion**

Database *Movielens* (GroupLens, 1998) is used for evaluation. It 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 Intel(R) Celeron(R) Dual-Core CPU N3050 @ 1.60GHz, RAM 4GB, Microsoft Windows 10 Pro 2015 64-bit, Java 8 HotSpot (TM) 64-bit Server VM. My proposed algorithm, named *GreenFall*, is compared to four other algorithms: neighbor item-based, neighbor user-based and SVD (Ricci, Rokach, Shapira, & Kantor, 2011, pp. 151-152). These algorithms are implemented upon the infrastructure *Hudup*. Note that Hudup is a middleware framework for e-commercial recommendation software, which supports scientists and software developers to build up their own recommendation algorithms with low cost, high achievement and fast speed. Hudup is accepted by European Project Space, whose trial version is available at <http://www.locnguyen.net/st/products/hudup>. Such algorithms follows pre-defined specifications so that evaluation mechanism of Hudup can test them according to standard metrics. There are 7 metrics (Herlocker, Konstan, Terveen, & Riedl, 2004, pp. 19-39) used in this evaluation: *MAE*, *MSE*, *precision*, *recall*, *F1*, *response* *time* and *setup* *time*. Response time metric, which measures speed of online recommendation process, is calculated in seconds. Setup time metric, which measures time consuming of offline modeling process (training process), is calculated in seconds.

MAE and MSE, abbreviations of absolute and mean squared errors, are average absolute and squared deviations between predictive ratings and users’ true ratings (Herlocker, Konstan, Terveen, & Riedl, 2004, pp. 20-21). Hence MAE and MSE are predictive accuracy metrics. The less they are, the high accuracy is. They are calculated by equations (1) and (2) in which *n* is the total number of recommended items while *pi* and *vi* are predictive rating and true rating of item *i*, respectively.

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |

Precision, recall and F1 are quality metrics that measure the quality of recommendation list – how much the recommendation list reflects user’s preferences. An item is relevant if its rating is larger than or equal to average rating. For example, within rating range {1, 2, 3, 4, 5}, the average rating is 3 = (1 + 5)/2. The larger quality metric is, the better algorithm is. An item is selective if it is recommended to users. Let *Nr* be the number of relevant items and let *Ns* be the number of selective items. Let *Nrs* be the number of items which are relevant and selective. According to equations (3) and (4), precision is the ratio of *Nrs* to *Ns* and recall is the ratio of *Nrs* to *Nr* (Herlocker, Konstan, Terveen, & Riedl, 2004, p. 23). In other words, precision is probability that selective item is relevant and recall is probability that relevant item is selective. F1 is regular combination of precision and recall, given equation (5).

|  |  |  |
| --- | --- | --- |
|  |  | (3) |
|  |  | (4) |
|  |  | (5) |

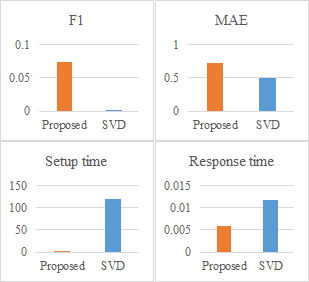
Table 5 shows the evaluation result on database Movielens 100K containing 100,000 ratings of 943 users on 1682 movies and are split into 5 folders. The setup time of neighbor item-based and user-based algorithms is zero because these algorithms are memory-based CF and they do not have internal modeling process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Proposed** | Item-based | User-based | SVD |
| MAE | **0.7279** | 0.5222 | 0.9430 | 0.5036 |
| MSE | **1.1737** | 0.6675 | 2.1895 | 1.0849 |
| Precision | **0.1326** | 0.0245 | 0.0013 | 0.0042 |
| Recall | **0.0524** | 0.0092 | 0.0005 | 0.0015 |
| F1 | **0.0740** | 0.0131 | 0.0008 | 0.0022 |
| Setup | **2.3532** | 0 | 0 | 120.64 |
| Response | **0.0058** | 10.918 | 9.5586 | 0.0117 |

