Set Overlap in Mining of Frequent Itemsets

László Kovács

Department of Information Technology, University of Miskolc
kovacs@iit.uni-miskolc.hu

Abstract: An important module of soft computing methods is the set overlap operation. If a query set is tested with a large pool of source sets, the signature-based or the inverted-file methods are used to reduce the cost of operation. The paper introduces a modified version of the inverted-file approach, which yields in lowest costs for sparse input samples, i.e. where the number of records containing an element is relatively low.

1 Introduction

One of the main application areas of data mining is the association analysis. In the frame of association analysis, the association rules existing among the elements of a data pool should be discovered. This helps the users to learn new structural rules related to the data set.

A key process step in the analysis is the generation of the frequent itemsets as the rules are based on examples with high frequency value. The filtering of itemsets based on frequency is a useful module not only in discovering of association rules. The elimination of infrequent elements can be used in reduction of the problem size too. There are several methods to generate the frequent itemsets. The main methods are the Apriori-based solutions and the closed itemset algorithms.

The paper focuses on the algorithms based on closed itemset. In processing of the closed itemsets, the new closed itemsets are generated using the intersection operation. A new transaction itemset should be intersected with every existing itemsets. The paper addresses the problem of efficient generation of the set of intersection sets.

2 Mining of Frequent Itemset

To describe the discovery process for frequent itemsets some basic definitions are introduced:

U : universe

$u \in U$: item

$t_i \subseteq U$: transaction

$H = \{t_i\}$: history

$U_H = \cup t_i$: domain of the history

$a \subseteq U_H$: itemset

$f(a) = |\{t_i \mid a \subseteq t_i\}|$: frequency of the itemset

An association rule is denoted by

$$X \Rightarrow Y$$

where

$$X,Y \subseteq U\_\{H\}$$

$$support(X,Y) = f(X \cup Y) > \varepsilon_1$$

and

$$confidence(X,Y) = f(X \cup Y) / f(X) > \varepsilon_2$$

are met. Here ε_1 and ε_2 are the thereshold values.

One of the prerequisites for an $X \Rightarrow Y$ association rule is that the frequency of $X \cup Y$ should be higher than a given threshold. Thus one the key elements in discovering association rules is the discovery of the frequent itemsets.

The basic variant of finding the frequent itemsets is the Apriori algorithm [7]. The algorithm employs an iterative approach to discover the frequent itemsets of increasing sizes. First, the frequent 1-itemsets are found, then the 2-itemsets and so on. To reduce the cost, the following property is used: every subsets of a frequent itemsets are frequent too. This means that a frequent itemset at level *k* can not contain an infrequent itemset from level *k-1*. A detailed description of the Apriori algorithm can be found for example in [7].

Beside the Apriori algorithm there exist some other approaches too. Our interest is focused on the algorithm based on closed itemset lattice, as the lattice structure provides some extra advantages, among others it can be used to discover semantical or generalization relationship among the frequent itemsets. A good summary on the theory of closed itemsets can be found among others in [6].

The term of closed itemset is related to the area of concept lattices or Galois lattices [6] which is used in many application areas to represent conceptual hierarchies among the objects in the underlying data. The field of Formal Concept Analysis [8] introduced in the early 80ies has grown to a powerful theory for data analysis, information retrieval and knowledge discovery. A *K* context is a triple *K(G,M,I)* where *G* and *M* are sets and *I* is a relation between *G* and *M*. The *G* is

called the set of objects and *M* is the set of attributes. The cross table *T* of a context *K(G,M,I)* is the matrix form description of the relation *I*:

$$t_{i,j} = \begin{cases} 1, & \text{if } g_i I a_j \\ 0 & \text{otherwise} \end{cases}$$

where $g_i \in G$, $a_j \in M$.

Two Galois connection operators are defined. For every $A \subseteq G$:

$$f(A) = A' = \{ a \in M \mid g \, I \, a \text{ for } \forall \, g \in A \}$$

and for every $B \subseteq M$

$$g(B) = B' = \{ g \in G \mid g \, I \, a \text{ for } \forall \, a \in B \} \qquad (3)$$

The Galois closure operator is defined by

$$h = f \circ g$$

and

$$A'' = h(A)$$

is the Galois closure of *A*. The pair *C(A,B)* is a closed itemset of the *K* context if

- $A \subseteq G$
- $B \subseteq M$
- $A' = B$
- $B' = A$
- $A = h(A)$

hold true. In this case *A* is called the extent and *B* is the intent of the *C* closed itemset. It can be shown that for every $A_i \subseteq G$,

$$(\cup_i A_i)' = \cap_i A^{'i}$$

and similarly for every $B_i \subseteq M$,

$$(\cup_i B_i)' = \cap_i B'_i$$

holds true.

