Volume 2, Issue 7, July 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Strategy for Initiate Support Check into Frequent Itemset Mining

G. Hari Prasad[1]
M.Tech, Dept of CSE
CREC, Tirupati, India

J.Nagamuneiah[2]
Assoc.Professor,Dept of CSE
CREC, Tirupati, India.

***Abstract*— Mining Association Rules means that, given a set of sales transactions, to discover all association among items such that the presence of some items in transaction will imply the presence of other items in the same transaction. The mining of association rules can be mapped into the problem of discovering large item sets. Interesting patterns often occur at different levels of support. The classic association mining based on a uniform minimum support, such as Apriori, either misses interesting patterns of low support or suffers from the bottleneck of itemset generation caused by a low minimum support. A better solution lies in exploiting *support constraints*, which specify what minimum support is required for what item sets, so that only the necessary item sets are generated. The support constraints are "pushed" into the Apriori item set generation so that the "best" minimum support is determined for each itemset at runtime to preserve the essence of Apriori. This strategy is called Adaptive Apriori.**

***Keywords*— Data Mining, association rules, domain-specific constraints, classification.**

I. INTRODUCTION

Nearly all later frequent itemset minings rely on Apriori as a basic pruning strategy. Constraints other than the minimum support are considered in some other research papers. However, none of these approaches considers pushing support constraints like in the paper. The correlation approach considers the support requirement relative to the independence assumption, but not general support constraints or constraint pushing. Instead of abandoning the support requirement, our approach is to make the requirement more realistic by allowing it different for different itemsets.
This specification is unnatural for three reasons.

1. The MIS of individual items has to reflect the minimum support of unseen itemsets at the specification time.
2. In some applications, the user may have a minimum support for an itemset as a single concept, e.g., {white, male}, but not for individual items in the itemset (e.g., white or male). This "minimum itemset support" is usually lower than the minimum item support.
3. Different minimum supports cannot be specified for two itemsets, like {white, male} and {white, male, grad} if a common item has the lowest MIS, like white. We overcome these difficulties by specifying the minimum support directly for itemsets. We will show that our specification can model the MIS specification, but the converse is not true.
With the background information furnished above we now proceed to the technical part of this project which begins with the definition of the problem and in-depth analysis which is followed by the design and implementation and finally concluded this through testing of the coded application.
"Market – Basket Analysis" is used to determine which products sell together. It assumes, we have some large number of items. Example: "bread, milk". Customers fill their baskets with some subset of the items and we get to know what items
people buy together. To explore this we use "Association Rules Mining".

1.1 Association Rules:

Association rules are statements of the form $\{X_1,X_2,\ldots,X_n\}$ => Y, meaning that if we find all of $X_1,X_2,\ldots,X_n$ in the Market – Basket, then we have a good chance of finding 'Y'. The probability of finding 'Y' for us to accept this rule is called the "confidence" of the rule. We normally would search only for rules that had confidence above a certain threshold.
For example, we can find a rule like
{milk,butter} => bread
because a lot of people buy bread.

1.2 Formal Model:

Let $X=I_1,I_2,\ldots,I_m$ be a set of binary attributes, called items. Let T be a set of transactions. Each transaction 't' is represented as a binary vector, with t[k] = 1 if 't' bought the item I_k, and t[k] = 0 otherwise. Let X be a set of some items in X. We say that a transaction 't' satisfies X if for all items I_k in X, t[k] = 1.
By an association rule, we mean an implication of the form
X => I_j is satisfied in the set of transactions T with the confidence factor
$0 \le c \le 1$ iff atleast c% of transactions in T that satisfy X also satisfy I_j. We will use the notation X => I_j | c to specify that the rule X => I_j has a confidence factor of c.
Given the set of transactions T, we are interested in generating all rules that satisfy certain additional constraints of two different forms.

© 2012, IJARCSSE All Rights Reserved

II. Syntactic Constraints

These constraints involve restrictions on items that can appear in a rule. For example, we may be interested only in rules that have a specific items I_x appearing in the consequent, or rules that have a specific item I_y appearing in the antecedent. Combinations of the above constraints are also possible – we may result all rules that have items from some predefined itemset X appearing in the consequent, and items from some other itemset Y appearing in the antecedent.

1. **Support Constraints:** These constraints concern the number of transactions in T that support a rule. The support for a rule is defined to be the fraction of transactions in T that satisfy the union of items in the consequent and antecedent of the rule.
2. **Confidence:** Confidence is a measure of the rule's strength, support corresponds to statistical significance. Besides statistical significance, another motivation for support constraints comes from the fact that we are usually interested only in rules with support above some minimum threshold for business reasons. If the support is not large enough, it means that the rule is not worth consideration or that it is simply less preferred.

