Asian Journal of Information Management

ISSN 1819-334X

Academic Journals Inc.

www.academicjournals.com

Asian Journal of Information Management, 3 (1): 7-17, 2009
ISSN 1819-334X

Improving the Performance of Association Rule Mining Algorithms by Filtering Insignificant Transactions Dynamically

[1]Rajendra K. Gupta and [2]Dev Prakash Agrawal
[1]Department of CSE and IT, MITS, Gwalior (MP), India
[2]Union Public Service Commission, New Delhi, India

Abstract: Present study proposes an algorithm for finding frequent itemsets. Algorithm uses a novel approach to the insignificant transactions dynamically. It divides the tuples of the database to be mined intelligently in clusters. During a particular pass only those clusters that seem to be statistically useful are to be scanned and as a consequence all insignificant tuples will be filtered out dynamically. Further, the algorithm is based on a vertical data layout and offers flexibility during mining process. Experiments have been performed on real databases and the results have been presented. The results show that by removing false frequent items and insignificant transactions dynamically, the performance of association rule-mining algorithms can be improved. It has also been observed that the performance gap increases with the large size of database and/or when there exist prolific size frequent itemset in the database at the given value of minimum support.

Key words: Knowledge discovery, association rules algorithm, interesting patterns, cluster based association rule mining

INTRODUCTION

Data mining relate to discovering of previously unknown patterns in large databases and is an important step in the knowledge discovery process. It is emerging as a major application area for databases (Agrawal *et al*., 1993; Bjorvand, 1998; Landau *et al*., 1998; Hunzikar *et al*., 1998; Aggarwal and Yu, 1999; Tiwari *et al*., 2008) had introduced different classes of data mining problem involving clustering, classification, prediction and association.

Association rules are used to show the relationships between the data items. These uncovered relationships are neither inherent nor casual, but rather based on co-occurrence of the data items. The discovery of association relationships among huge amount of data is useful in various problems *viz*., decision analysis, Customer Relationship Management (CRM), cross marketing and fraud detection etc. A popular area of application is introduced by Agrawal *et al*. (1993) is market basket analysis, which studies the buying pattern of customers by searching for sets of items that are frequently purchased together or in a sequence. It is important to improve the quality of business decisions by analyzing past transaction data to discover customer purchasing behaviors. In order to support this analysis, a sufficient amount of transactions needs to be collected and stored in a database. A transaction in the database typically consists of customer identifier, transaction date and items purchased in the transaction. Because the amount of these transaction data can be very large, an efficient algorithm needs to be designed for discovering useful information.

Most of the algorithms to mine the association rules are divided into two phases, the first is to find the frequent itemset; the second is to use the frequent itemset to generate association rules. The identification of the frequent itemset is a computationally expensive task. Present study proposes an algorithm for finding frequent itemsets. Algorithm uses a novel approach to the insignificant

Corresponding Author: Rajendra K. Gupta, Department of CSE and IT, MITS, Gwalior (MP), India

transactions dynamically. It divides the tuples of the database to be mined intelligently in clusters. During a particular pass only those clusters that seem to be statistically useful are to be scanned and as a consequence all insignificant tuples will be filtered out dynamically. Further, the algorithm is based on a vertical data layout and offers flexibility during mining process. Experimental results show that the algorithm reduces the size of database scanned during each pass causing reduction in execution time to a large extent, especially when there exists prolific patterns in the database and/or the size of database is very large.

Appendix shows the different notations used in the study.

ASSOCIATION RULE PROBLEM

The problem of mining association rules is to generate all rules that have support and confidence greater than or equal to some user specified minimum support and minimum confidence threshold, respectively. A formal statement of the association rule problem is given by Agrawal *et al*. (1993).

Let $I = \{ i_1, i_2, i_3, i_4 \ldots\ldots\ldots i_m \}$ be a set of m distinct literals called items, D is a set of transactions (variable length) over I. Each transaction contains a set of items $i_1, i_2, i_3, i_4 \ldots\ldots\ldots i_k \subseteq I$. Each transaction is associated with an identifier, called TID. An association rule is an implication of the form $X \Rightarrow Y$, where, $X, Y \subset I$ and $X \cap Y = 0$. Here, X is called the antecedent and Y is called the consequent of the rule. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence α if among those transactions that contain X α% of them also contain Y. The rule $X \Rightarrow Y$ has support S in the transaction set D if S% of transactions in D contains $X \cup Y$. The selection of association rules is based on these two values (some additional constraints may also apply). These are two important measures of rule interestingness. They respectively reflect usefulness and certainty of a discovered rule. They can be described by the following equations:

$$\text{Support } (X \Rightarrow Y) = \text{Frequency } (X \cup Y) \mid D \mid$$

$$\text{Confidence } (X \Rightarrow Y) = \text{Frequency } (X \cup Y) / \text{Frequency } (X)$$

where, | D | represents the total number of transactions (tuples) in D.