**Table 5.** Evaluation result on database Movielens 100K

As a result, my algorithm is more effective than other algorithms when it gets high quality with metrics: precision, recall, F1. Its quality via F1 metric is 5.6 times, 98.5 times, 34 times better than item-based, user-based, and SVD, respectively. The reason is that it makes recommendations based on frequent itemsets which represent users’ interesting patterns. However its accuracy is lower than item-based and SVD with metrics: MAE and MSE. Its MAE metric is 1.4 times and 1.45 times worse than item-based and SVD, respectively. The reason is that it does not do arithmetic operations like addition and multiplication to estimate missing rating values. In general, accuracy is strong point of SVD. Author (Paterek, 2007) proposed three improved variants of SVD such as RSVD2 which adds of biases to the regularized SVD, SVD\_KNN which post-processes SVD with k-nearest neighbor similarity, and SVD\_KRR which post-processes SVD with kernel ridge regression. The author (Paterek, 2007, p. 42) evaluates these variants on Netflix Prize database, which results out that the MSE of methods: basic SVD, RSVD2, SVD\_KNN, and SVD\_KRR are 0.98262 = 0.9655, 0.90392 = 0.8170, 0.95252 = 0.9073, and 0.90062 = 0.8111, respectively. Although Netflix Prize database is different from Movielens database, the evaluation result of author (Paterek, 2007) asserts the preeminence of SVD with regard to accuracy in recommendation. Therefore quality is just preeminence of my algorithm.

Moreover it runs much faster than other methods. It responses 1870 times, 1637 times, 2 times faster than item-based, user-based and SVD, respectively. It is not dominant over SVD in response time but its modeling process consumes least time according to setup time metric which is 51 times faster than SVD. It consumes less memory because storage capacity of frequent itemsets is insignificant whereas SVD requires a large amount of memory to store decomposed matrices. In fact, mining frequent itemsets is faster than learning decomposed matrices. Figure 1 shows comparison between the proposed algorithm and SVD with regard to metrics MAE, F1, setup time, and response time given database Movielens 100K.



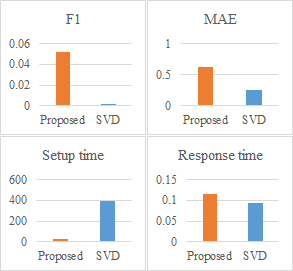
**Figure 1.** Comparison between proposed algorithm and SVD given database Movielens 100K

When database is enlarged with much more ratings, given database Movielens containing 1,000,000 ratings (1M), memory-based CF algorithms like user-based and item-based can not run whereas the proposed algorithm and SVD keep their execution stable. The proposed algorithm is dominant over SVD in quality of recommendation and modeling process. Table 6 shows the evaluation result on database Movielens 1M.

|  |  |  |
| --- | --- | --- |
|  | **Proposed** | SVD |
| MAE | **0.6231** | 0.2550 |
| MSE | **0.9790** | 0.4208 |
| Precision | **0.1048** | 0.0029 |
| Recall | **0.0342** | 0.0009 |
| F1 | **0.0516** | 0.0014 |
| Setup | **23.405** | 389.71 |
| Response | **0.1152** | 0.0946 |

**Table 6.** Evaluation result on database Movielens 1M

Its quality via F1 metric is 37 times better than SVD whereas its accuracy via MAE metric is 2.44 times worse than SVD. Its setup time is 16.65 times faster than SVD while its response time is 0.82 times approximated to SVD. The reason is that it does bitwise operations on both offline modeling process and online recommendation process; the cost of these bitwise operations is low, regardless of database volume. Figure 2 shows comparison between the proposed algorithm and SVD given database Movielens 1M. When comparing two evaluation results on database Movielens extended from 100K to 1M, recommendation quality is prominence of the proposed algorithm because its F1 scale is increased from 34 times to 37.08 times better than SVD. In general, the proposed algorithm is appropriate to fast and high quality recommendation applications when customers’ purchase patterns are concerned most. It can mine frequent itemsets regardless of missing rating values and so, the problem of sparse matrix aforementioned in the introduction section is solved.



**Figure 2.** Comparison between proposed algorithm and SVD given database Movielens 1M

If rating values are decimal numbers, it is required to convert decimal numbers to integer numbers in use of the proposed algorithm. For example, if rating values range in real interval [0, 1] then, values 0.0, 0.5, and 1.0 are converted to 0, 50, 100, respectively given integer interval [0, 1, 2,…, 100]. The bit representation reduces storage space. For instance, database needs at least 3\*10\*8 = 240 bits to store ratings for 3 items and 10 users because each value needs 1 byte = 8 bits whereas it is only required 3\*10\*5 = 150 bits with bit representation given integer interval {1, 2, 3, 4, 5}. In other words, saving storage space is a strong point of the proposed algorithm.

**5. Conclusion**

My CF algorithm is different from other model-based CF algorithms 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 I 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 advantages of enhancement technique with the concept of pseudo-maximum support.

In the future, I will propose another model-based CF algorithm which applies Bayesian network into inferring user interests. Such algorithm based on probabilistic inference will be compared with the mining approach in this paper so that we have an open and objective viewpoint about recommendation study.

