Generating (fuzzy) frequent itemsets by a bitmap based approach - the word's smallest frequent itemset miner

Janos Abonyi

University of Pannonia, Department of Process Engineering
POB. 158., H-8200, Veszprem, Hungary
abonyij@fmt.uni-pannon.hu

Mining frequent itemsets in databases is an important and widely studied problem in data mining research. The problem of mining frequent itemsets is usually solved by constructing a candidate set of items first, and then, identifying those items that meet requirement of the the frequent itemset. This paper proposes a novel algorithm based on the BitTable representation of the data. Data related to frequent itemsets are stored in spare matrices and simple matrix and vector multiplications are used to calculate the support of the potential n+1 itemsets. The main benefit of this approach is that only the BitTables of the frequent itemsets are generated. The concept is simple and easily interpretable that supports the compact and effective implementation (in MATLAB). An application example related to the BMS-WebView-1 benchmark data is presented to illustrate the applicability of the proposed algorithm.

Keywords: frequent itemsets, BitTable

1 Introduction

Mining frequent itemsets is an important and widely studied problem in data mining research []. The task of mining frequent itemsets usually is solved by constructing a candidate set of item-sets first, and then, identifying those itemsets that meet the frequent itemset requirement within this candidate set. Most of the early research mainly focused on pruning to reduce the candidate itemsets amounts and the time of database-scanning. A lot of algorithms adopt an Apriori-like candidate itemsets generation and support count approach which is a time-demanding process and needs a huge memory capacity. When the minimum support is small, and hence the number of frequent itemsets is very large, most algorithms either run out of memory or run over of allowed disk space due to the huge number of frequent itemsets. Recently new approaches were published using new ideas to solve the problems mentioned above.

Among the wide range of the developed algorithms this paper focuses to bitmap oriented algorithms. Among these approaches MAFIA (MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases Doug Burdick, Manuel Calimlim, Johannes Gehrke Department of Computer Science, Cornell University) and BitTableFI (Jie Dong, Min Han's (2007)) are the most well known. In the BitTableFI algorithm a special data structure, BitTable is used horizontally and vertically to compress database for quick candidate itemsets generation and support count. The algorithm can also be used in many Apriori-like algorithms to improve the performance. Experiments with both synthetic and real databases show that BitTableFI outperforms Apriori and CBAR which uses ClusterTable for quick support count. Wei Song at al. (2008) has further developed this concept resulting the Index-BitTableFI algorithm.

The motivation of this paper is to develop an extremely simple and easily implementable algorithm based on the bitmap-like representation of the frequent itemsets. The key idea is to store the data related to a given itemset in a binary vector. Hence, data related to frequent itemsets is stored in spare matrices and simple matrix and vector multiplications are used to calculate the support of the potential $k+1$ itemsets. The main benefit of this approach is that only the bitmaps of the frequent itemsets are generated based on the elementwise products of the binary vectors corresponding the building elements of the frequent itemsets. The concept is simple and easily interpretable that supports the compact and effective implementation of the algorithm (in MATLAB). Furthermore, when tables store fuzzy membership values, the algorithm can directly be used to generate fuzzy frequent itemsets.

The remaining of the paper is organized as follows. In Section 2 we briefly revisit the problem definition of frequent itemset mining by basic definitions. In Section 4, we show an Application for web usage mining. We conclude this study in Section 5.

2. The proposed algorithm

2.1 Definitions – matrix representation

Finding frequent itemsets can be formally stated as follows: let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of distinct literals, called items. Let $D = \{T_1, T_2, \ldots, T_N\}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A transaction T is said to support an itemset X if it contains all items of X, i.e., $X \subseteq T$. The support of an itemset X is the number of transactions that support X. An itemset is frequent if its support is greater than or equal to a user-specified minimum support threshold, denoted *MinSup*.

Table 1. Illustrative example for a transactional dataset and its binary representation.

tid	**items**
T_1	a, c, d
T_2	b, c, e
T_3	a, b, c, e
T_4	b, e

	Items				
Tid	1(a)	2(b)	3(c)	4(d)	5(e)
T_1	1	0	1	1	0
T_2	0	1	1	0	1
T_3	1	1	1	0	1
T_4	0	1	0	0	1
Sum	**2**	**3**	**3**	**1**	**3**

An illustrative example for D transactional database is shown in Table 1(a). The transactional database can be transformed into a bitmap-like matrix representation, where if an item *i=1,…,m* appears in transaction T_j *j=1,…,N*, the bit *i* of Bittable's j-th row is marked as one (as it is illustrated in Table 1).