A frequent itemset is an itemset whose number of occurrences is above a minimum support threshold. An itemset of length k is called k-itemset and a frequent itemset of length k as k-frequent itemset. An association rule is considered strong if it satisfies a minimum support threshold and minimum confidence threshold.

Factors Affecting the Efficiency

As explained earlier that the identification of the frequent itemsets is computationally expensive. Once all sets of frequent itemsets are obtained, there is a straightforward algorithm, given by Agrawal and Srikant (1994), for finding association rules. The naive algorithm for finding frequent itemsets is not practical in real world applications, since, it requires exhaustive search, which may behave well in a small problem domain, but is not practical when applied to large databases. The efficiency of an association rule algorithm usually depends upon the number of database scans, size of database during scan, number of candidate itemset generated and tested (counted) and efforts required to remove the duplicate rules etc. Hence, the primary goals of any association rule algorithm are to reduce the number of candidate itemsets generated and tested as well as the number of scans of database required and scan the database as small as possible. These activities require considerable amount of processing time and memory. Therefore, it is crucial that exhaustive search is avoided in real world applications and some heuristics should be introduced to eliminate statistically insignificant items and/or transactions as early as possible in the frequent pattern discovery process.

A Critical Look on Currently Used Algorithms

In recent years several fast algorithms for generating frequent itemsets have been suggested in literature by Mannila *et al*. (1994), Savasere *et al*. (1995), Agrawal and Srikant (1994), Yen and Arbee Chen (2001), Park *et al*. (1997), Houtsma and Swami (1995), Coenen *et al*. (2001) and Tiwari *et al*. (2008). An analysis of these has led the authors to identify the following limitations/shortcomings in them:

- Recently developed graph based approaches (Yen and Arbee Caen, 2001; Han and Yin, 2000) for mining association patterns (frequent itemsets) require the construction of the association graph (Yen and Arbee Chen, 2001) and the frequent pattern tree (Han and Yin, 2000), respectively for every new value of minimum support, by scanning the entire database. Even for the case when database is incremented or mining is performed at some higher concept level existing association graph or frequent pattern tree has to be reconstructed from scratch level. Construction of the association graph or frequent pattern tree is a time consuming activity. Further, these approaches do not offer flexibility and reusability of computation during mining process
- Most of the existing algorithms designed for generating frequent itemset mining algorithms tend to generate and test unnecessary insignificant 2-candidate itemsets due to the presence of false frequent items. False frequent items are those items which look like frequent but are actually not frequent. Park *et al*. (1997) observed that execution time of the first two passes required by the apriori algorithm (Agrawal and Srikant, 1994) is about 62% of the total execution time. This calls for filtering false frequent items at the early stage, which reduces the generation and testing of 2-candidate itemsets, C
- AIS (Agrawal *et al*., 1993), SETM (Houtsma and Swami, 1995), Apriori (Agrawal and Srikant, 1994) and its variants and the graph based association pattern mining algorithms (Yen and Arbee Chen, 2001; Han and Yin, 2000; Agarwal *et al*., 2000) do not have any advance information/ prediction regarding the maximum size of frequent itemsets present in the database prior to actual mining, at a given minimum support. Hence, they continue to generate and test the candidate itemsets till candidate itemsets in a pass become null. Due to this many insignificant candidate itemsets are generated and tested. Testing of such insignificant candidate itemsets may also demand one more database scan for algorithms (Agrawal *et al*., 1993; Agrawal and Srikant, 1994; Yen and Yin, 2001; Houtsma and Swami, 1995) and one more frequent pattern tree scan for algorithm (Han and Yin, 2000)
- Algorithms given (Agrawal *et al*., 1993; Mannila *et al*., 1994; Savasere *et al*., 1995; Agrawal and Srikant, 1994; Park *et al*., 1997; Han and Yin, 2000; Toivonen, 1996) are not suitable for verification driven association rule mining as they are based on horizontal data layout
- Most of the time users run association rule mining algorithm at different value of support, confidence and abstract level to discover hidden, previously unknown and ultimately useful knowledge from the huge volume of data. This may take a long time before giving the desired results. Repeated execution of mining algorithm with varied constraints incurs prohibitive costs. This calls for flexibility and reusability in the algorithm which does not exists in most of the algorithm