2.1 Apriori:

Apriori algorithm is used for mining frequent itemsets for "boolean association rules". Apriori employs an iterative approach known as level-wise search, where k-itemsets are used to explore (k+1) itemsets.

First, the set of frequent 1-itemset is found. This set is denoted as L_1. L_1 is used to find L_2, the set of frequent 2 itemsets, which is used to find L_3 and so on. The finding of each L_k requires on full scan of the database.

Drawbacks:

- Misses interesting patterns of low support
- Suffers from the bottleneck of itemset generation caused by a low minimum support

2.3 Adaptive Apriori:

Adaptive Apriori is used to push SCs following the "dependency chain" of itemsets in the itemset generation in Apriori. This dependency is best described by a schema enumeration tree. In a *schema enumeration tree*, each node (except the root) is labeled by a bin B_i. A node *v* represents the schema given by the labels $B_1 \ldots B_k$ along the path from the root to *v*. The ordering of nodes in a schema enumeration tree is determined dynamically on a per-node basis to achieve a certain optimality of constraint pushing.

Advantages:

Adaptive Apriori uses support constraints, which specify what minimum support is required for what itemsets, so that only the necessary itemsets are generated.

The support constraints are "pushed" into the Apriori itemset generation so that the "best" minimum support is determined for each itemset at runtime to preserve the essence of Apriori.

III. Association Rules

Data mining has recently attracted tremendous amount of attention in data and database research because of its applicability in many areas, including decision support, marketing strategy and financial forecast. Our capabilities to both generating and collecting data have been increasing rapidly. The wide spread use of bar codes for most commercial products, the computerization of many business and government transaction and the advances in data collection tools have provided us with huge amounts of data. This explosive growth in data and databases has generated an urgent need for new techniques and tools that can intelligently and automatically transform the processed data into useful information and knowledge. Consequently, data mining has become a research are with increasing importance.

One of the most important data-mining problems is mining association rules.

Association rule mining finds interesting associations and/or correlation relationships among large set of data items. Association rules show attribute value conditions that occur frequently together in a given dataset. A typical and widely-used example of association rule mining is Market Basket Analysis.

For example, data are collected using bar-code scanners in supermarkets. Such 'market basket' databases consist of a large number of transaction records. Each record lists all items bought by a customer on a single purchase transaction. Managers would be interested to know if certain groups of items are consistently purchased together. They could use this data for adjusting store layouts (placing items optimally with respect to each other), for cross-selling, for promotions, for catalog design and to identify customer segments based on buying patterns.

Association rules provide information of this type in the form of "if-then" statements. These rules are computed from the data and, unlike the if-then rules of logic, association rules are probabilistic in nature.

One of the tasks is to derive a set of strong association rules in the form

" $A_1 \Lambda \ldots .. \Lambda A_m \Rightarrow B_1 \Lambda \ldots .. \Lambda B_n$ "

Where A_i (for i ε {1,….,m}) and B_j (for j ε {1,….,n}) are sets of attributes values, from the relevant data sets.

Frequent Itemsets:

In many situations, we only care about association rules involving sets of items that appear frequently in baskets. For example, we can run a good marketing strategy involving items that no one buys anyways. Thus, much data mining starts with the assumption that we only care about sets of items with high "support" that is., they appear together in many baskets. We then find association only involving a high-support set of items (i.e. $\{ X_1, X_2, \ldots, X_n, Y \}$) must appear in at least a certain percent of baskets, called the support threshold.

Frequent Itemset Mining:

We consider the term "frequent itemset" for "a set 'S' that appears in at least fraction 'S' of the baskets", where 'S' is some chosen constant, typically 0.01 or 1%.

We assume data is too large to fit in main memory. It is either stored RDB, say as a relation baskets (BID, item) or as a flat file of records of the form (BID, item1,…, itemn).

© 2012, IJARCSSE All Rights Reserved

If a set of items 'S' is frequent (i.e. appears in at least fraction 'S' of the baskets), then every subset of 'S' is also frequent.

To find frequent itemsets, we can:

- Proceed level wise, finding first the frequent itemsets (sets of size 1), then the frequent pairs, the frequent triples, etc.
- Find all maximal frequent itemsets (i.e. set 'S' such that no proper
  super set of 'S' is frequent) in one pass or few passes.