Considering the Φ set of all concepts for the *K* context, an ordering relation can be introduced for the set of closed itemsets in the following way:

$$C_1 \leq C_2$$

if

$$A_1 \subseteq A_2$$

where C_1 and C_2 are arbitrary closed itemsets. It can be proved that for every (C_1, C_2) pair of closed itemsets, the following rules are valid:

$$C_1 \vee C_2 \in \Phi$$

and

$$C_1 \wedge C_2 \in \Phi$$

Based on these features $(\Phi, \cap)$ is a lattice, called closed itemset lattice. According to the Basic Theorem of closed itemset lattices, $(\Phi, \cap)$ is a complete lattice, i.e. the infinum and suprenum exist for every set of closed itemsets. The following rules hold true for every closed itemsets:

$$\vee_i (A_i, B_i) = (\cap_i A_i, (\cup_i B_i)'')$$

$$\wedge_i (A_i, B_i) = ((\cup_i A_i)'', \cap_i B_i)$$

The structure of a Galois lattice is usually represented with a Hasse diagram. The Hasse diagram is a special directed graph. The nodes of the diagram are the closed itemsets and the edges correspond to the neighbourhood relationship among the itemsets. If C_1, C_2 are itemsets for which

$$C_1 < C_2$$

$$! \exists C_3 \in (\Phi, \leq) : C_1 < C_3 < C_2$$

hold true then there is a directed edge between C_1, C_2 in the Hasse diagram. In this case, the C_1 and C_2 concepts are called neighbour concepts.

According to the Apriori property, there is no need to explicit generate all frequent itemsets, as every subset of a frequent itemset are frequent. Thus, only those frequent itemsets are located which have no frequent container itemset. These frequent itemsets are called maximal frequent itemsets. The relationship between the frequent itemsets and the maximal frequent closed itemsets is based on the following properties [6]:

- all sub closed itemsets of a frequent closed itemset is frequent
- all sup closed itemsets of an infrequent closed itemset is infrequent
- the set of maximal frequent itemsets is identical to the set of maximal frequent closed itemsets
- the support of a frequent not closed itemset is equal to the support of the smallest frequent closed itemset containing the itemset in question.

In the different lattice building algorithms, the extension of the input database with one new transaction may yield in extension of the lattice with several new itemset nodes. At some other nodes, the frequency counter should be incremented by one. All of the nodes affected during the update operation, are subsets of the

new itemset. Thus, a key step in updating the lattice is to define these subsets as the intersection with the existing itemsets of the lattice.

The performance bottleneck of frequent itemsets generation lies in the following problems:

- generating all of the intersections
- elimination of redundant itemsets
- insertion of the new itemset into the lattice

All of these steps have a high computation costs. Although the proposals usually do not contain cost formulas, the experimental shows [1][2] that the cost functions may be $O(2^N)$, as the number of possible itemsets is an exponential function of the number of elements.

3 Set Overlap Algorithms

For description of this problem area the following new denotations are introduced:

s_i : a set indexed by i

$S = \{s\}$: the set of sets

$D = \cup s_i$: the domain set of the set elements

q : a query set

$sup_e \subseteq S$: the set of sets containing the element $e \in D$

N : number of sets in S

H : number of elements in D

M : the average size of a set

The task is to determine the intersection sets $I_{q,S}$ which contains the results of intersections with every elements of S:

$$I_{q,S} = \{ q \cap s_i \mid s_i \in S \}$$

Taking the basic approach, the sets are processed sequentially and the results are merged into the result set. The main steps of the algorithm are the followings:

```
I = {}
foreach s_i ∈S begin
        r = q ∩ s_i
        I = I ∪ r
```

end

The cost of the algorithm can be calculated with

$$C_1 = O(M (N + R \cdot \log R))$$

where

R : number of sets in I

The problem of set overlap was mainly investigated in the database implementation field, related to the problem of efficient joins of set valued attributes[3]. In the literature, for the set overlap problem two basic approach can be found. The signature-based indexing[1] and the inverted-file[1] approach.

The signature is a bitmap used to represents sets in the form of a bitmap matrix. The rows of the matrix corresponds to the elements and the columns is assigned to the sets. The $b_{i,j}$ bit is set to *1* if the *j*-th set contains the *i*-th element. To reduce the size of the signature matrix, the number of rows is usually significantly reduced. This means that not every element will have an own row in the matrix. Thus, several elements may share the same bitmap mask. In this case a hash function is used to assign the bitmap mask to the elements. Considering the lookup function, in the case of the reduced matrix, the search process should be performed in two distinct steps. First, the signature of the query set is generated and the sets containing this mask are retrieved. This step is called filtering step. In the second phase, the resulted sets are tested to decide whether the candidate is a valid overlap or not. The candidates having no overlap with the query set, are called false drops.