PROPOSED ALGORITHM

To address the above-mentioned limitations/shortcomings, a new algorithm called Cluster Based Association pattern mining algorithm (CBA) has been proposed in this study for mining the frequent

itemsets. CBA uses a complete, level-wise bottom-up search, with a vertical data layout (encoded) and enumerates all frequent itemsets. It is an iterative algorithm that counts k-itemsets in pass k. CBA uses a novel approach to reduce the disk I/O operations. First it predicts the size of the maximum length frequent itemset, designated by γ in the present study, which may present in the data in the worst case at the given minimum support. Then divides the tuples of the database (encoded) to be mined in γ-1 clusters. During a particular pass only those clusters (or tuples) that seem to be statistically useful are to be scanned, without losing any frequent itemset as illustrated by the following example:

- Suppose for a given database at the assumed minimum support of 15%, it is predicted that six is the maximum size of the frequent itemset which may be present in the data in the worst case (i.e., $\gamma = 6$). Then tuples of the database are divided in 5 clusters (i.e., γ -1). First four clusters: $\omega_2, \omega_3, \omega_4$ and ω_5 contain all those tuples containing 2, 3, 4 and 5 frequent items, respectively and the last cluster, ω_6 contain those tuples in which number of frequent items are greater than or equal to 6. When frequent itemsets of size 5 are searched, these itemsets may only be present in clusters ω_5 and ω_6. Thus, for this particular case transactions available in clusters ω_2, ω_3 and ω_4 are statistically insignificant and should not be scanned. Therefore, during each higher pass, number of transactions scanned are reduced compared to the previous pass. This reduces the amount of disk I/O required and makes CBA more efficient especially in the situation where size of database is large and number of 1-frequent items is also more. In the light of above following lemma can be derived

Lemma 1

A frequent itemset of size k can not be present in any cluster ω_x, where x<(k-1) and ω_x is a cluster containing tuples having x 1-frequent items.

Formal description of the proposed algorithm is given below:

Algorithm

Cluster Based Association pattern mining algorithm (CBA).

```
Input:
Data file, D
Minimum support, min_supp
Output:
Frequent itemsets, L
Process:
    begin
        L1 = ϕ;     //L1 is the set of 1-frequent items
        L  = ϕ;
        L1 = identify_frequent_item(count_table, min_supp);
             //Identifies 1-frequent items
        if |L1| < 2 then
        print "No association pattern exist at given support"
        exit(0);
      else
        ξ = make_concentrator(D, L1);
           //Constructs the Concentrator – database in encoded form and having
            // only significant items
```

```
                    //Loop 1
                    //Detection and filtering of false frequent items
                    for all 1 ∈ L1 do
                        if 1.count < min_supp then
                            // Detection of false 1-frequent item
                            L1 = L1\1 ;
                            // Filtering of false 1-frequent item from L1
                        endif;
                    endfor;
                    if |L1| < 2 then
                        print "No association pattern exists at given support"
                        exit(0);
                    else //Prediction of maximum length of largest frequent itemset
                        alterable ξ add (hcount) attribute;
                        k = no_of_ attribute(ξ);
                        //Loop 2
                        for (I = 1; I ≤ |ξ|; I++) do begin
                                for (j = 2; j ≤ |ξ|; j++) do begin
                                        t[i.hcount] = t[i.hcount] + t[i,j];
                                endfor;
                        endfor;
                        position = ⌈(|ξ|*min_supp)/100⌉;
                        ξ' = select all from ξ order by hcount desc;
                        γ = ξ'[position.hcount];
                        ξ = ξ';
                        // Association Pattern Generation
                        //Loop 3 Making of clusters
                        for (I = 1; I ≤ γ; I++) do begin
                                ξ(I) = select all from ξ where ξ.hcount ≥ I;
                        endfor;
                        //Loop 4 Candidate itemset generation
                        for (k = 2; L[k-1] ≠ ϕ AND k ≤ γ; k++) do begin
                                C[K] = gen_pattern_cluster(L[k-1], ξ(k));
                                //Generates new candidate itemsets of size k,
                                //where 2 ≤ k ≤ γ
                                forall c ∈ C[K] do
                                //Tests whether a candidate itemset is frequent
                                        if c.count ≥ min_supp then
                                            L[k] = L[k] ∪ c;
                                        endif
                                endfor;
                                L = L ∪ L[k];
                            endfor;
                    endif;
            endif;
end;
```