**Acknowledgment**

This research is the place to acknowledge Madam Do, Phung T. M. – University of Information Technology, Vietnam National University and Sir Vu, Dong N. who gave me valuable comments and advices. These comments help me to improve this research.

**Reference**

Bashir, S., Jan, Z., & Baig, A. R. (2009, April 21). Fast Algorithms for Mining Interesting Frequent Itemsets without Minimum Support. *arXiv preprint, arXiv:0904.3319*, 1-25. Retrieved November 30, 2015, from http://arxiv.org/abs/0904.3319

Dong, J., & Han, M. (2006, September 18). BitTableFI: An efficient mining frequent itemsets algorithm. (H. Fujita, & J. Lu, Eds.) *Knowledge-Based Systems, 20*(4), 329-335. doi:10.1016/j.knosys.2006.08.005

GroupLens. (1998, April 22). *MovieLens datasets*. (GroupLens Research Project, University of Minnesota, USA) Retrieved August 3, 2012, from GroupLens Research website: http://grouplens.org/datasets/movielens

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques* (2nd Edition ed.). (J. Gray, Ed.) San Francisco, CA, USA: Morgan Kaufmann Publishers, Elsevier.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems (TOIS), 22*(1), 5-53.

Király, A., Gyenesei, A., & Abonyi, J. (2014, January 30). Bit-Table Based Biclustering and Frequent Closed Itemset Mining in High-Dimensional Binary Data. *Scientific World Journal, 2014*, 1-7. doi:10.1155/2014/870406

León, R. H., Suárez, A. P., & Feregrino-Uribe, C. (2010). A Compression Algorithm for Mining Frequent Itemsets. (C. A. Coello, A. Poznyak, J. A. Cadenas, & V. Azhmyakov, Eds.) *New Trends in Electrical Engineering, Automatic Control, Computing and Communication Sciences*, 485-499. Retrieved December 1, 2015

Li, H.-F., Ho, C.-C., Shan, M.-K., & Lee, S.-Y. (2006). Efficient Maintenance and Mining of Frequent Itemsets over Online. Data Streams with a Sliding Window. *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on.* *3*, pp. 2672-2677. Taipei: IEEE. doi:10.1109/ICSMC.2006.385267

Raja, A., & Raj, E. D. (2014, September). CBT-fi: Compact BitTable Approach for Mining Frequent Itemsets. *Advances in Computer Science : an International Journal, 3*(5), 72-76. Retrieved November 30, 2015, from http://www.acsij.org/documents/v3i5/ACSIJ-2014-3-5-535.pdf

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (2011). *Recommender Systems Handbook* (Vol. I). (F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor, Eds.) Springer New York Dordrecht Heidelberg London.

Shyu, M.-L., Haruechaiyasak, C., Chen, S.-C., & Zhao, N. (2005). Collaborative Filtering by Mining Association Rules from User Access Sequences. *Web Information Retrieval and Integration, 2005. WIRI '05. Proceedings. International Workshop on Challenges in* (pp. 128-135). Tokyo: IEEE. doi:10.1109/WIRI.2005.14

Song, W., Yang, Y., & Xu, Z. (2008, March 25). Index-BitTableFI: An improved algorithm for mining frequent itemsets. (H. Fujita, & J. Lu, Eds.) *Knowledge-Based Systems, 21*(6), 507-513. doi:10.1016/j.knosys.2008.03.011

Su, X., & Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. (J. Hong, Ed.) *Advances in Artificial Intelligence, 2009*.

**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).

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, 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, 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 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 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**

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 của 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**

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 [3]: *MAE, MSE, RMSE, độ chuẩn xác, độ bao phủ, F1, ARHR* và *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:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Giải thuật của tôi** | **Item-based** | **User-based** | **SVD** |
| MAE | *0.7241* | *0.5222* | *0.9319* | *0.5363* |
| MSE | *1.1640* | *0.6675* | *2.1664* | *1.1734* |
| Độ chuẩn xác | *0.1328* | *0.0245* | *0.0014* | *0.0041* |
| Độ bao phủ | *0.0523* | *0.0092* | *0.0005* | *0.0015* |
| F1 | *0.0739* | *0.0131* | *0.0008* | *0.0021* |
| ARHR | *0.0442* | *0.0043* | *0.0005* | *0.0016* |
| Thời gian xử lý | *0.0050* | *9.3706* | *8.3831* | *0.0176* |

**Bảng 5.1**. Kết quả đánh giá

Giải thuật của 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 của 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. 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.

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.