In the case of inverted file approach, a list of container sets is generated for every element. This list contains the set identifiers of the sets containing the element. This list is called an inverted list. The size of the inverted-file depends on the average number of sets containing an element. In the case of a query, the processing is performed in the following steps. First, the elements of the query set are parsed. The inverted lists of the elements are retrieved from the inverted file. This can be done very efficiently as the inverted files are usually indexed by the element values. In the second step, the yielded inverted lists are processed to calculate the result set. An important cost parameter of this method is the average length of the inverted lists denoted by *L*.

In recent comparison studies [1] the inverted file was proven significantly faster than signature-based indexes. The main reasons for difference, according to [1], are the followings:

- The inverted file is more suitable for real-life applications where the sets are sparse and the domain cardinality is large.
- The signature-based approach generates usually a large number of false drops.

- The inverted file uses an exact indexing instead of the hashing approach of the signature file.

Based on these considerations, the inverted file method was selected for implementation in our system. The methods mentioned in the literature could not applied directly in our system, as we had different goals and requirements. The special characteristics of our system can be summarized in the following points:

- the intersections itself are of interest not the transactions having intersections,
- all of the new possible intersections should be retrieved,
- the intersections already stored in the pool, should be neglected,
- the transactions usually have small number of elements,
- the element domain is very large.

To meet these requirements, a modification was performed on the basic inverted-file algorithm. To retrieve the requested result set, the following steps should be done with the inverted lists:

- generating all possible intersection groups, theoretically the number of these groups is 2^M,
- performing the intersections for every groups with a cost value of $M*L$.

This method yields a total cost of

$$C_2 = O(M \cdot L \cdot \log H + 2^M \cdot M \cdot L)$$

In our approach an extra reduction was invented, to reduce number of performed intersections. In this approach, a special binary tree is applied to manage the intersections. Every layer of the tree is assigned to one element from the query interface. Every nodes symbolize the different partial selection possibilities. The leaf nodes are the subsets of the query set. Thus the number of the leaves is equal to

$$2^M$$

A node stores the elements meeting the selection criteria. An edge from the parent node to the child node denotes the addition of the next element to the selection possibilities. Each edge is assigned to the inverted list of the actual element. The edge to the right-hand child is assigned to the set in the inverted list (positive sign), the left-hand child is assigned to the complementary set of the inverted list (negative sign). The content of the parent node and of the edge determines the content of the child. The generation rule can be summarized in the following:

- positive parent and positive edge: positive sign and intersection operation
- positive parent and negative edge: positive sign and minus operation

- negative parent and positive edge: positive sign and minus operation
- negative parent and negative edge: negative sign and union operation

The main benefits of this approach is that is can reduce the number of performed operations and can cut a branch of the tree if one internal node gets empty. This method yields a total cost of

$$C_3 = O(M \cdot L \cdot \log H + 2^M \cdot L)$$

The following table shows some test result on this method.

N	C_1	C_2	C_3
20000	3.0	0.012	0.006
40000	6.0	0.027	0.014
80000	11.0	0.067	0.037
160000	20.0	0.19	0.13

Table 1

Comparison of the cost values

The first test results show that the inverted file approach provides a superior performance regarding the retrieval operation. The proposed modification yields in further improvement if the elements are relatively rare in the training pool. For the cases where

$$L / N < 0.001$$

the proposed method gives the best cost value. In our project, the proposed algorithm will be built into a module for mining frequent itemsets.

References

[1] Mamuloulis N., Efficient Processing of Joinsd on Set-valued Attributes, Proc. of SIGMOD, 2003, pp.

[2] Jampani R. and Pudi V.: Using prefix trees for efficiently computing set joins

[3] Ramassamy K. and Patel J. and Naughton J. and Kaushik R., Set containment joins: The good, the bad and the ugly, Proc. of VLDB, 2000

[4] Saragawi S and Kirpal A, Efficient set joins on similarity predicates, Proc. of ACM SIGMOD, 2004

[5] Hellerstein J. and Pfeffer A., The RD-tree: An index structure for sets

[6] Pei J. and Han J. and MAo R., CLOSET: An efficient algorithm for mining frequent closed itemsets, 2001

[7] Agrawal R. and Srikant R., Fast algorithms for mining association rules, Proc. of VLDB, 1994

[8] Ganter B. and Wille R., Formal concept analysis, Springer Verlag, 1999