Beside the analysis of classical transactional datasets the analysis of fuzzy data is also studied in this paper. In this case let $D = \{t_1, t_2, \ldots, t_N\}$ be a transformed fuzzy dataset of *N* tuples (data points) with a set of variables $Z = \{z_1, z_2, \ldots, z_n\}$ and let $c_{i,j}$ be an arbitrary fuzzy interval (fuzzy set) associated with attribute z_i in *Z*. From this point, we use the notation $\langle z_i : c_{i,j} \rangle$ for an *attribute-fuzzy interval pair*, or simply *fuzzy item*. An example could be $\langle Age : young \rangle$. For *fuzzy itemsets*, we use expressions like $\langle Z : C \rangle$ to denote an ordered set $Z \subseteq \mathrm{Z}$ of attributes and a corresponding set *C* of some fuzzy intervals, one per attribute, i.e. $\langle Z : C \rangle$.

Table 2. Example database containing membership values

	Items		
	$\langle \text{Balance} : \text{medium} \rangle$	$\langle \text{Credit} : \text{high} \rangle$	$\langle \text{Income} : \text{high} \rangle$
T_1	0.5	0.6	0.4
T_2	0.8	0.9	0.4
T_3	0.7	0.8	0.7
T_4	0.9	0.8	0.3
T_5	0.9	0.7	0.6

As can be seen, the classical and fuzzy data can also be stored in the same matrix structure, where the columns represent the items and the rows the transactions. The sum of the columns of this $\mathbf{B}^0_{N \times n}$ matrix represent the support of the $j = 1, \ldots, n$ items. (see the bottom of Table 1(b)) Hence, if b^0_j represents the j-th column, related to the i_j -th item, then the support of the i_j item can be easily calculated as

$$\sup(X_i) = \left(\mathbf{b}^0_i\right)^T \mathbf{b}^0_i / N$$

(in this case the result is given in percentage). Similarly, the support of an $X_{i,j} = \{i_i, i_j\}$ itemset can be easily calculated by a simple vector product of the two related bitvectors, since when both *i* and *j* items appear in a given transaction the product of the two related bits can represent the AND connection of the two items:

$\sup(X_{i,j}) = (\mathbf{b}_i^0)^T \mathbf{b}_j^0 / N$,

The matrix representation allows the effective calculation of all of the itemsets …

$$\mathbf{S}^2 = (\mathbf{B}^0)^T \mathbf{B}^0$$

where the i,j-th element of the $\mathbf{S}^2$ matrix represent the support of the $X_{i,j} = \{i_i, i_j\}$ itemset.

Similarly, the same equation can be used in case the $\mathbf{B}_{N\times n}^0$ matrix stores fuzzy membership values. In the literature, the fuzzy support value has been defined in different ways. Some researchers suggest the minimum operator as in fuzzy intersection, others prefer the product operator (as examples, see [8, 10]). They can be defined formally as follows: value $T_k(z_i)$ for attribute z_i, then the ***fuzzy support*** of $\langle Z : C\rangle^2$ with respect to *D* is defined as

$$FS(Z:C) = \frac{\sum_{k=1}^{N} \prod_{\langle z_i : c_{i,j}\rangle \in \langle Z:C\rangle} T_k(z_i)}{N}$$

A fuzzy support reflects how the record of the dataset supports the itemset. An itemset $\langle Z : C\rangle$ is called ***frequent*** if its fuzzy support value is higher than or equal to a user-defined minimum support threshold σ.

The following example illustrates the calculation of the fuzzy support value. Let $\langle X : A\rangle = [\langle \text{Balance} : \text{medium}\rangle \cup \langle \text{Income} : \text{high}\rangle]$ be a fuzzy itemset, the dataset shown in Table 2. The fuzzy support of $\langle X : A\rangle$ is given by:

$$FS(X:A) = \frac{0.5\cdot 0.4 + 0.8\cdot 0.4 + 0.7\cdot 0.7 + 0.9\cdot 0.3 + 0.9\cdot 0.6}{5} = 0.364$$

2.2 Mining Frequent Itemsets based on bitmap-like representation

2.2.1 Apriori algorithm for mining frequent itemsets

The best-known and one of the most commonly applied frequent pattern mining algorithms, *Apriori*, was developed by Agrawal et al. [1]. The name is based on the fact that the algorithm uses prior knowledge of frequent itemsets already determined. It is an iterative, breadth-first search algorithm, based on generating stepwise longer *candidate* itemsets, and clever pruning of non-frequent itemsets. Pruning takes advantage of the so-called *apriori* (or *upward closure*) *property* of frequent itemsets: all subsets of a frequent itemset must also be frequent. Each candidate generation step is followed by a counting step where the supports of candidates are checked and non-frequent ones deleted.