The main problem with association rule mining is finding frequent itemsets and their support.

3.1 Rule generation

Most data mining tools generate their findings in the format of "if … then" rules. Here's an example of a data mining process that discovers buying patterns of customers.

Eg:
If buys (CPU, monitor)
Then
Buys (speakers)

3.2 Basic association rule algorithm

- In the first pass, the support of each individual item is counted, and the large ones are determined.
- In each subsequent pass, the large itemsets determined in the previous pass is used to generate new itemsets called candidate itemsets.
- The support of each candidate itemset is counted, and the large ones are determined.
- From the large itemsets found, frame the Association Rules present.

3.3 Apriori:

Apriori algorithm is used for mining frequent itemsets for "boolean association rules". Apriori employs an iterative approach known as level-wise search, where k-itemsets are used to explore (k+1) itemsets.
First, the set of frequent 1-itemset is found. This set is denoted as L_1. L_1 is used to find L_2, the set of frequent 2 itemsets, which is used to find L_3 and so on. The finding of each L_k requires on full scan of the database.
To improve the efficiency of the level-wise generation of frequent itemsets, we use Apriori property, which is used to reduce the search space.

In order to use the Apriori property, all non-empty subsets of a frequent itemset must also be frequent.

By definition, if an itemset 'I' doesnot satisfy the minimum support threshold, min_sup, then 'I' is not frequent, that is, P(I) < min_sup. If an item 'A' is added to the itemset 'I', then the resulting itemset (i.e.I U A) cannot occur more frequently than 'I'. Therefore, I U A is not frequent either, i.e. P(I U A) < min_sup.

"How is the Apriori property used in the algorithm?"
A two step process is followed, consisting of join and prune actions.

1. Join: To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the jth item in l_i (eg.. $l_1[k-2]$ refers to the second to the last item in l_1). By convention, Apriori assumes that items within a transaction or itemset are stored in lexicographic order. The join, L_{k-1} -----L_{k-1}, is performed, where members of L_{k-1} are joinable if their first (k-2) items are $(l_1[2] = l_2[2])\Lambda\ldots.\Lambda(l_1[k-2] = l_2[k-2])\Lambda(l_1[k-1] < l_2[k-1])$. The condition $(l_1[k-1] < l_2[k-1])$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $l_1[1]\ l_1[2]\ldots.l_1[k-1]\ l_2[k-1])$.

2. Prune: C_k is super set of L_k, i.e. its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. a scan of the database to determine the count of each candidate in C_k would resulting the determination of L_k. C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.
Algorithm:

Apriori: Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input: Database D, of transactions; minimum support threshold, min_sup

Output: L, frequent itemsets in D

Method:

1. L_1 = find frequent_1-itemsets(D) ;
2. for(k=2; $L_{k-1}\neq\varphi$;k++)
3. C_k = apriori_gen(L_{k-1} , min_sup);
4. for each transaction t ε D { // scan D for counts
5. C_t = subset(C_k , t); // get the subsets of t that are candidates
6. for each candidate C ε C_t
7. C.count++ ;
8. }
9. $L_k = \{ C \varepsilon C_k \mid C.count \geq min\_sup \}$
10. }
11. return $L = U_k L_k$;

procedure apriori_gen (L_{k-1}: frequent (k-1)-itemsets; min_sup: minimum support threshold)

1. for each itemset l_1 ε L_{k-1}
2. for each itemset l_2 ε L_{k-1}
3. if $(l_1[1] = l_2[1])\ \Lambda\ (l_1[2] = l_2[2])\ \Lambda\ldots\ldots\ \Lambda\ (l_1[k-2] = l_2[k-2])\ \Lambda\ (l_1[k-1] < l_2[k-1])$ then
4. $C = l_1$ -----l_2
5. if has_infrequent_subset(C , L_{k-1}) then

© 2012, IJARCSSE All Rights Reserved

6. delete C ; // prune step: remove unfruitful candidate
7. else add C to C_k ;
8. }
9. return C_k ;

procedure has_infrequent_subset(C: candidate k-itemset; L_{k-1}: frequent (k-1)-itemsets) ; // use prior knowledge

1. for each (k-1)-subset S of C
2. if S doesnot belongs L_{k-1} then
3. return TRUE ;
4. return FALSE ;

We need to scan the database of transactions to compute the support for an itemset but some itemsets can be discarded before the scan. A candidate K-sized itemset, I1, can be discarded before scanning the transactions if any of its K-1 sized subsets do not appear in the list of K-1 sized itemsets.