Brief description of the algorithm is given below:

First of all, the items present in the pre-mined data are encoded and individually counted. Information regarding encoding and counting is recorded in encode_decode_table and count_table, respectively for use during frequent itemset generation and rule generation phase, respectively. Further mining processes are performed only on this encoded data. Now, L_1 (set of frequent items of size 1) and L (set of frequent itemsets) are initialized to null. Function identify_frequent_item() is called which identifies 1-frequent items and assigns to L_1. The input parameters to this function are count_table and user defined minimum support, min_supp. If $|L_1| < 2$ then no association pattern exists in the database at the given minimum support and algorithm terminates. Otherwise, function make_concentrator() is invoked which constructs the concentrator for given database at the user defined support. The concentrator is a predefined structure containing mostly statistically significant items and transactions in the encoded form. Hence, its overall size is reduced as compared to the original data to be mined. Its construction requires one complete scan of the entire data to be mined. After this, the algorithm will not scan the original data during mining performed at the support greater than or equal to at which concentrator is constructed. The input parameters of this function are data, D and 1-frequent items, L_1. Formal description of function make_concentrator() is given below:

```
function make_concentrator(D, L1)
create table ξ(tid, L1) as attributes;
    I = 1;
    forall transaction t ∈ D do
        items = t ∩ L1;
            if | items | ≥ 2, then
                        forall j ∈ items do
                        ξ(I, j) = 1;
                        endfor;
                        I = I +1;
                else
                        forall j ∈ items do
                                j.count = j.count – 1;
                        endfor;
                endif;
        endfor;
return ξ;
```

Brief description of this function is as below:

When function make_concentrator() is invoked, it creates a table (ξ) with tid and 1-frequent items (each member of L_1 will be treated as independent attribute) as attributes. It then reads all the transactions of data file one by one. For each transaction containing at least two frequent items, 1 is entered in location (I, j) of the concentrator for each frequent item present in the transaction; where I and j are row and column in the concentrator corresponding to a Tid and frequent item of the database respectively. If any transaction contains less than two frequent items then such a transaction is not entered in the concentrator as a row (such transactions are statistically insignificant). Thus, the concentrator contains ($|L_1|+1$) columns and each column (except the first) is a bit vector corresponding to a specific element of L_1 and the number of rows in the concentrator will be less than or equal to $|D|$. The bit vector associated with item i is denoted as β_i and number of 1s in a bit vector β_i by $\beta_{i(1)}$. The resulting concentrator can be preserved in a secondary storage for future use in mining process. Computations done during construction of the concentrator are shared and reused every time the user requests for mining of association rule at the same or higher support at which the concentrator is constructed.

The extra space, that is required to store the concentrator is apparently an overhead. However, the benefits in terms of faster response time, flexibility and reusability outweigh the expense.

Due to pruning of insignificant transactions during construction of the Concentrator, it may be possible that the support of some of the items in the concentrator may fall below the minimum required support. Such items have been designated as false frequent items in this study. Presence of such items in the concentrator will require unnecessary generation and testing of candidate itemsets. Thus, such items needed to be filtered out before starting the actual mining process. This also results in the further reduction of the size of the concentrator. The false frequent items present in the concentrator can be detected by using the following lemma.

Lemma 2

If for any item I of the concentrator the value of $\beta_{i(1)}$ that is number of 1's in column corresponds to item I is less than the minimum support i.e., $|\beta_{i(1)}| <$ min_supp, then item I is the false frequent item.

The real support of an item I is obtained by counting number of 1s in β_i i.e., the value of $\beta_{i(1)}$. If the value of $\beta_{i(1)}$ is less than the minimum support then item I is the false frequent item. Columns corresponding to such items are filtered out from the concentrator and also removed from L_1. Remaining items in L_1 are actual frequent items. The support of different items, which was calculated before making the concentrator, is apparent support. The value of apparent support is always greater than or equal to its real support.

After this function predict_size() is called that could predict the maximum size (γ) of the frequent itemset present in the database at a given minimum support. Now tuples of the concentrator are divided in γ-1 clusters (loop 3) in such a way that each cluster I, where $2 \leq I < \gamma$, contains only those tuples in which number of 1's are I (containing I 1-frequent items) except last cluster which will contain all those tuples for which number of 1's are greater than or equal to γ.