Given a user-specified *MinSup*, Apriori makes multiple passes over the database to find all frequent itemsets. In the first pass, Apriori scans the transaction database to count the support of each item and identify the frequent *1-itemsets* marked as L_1. In a subsequent *k*-th pass, Apriori establishes a candidate set of frequent *k-itemsets* (which are itemsets of length *k*) marked as Ck from L_{k-1}. Two arbitrary L_{k-1} join each other, when their first *k-1* items are identical. Then, the downward closure property is applied to reduce the number of candidates. This property refers to the fact that any subset of a frequent itemset must be frequent. Therefore, the process deletes all the *k-itemsets* whose subsets with length *k - 1* are not frequent. Next, the algorithm scans the entire transaction database to check whether each candidate *k-itemset* is frequent.

Generation and counting alternate, until at some step all generated candidates turn out to be non-frequent. A high-level pseudocode of the algorithm used for mining fuzzy frequent itemsets based on the *apriori* principle is given in Table 3.

Table 3. Algorithm *Mining Frequent Fuzzy Itemsets*
(minimum support σ, dataset *D*)

```
k = 1
    $(C_k; D_F) = Transform(D)$
    $F_k = Count(C_k, D_F, \sigma)$
while $|C_k| \neq 0$ do
    $inc(k)$
    $C_k = Generate(F_{k-1})$
    $C_k = Prune(C_k)$
    $F_k = Count(C_k, D_F, \sigma)$
    $F = F \bigcup F_k$
end
```

The subroutines are outlined as follows:

Transform(D): Generates a fuzzy database D_F from the original dataset D (see next section). At the same time the complete set of candidate items C_1 is found.

Count(C_k, D_F, σ): In this subroutine the fuzzy database is scanned and the fuzzy support of candidates in C_k is counted. If this support is not less than minimum support σ for a given itemset, we put it into the set of frequent itemsets F_k.

Generate(F_{k-1}): Generates candidate itemsets C_k from frequent itemsets F_{k-1}, discovered in the previous iteration *k-1*. For example, if $F_1 = \{\langle \text{Balance : high} \rangle, \langle \text{Income : high} \rangle\}$ then $C_2 = \{\{\langle \text{Balance : high} \rangle \cup \langle \text{Income : high} \rangle\}\}$.

Prune(C_k): During the prune step, the itemset will be pruned if one of its subsets does not exist in the set of frequent itemsets F.

2.2.2 The proposed algorithm for mining frequent itemsets

Before quick itemsets generation can be performed, frequent 1-itemsets should be found first. All non-frequent 1-itemsets is neglected to reduce the size of Bittable and improve the performance of the generation process, because all non-frequent 1-itemsets are unuseful. We leave out – based the decision on he given support threshold - this column of the Bittable and shift the remaining coloums left. If the colomn remains, the index of the original position is written into the actual i'th element of the first set of multidimensional array-set. This array-set shows the one-element frequent itemsets. Data related to frequent itemsets are stored in spare matrices and simple matrix and vector multiplications are used to calculate the support of the potential *n+1* itemsets. The main benefit of this approach is that only the BitTables of the frequent itemsets are generated. The concept is simple and easily interpretable that supports the compact and effective implementation (in MATLAB).

This means, this representation allows the effective calculation of the support of the candidate itemsets without the generation of the k+1 candidate itemsets.

After frequent 1-item set was generated, frequent 2-item set Bittable and baskets can be constructed. Through an n-circle cycle each column to the other of bittable are conjugated. Depending on the result column satisfies the support constraints, the column remains or is dropped. If remains, the original indexes of colums – got from the previous array-set - are put/added into the 2-itemsets proper array element.

$$\sup_k = \left(B^{k-1}\right)^T B^{k-1}$$

(which are itemsets of length k)

L_1 set of the …. The corresponding bittable"can be B^1, where only the columns of the frequent itemsets …

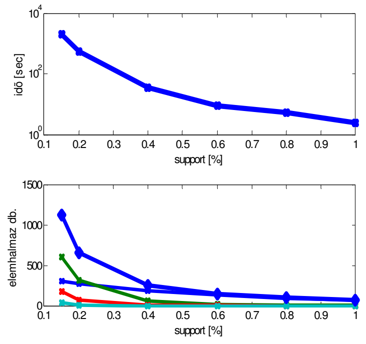
$$L_k = \left\{ i_j \middle| j, \sup\left(X^k_{\{i,j\}}\right) = \left(b^1_i\right)^T B^{k-1} / N > MinSup \right\}$$

$i = 1, \ldots n_1$, $j = 1, \ldots n_{k-1}$

3. Application example

www.gazelle.com BMS-WebView-1 59,602 tranzakció 497 termék 267 maximális kosárméret 2.5 átlagos látogatáshossz

Running Time for Generating Frequent Itemsets (sec)



Rendkívül egyszerű implementáció

„Transzparens"

Kisebb memóriaigény

Bináris – decimális

Ritka mátrixok (MATLAB-ban)

Az induló állapot előállítása után a szorzás-tömörítés-kosárkészítés egy menetben megtörténik.