For example, suppose the itemsets from K = 3 are
K3 = {{a, b, d}, {a, b, f}, {a, d, f}, {b, d, f}, {a, c, d}, {c, e, g}, {c, e, h}, {c, g, h}}

The candidate 4-sized itemset {c, e, g, h} can be discarded because a subset, {e, g, h}, does not appear in K3. In other words, for {c, e, g, h} to have min Support, all of its subsets must have minSupport but the subset {e, g, h} does not, for if it did, it would appear in K3.

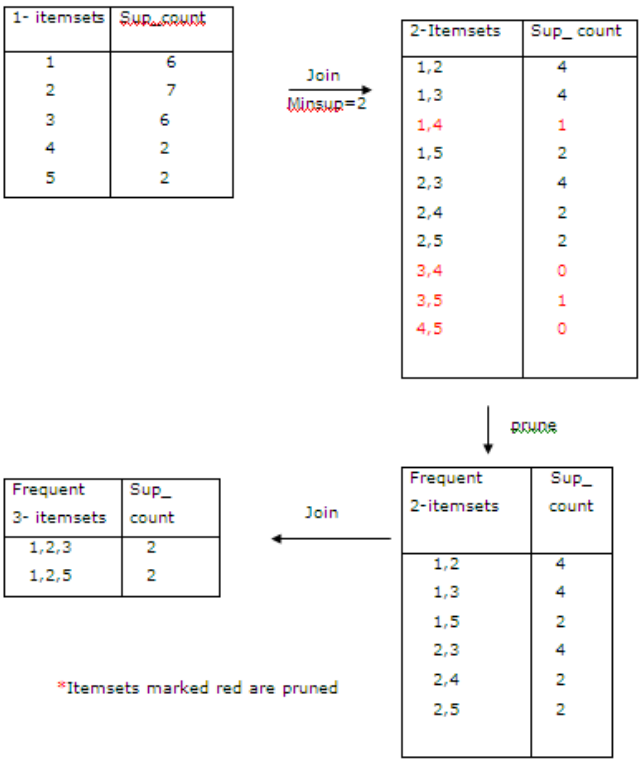
Consider the generation of candidate 4-sized itemsets from joining 3-sized itemsets. Here, we assume that the items within an itemset are stored in lexicographic order. Suppose we have two itemsets, the first denoted by I1 and the second by I2. Denote the first item in I1 by I1(1) and the second by I1(2), etc, and similarly for I2. Itemsets I1 and I2 can be joined to generate a 4-sized itemset provided.

Let us consider an example to find frequent itemsets.
Support Specification:

TID	List of item_Ids
T1	I1, I2,I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

The formula to find the support of each item is
SUPPORT (A=>B) = no of tuples containing both A & b

Total no of tuples



3.4 Support Constraints Specification:

It is a way to specify general constraints on minimum support. Minimum support range is [0..1]

Support Specification:

The task of support specification is to specify the minimum support for each itemset. Our approach is to partition the set of items into bins, denoted as B_j, such that items that need not be distinguished in the specification are in the same bin. To specify the minimum support for itemsets, we will specify the minimum support for schemas.

Support Constraints:

A support constraint (SC) has the form $SC_i(l_1,\ldots,l_s) \geq \theta_i$ (or $SC_i \geq \theta_i$), $S \geq 0$. Each l_j is either a bin or a variable for bins. θ_i , called a minimum support, is a function over $l_1,\ldots,l_s$ and returns a real in [0..1]. An SC is ground if it contains no variable, otherwise, non-ground. A non-ground SC can be instantiated to a ground SC by replacing each variable with a bin. A support specification is a non-empty set of SCs.

Frequent Itemsets:

An itemset 'I' matches a ground $SC_i \geq \theta_i$ in the open interpretation if 'I' contains (atleast) one item from each bin in SC_i and these items are distinct. An itemset 'I' matches a ground $SC_i \geq \theta_i$ in the closed interpretation. If 'I' contains one item from each bin in SC_i and these items are distinct, and 'I' contains no other items. An itemset 'I' matches a non-ground SC if 'I' matches some instantiation of the SC.

© 2012, IJARCSSE All Rights Reserved

Association Rules:

For each pair of frequent itemsets I and I′ such that I $\underline{C}$ I′

- if sup(I′) / sup(I) ≥ minconf, Type-I association rule I → I′ - I is constructed.
- if sup(I′) / sup(I′ - I) ≥ minconf, Type-II association rule I′ - I → I is constructed.
- if sup(I′) / sup(I′ - I) ≥ minconf, and I′ - I is frequent, Type-III association rule I′ - I → I is constructed.

Now let us consider an example for constructing the bins according to the support constraints given.
Let us consider the transactions and support specification by specifying some items in an itemset. Each item is represented by an integer from 0 to 8.

database

TID	Items
100	0,2,7
200	0,4,7,8
300	2,4,5,7,8
400	1,2,4,7,8
500	2,4,6,7,8

Now, let us consider four support constraints

A specification

$SC_1(B_1,B_2)>=0.2$
$SC_2(B_3)>=0.4$
$SC_3(B_2)>=0.6$
$SC_0()>=0.8$

Each bin Bi contains a disjoint set of items. We assume that, if more than one support constraint is applicable to an itemset, the one specifying the lowest minimum support is adopted. This is because adding more items to an itemset should not increase the minimum support of the itemset.

Let us consider each case

Case 1. $SC_1(B_1,B_3)>=0.2$ specifies minimum support 0.2 for any itemset containing (at least) one item in each of B1 and B3.

Case 2. $SC_2(B_3)>=0.4$ specifies minimum support 0.4 for any itemset containing one item in B3 but no item in B1 (otherwise, Case 1applies).

Case 3. $SC_3(B_2)>=0.6$ specifies minimum support 0.6 for any itemset containing one item in B2 but no item in B3 (otherwise, Case 2 applies).

Case 4. $SC_0()>=0.8$ specifies minimum support 0.8 for any other itemset (i.e., the default minimum support).
According to above constraints we construct bins.

For any itemset I containing an item from $B_1$and an item from B_3, I matches both $SC_1(B_1,B_3)>=0.2$ and $SC_2(B_3)>=0.4$, minsup(I)=0.2

Because the lowest minimum support of matched SC's is used.
For example, {0,2} {0,2,3} and {2,3,4}, {2,4,7} {2,4,8} {4,7,8} and {2,4,7,8} all have only $SC_3(B_2)>=0.6$ and are frequent. {2,7} and {2,8} match only $SC_0()>=0.8$ and {2,7} is frequent but not {2,8}.

Now the bins are

B_0	1,7,8
B_1	2,6
B_2	4,5
B_3	0,3

3.5 Typical Scenarios Of Specification:

Support-based specification: The minimum support for an itemset is a function of the support of some or all items contained in the itemset. A bin B_j usually contains similarly supported items. Such bins can be found by computing the support of items in one pass of the transactions and then clustering the items based on their supports. The bin θ_i is usually a function of some representative supports of bins (such as the maximum, minimum, or average support in the bin), and the function of θ_i can be either chosen from a menu of built-in functions or supplied by the user. If the user does not have particular schemas in mind for specification, a generic specification in the form of a nonground SC can be used.

Concept-based specification*:* When an item is present, it is desirable to specify SCs based on the generality of the item concepts. For example, $SC_1(c_1,c_2) \geq 2$ X *sup(*c_1*)* / m X *sup(*c_2*)*/ n states that any itemset containing at least one child of c_1 and one child of c_2 has the minimum support 2 X *sup(*c_1*)/ m* X *sup(*c_2*)/ n* , where c_1 and c_2 are variables representing concepts, and *m* and *n* are the number of child concepts of c_1 and c_2.

Attribute-based specification: For a database in the form of a relational table, each bin is corresponded to the set of (attribute,value) pairs from the same attribute. For example, if *States* and *Gender* are attributes in the table, SC_1(*States, Gender*) ≥ *N* /50 X *N*/2 specifies that any itemset containing a state code and a gender has the minimum support *N* /50 X*N*/2 , where *N* is the number of tuples in the relational table, *N*/50 and *N* /2 are the average support of state codes and the average support of gender.

Enumeration-based specification: The most flexible specification is explicitly enumerating the items in a bin, on the basis that they are not distinguishable with respect to the specification. For example, $SC_1(B_1,B_2) \geq 0.1$, where B_1 = {*milk, cheese*} and B_2 = {*boots, sock*}, says that any itemset containing at least one item in B_1 and one item in B_2 has minimum support 0.1. In this case, the user is interested in only *milk* and *cheese*, rather than all footwear products.