Bit-sztring technikák bevethetők (csoportosítás, vizualizáció Tanimoto hasonlóságmérték alapján)

Conclusions

Compressing frequent itemsets into bittable saves a lot of memory, and bitwise AND operation is greatly faster than comparing each item in two frequent itemsets (as at Apriori).

By our results – based on BMS-WebView-1 (dataset) – we can declare, that our algorithm has several advanteges compared to others. It is a very simple implementation, easy to understand and extend its services is transparent, code generation is easy (our one is the shortest we found until now) has a very small memory capacity demand because of the bittable format, so easily can be used by medium-sized firms having 100 000 rows with 500 items on a commercial laptop in the case of sparse matrices using MATLAB even greater tasks can be solved easily after producing the 1-item sets, all other tasks are solved in one „package": the multiplication, - pruning, - basket producing is done in one step.

Bit-string techniques can be used (classification and visualization are simple, based on Tanimoto similarity measures)

initiate the „dinamic" support threshold, which enables more specific search, offers more precise measurement of data. This support threshold will be calculated on the basis of the dataset and will be given as an opportunity for the end-user. User-friendly solution.

the bit-string techniques helps linking classification and association tasks

some worthy of note (interesting, valuable) rules can be identified in the early phase of work

http://fuzzy.cs.uni-magdeburg.de/˜borgelt/software.html

Acknowledgement

The financial support from the TAMOP-4.2.2-08/1/2008-0018 (Élhetőbb környezet, egészségesebb ember - Bioinnováció és zöldtechnológiák kutatása a Pannon Egyetemen, MK/2) project is gratefully acknowledged.

Appendix: the word's smallest frequent itemset miner in MATLAB

```
function [items,Ai]=bittable(A, suppp)
[N,n]=size(A);
items{1}=find(sum(A,1)>=suppp)';
k=1; Ai{1}=A(:,items{1});
while ~isempty(items{k})
  k=k+1; Ai{k}=[]; items{k}=[];
  index=[0; find(sum(abs(diff((items{k-1}(:,1:end-1)))),2)~=0); size(items{k-1},1)];
    for i=1:length(index)-1
        v=[index(i)+1:index(i+1)];  m=Ai{k-1}(:,v)'*Ai{k-1}(:,v);
        m=triu(m,1); [dum1,dum2]=find((m)>suppp);
        for j=1:length(dum1)
         items{k}=[items{k}; [items{k-1}(v(dum1(j)),:) items{k-1}(v(dum2(j)),end) ] ];
         Ai{k}= [ Ai{k} Ai{k-1}(:,v(dum1(j))).*Ai{k-1}(:,v(dum2(j)))];
        end
    end
    [items{k},I]=sortrows(items{k}); Ai{k}=Ai{k}(:,I);
end
```

References

[1] Authors: Title, in Proceedings of …, place and date of edition, pp.

[2] Marci K. Sun, Devon Prince: Development Strategy for Mechanical Evaluation Formation, in Proceedings of 5th International Conference on Industrial Application on Computational Intelligence, Budapest, Hungary, July 10-13, 1980, pp. 159-165

János Abonyi at al. (2006) Adatbányászat – a hatékonyság eszköze. Computerbooks

Jiawei Han, Micheline Kamber (2001) Data Mining Concepts and Techniques Elsevier

Jie Dong, Min Han (2006) BitTableFI: An efficient mining frequent itemsets algorithm, Science Direct

Park J.S, Chen M.S., Yu S., An effective hash-based algorithm for mining associacion rules In Proc. 1995 ACM-SIGMOD pp. 175-186, San Jose, CA

R. Agrawal, R. Srikant, (1994), Fast algorithm for mining association rules in large databases, in: Proceedings of 1994 International Conference on VLDB, pp. 487–499.

Wei Song, Bingru Yang, Zhangyan Xu (2008) Index-BitTableFI: An improved algorithm for mining frequent itemsets; Knowledge-Based Systems journal homepage: elsevier.com/locate/knosys

Zijian Zheng, Ron Kohavi, Llew Mason (2001) Real World Performance of Association Rule Algorithms; ACM,.