3.6 Adaptive Apriori:

Adaptive Apriori is used to push SCs following the "dependency chain" of itemsets in the itemset generation in Apriori. This dependency is best described by a schema

© 2012, IJARCSSE All Rights Reserved

enumeration tree. In a *schema enumeration tree*, each node (except the root) is labeled by a bin B_i. A node *v* represents the schema given by the labels $B_1 \ldots B_k$ along the path from the root to *v*. The ordering of nodes in a schema enumeration tree is determined dynamically on a per-node basis to achieve a certain optimality of constraint pushing.

3.7 Pushed Minimum Support:

Consider schema $s = B_1 \ldots B_{k-2}B_{k-1}B_k$, and its generating schemas $s_1 = B_1 \ldots B_{k-2}B_{k-1}$ and $s_2 = B_1 \ldots B_{k-2}B_k$. In the case of a uniform minimum support, if an itemset

$$I = \{i_1, \ldots, i_{k-2}, i_{k-1}, i_k\}$$

of 's' is frequent, so are $I_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$ of s_1 and $I_2 = \{i_1, \ldots, i_{k-2}, i_k\}$ of s_2. This property enables Apriori to generate candidate k-itemsets I using *frequent* (k-1)-itemsets I_1 and I_2. However, this generation is not available for non-uniform minimum support. Our approach is to replace *minsup* with a new function, *Pminsup*, called the "pushed minimum support", such that *Pminsup* defines a superset of the frequent itemsets and this superset can be computed in the manner of Apriori.

Let *Pminsup* be a function from (the schemas of) schema enumeration tree 'T' to [0..1] satisfying:

- Completeness. For every schema 's' in 'T' such that *minsup(s)* is defined, *Pminsup(s)* ≤ *minsup(s).*
- Apriori-like. For every schema 's' and its generating schemas s_1 and s_2, whenever an itemset $\{i_1, \ldots, i_{k-2}, i_{k-1}, i_k\}$ of 's' is *frequent(Pminsup)*, so are $\{i_1, \ldots, i_{k-2}, i_{k-1}\}$ of s_1 and $\{i_1, \ldots, i_{k-2}, i_k\}$ of s_2.
- Maximality. Pminsup is maximal with respect to Completeness and Apriori-like.
-

IV. Conclusion

In this project, we proposed a strategy for introducing support constraints into frequent Itemset mining and a frame work for pushing support constraints into the Apriori itemset generation. Instead of using the lowest minimum support specified, as in Apriori we use the best "runtime" minimum support pushed for each itemset that preserves the Apriori itemset generation. We call this strategy Adaptive Apriori.
All the improvements of Apriori are applicable to Adaptive Apriori. Moreover, this strategy does not rely on a uniform support requirement.

V . Future Work

This project can be further extended by studying how the mining framework for non-uniform minimum support can be extended beyond the Apriori itemset generation. There is also a scope of improvement by introducing trees for generating the supports for bins which improves the order sensitivity in specifying the constraints

REFERENCES

1. Agrawal, R. et al, 1993. Mining association rules between sets of items in large databases. *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD '93)*. San Diego, CA, pp. 207-216.
2. Agrawal, R. et al, 1994. Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Databases (VLDB '94)*. Santiago de Chile, Chile, pp. 478-499.
3. Berners-Lee, T. et al, 2001. The Semantic Web, *Scientific American*, Vol. 284, No. 5, pp. 34-43.
4. Bonchi, F. et al, 2005. Pushing Tougher Constraints in Frequent Pattern Mining. *Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*. Hanoi, Vietnam, pp. 114-124.
5. Broekstra, J. et al, 2002. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. *Proceedings of the first International Semantic Web Conference (ISWC 2002)*. Sardinia, Italy, pp. 54-68.
6. Broekstra, J. and Kampman, A., 2004. SeRQL: An RDF Query and Transformation Language. http://www.cs.vu.nl/ jbroeks/papers/SeRQL.pdf.
7. Grahne, G. et al, 2000. Efficient mining of constrained correlated sets. *Proceedings of 16th International Conference on Data Engineering (ICDE' 00)*. San Diego, CA, pp. 512-524.
8. Han, J. and Fu, Y., 1995. Discovery of multiple-level association rules from large databases. *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB '95)*. San Francisco, CA, pp. 420-431.
9. Han, J. et al, 2002. Mining Top-k frequent closed patterns without minimum support. *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM'02)*. Maebashi City, Japan, pp. 211-218.
10. Hou, X. et al, 2005. Application of Data Mining in Fault Diagnosis Based on Ontology. *Proceedings of the 3rd Conference on Information Technology and Applications (ICITA '05)*. Sydney, Australia, pp. 260-263.

© 2012, IJARCSSE All Rights Reserved