Now function gen_pattern_cluster() is called (loop 4) to generate candidate itemset of size k i.e., C_k by using frequent itemsets of previous pass i.e., (k-1). Each $c \in C_k$ is to be tested whether, it is frequent or not by searching it in all those clusters, ω_i for which $I \geq k-1$. Function gen_pattern_cluster() scans only those transactions which are statistically significant (Lemma 1). Thus, during each higher pass number of scanned transactions reduces drastically. For each itemset, occurrences are counted and the itemset for which item.count$\geq$ min_supp is a frequent itemset and this itemset is appended in L_k, which is k-frequent itemset. At the end of pass k, L_k is appended to L. This process is repeated till $L_k \neq \phi$ AND $k \leq \gamma$. At the end, L gives the frequent itemsets. The formal description of gen_pattern_cluster() is given below:

```
function gen_pattern_cluster(L_{k-1}, ξ(k))
//This function generates the k-candidate itemsets from the given (k-1)-frequent itemsets
//and also calculates the actual support of generated candidates.

    C_k = φ;
    //Loop 5
    for (I = 1; I ≤ |L_{k-1}|-1; I++ ) do begin
        for (j = I+1; j ≤ | L_{k-1}|; j++) do begin

            if (left(t_i, | t_i |-1) = = (left(t_j, | t_j |-1)) then
                // t_i is the i^th row of L_{k-1}
```

```
                    // t_j is the j^th row of L_k-1
                    if left(t_i, | t_i |-1) = ϕ then
                            c.item = t_i ∪ t_j;
                    else
                            c.item = left(t_i, | t_i |-1) + right(t_j, 1);
                    endif;
                    c.count = supp(c.item, ξ(k));
                    C_k = C_k ∪ c;
                else
                    exitfor;
                endif;
            endfor;
        endfor;
return C_k;
```

Performance Evaluation of the Proposed Algorithm

To see the effects of filtering insignificant tuples dynamically during mining process experiments have been performed on a Pentium-4 based PC running on Window 98. Proposed Algorithm was executed and results were compared with similar algorithm without clustering designated by CBA*. All experiments have been performed on real-life datasets obtained from retail departmental stores. A transaction contained data about the items purchased by the customer in a visit to the store.

The results indicate that execution time of CBA also increases like CBA* with the decrease in minimum support (Table 1). However, in this case rate of increase of execution time on decreasing support is less compared to CBA*. It was as per expectation because at reduced minimum support there were more candidate itemsets for testing. Table 1 also indicates that difference in execution time taken at support 0.51 and 1.01% were approximately same. The reason is number of candidate itemsets generated and tested at these support were approximately same.

Table 2 shows that for a given number of frequent items (eight) and a specified minimum support of 0.01% the execution time increases continuously as the size of the dataset (number of transactions) increases. Similar trends at lower number of frequent items (five) and same minimum support of 0.01% is also noticed through Table 3. However rate of increase is rapid for the case given in Table 2. It may be attributed increase in the number of frequent items.

It is noticed that the execution time increases as the number of frequent items increase. However, the rate of increase for CBA is much less compared to CBA*. Further, it is to be noted that (Table 4)

Table 1: Execution times for different datasets at different minimum support

		Minimum support (%)		
Algorithm	Dataset	0.01	0.51	1.01
CBA*	T12.I8.D70K	859	389	378
	T12.I11.D70K	3091	820	836
CBA	T12.I8.D70K	265	248	242
	T12.I11.D70K	519	296	283

CBA* is the algorithm without clustering

Table 2: Execution time for data having 8 frequent items at min supp of 0.01%

	No. of transactions in K											
Algorithm	15	20	25	30	35	40	45	50	55	60	65	70
CBA*	154	168	189	211	232	251	269	289	310	324	347	859
CBA	122	138	147	162	169	179	193	202	215	229	242	267

CBA* is the algorithm without clustering

Table 3: Execution times for data having 5 frequent items at min_supp of 0.01%

	No. of transactions in K											
Algorithm	15	20	25	30	35	40	45	50	55	60	65	70
CBA*	68	74	77	83	90	96	98	106	109	115	119	126
CBA	70	79	88	99	102	111	123	133	143	156	174	193

CBA* is the algorithm without clustering

Table 4: Execution times at min_ supp of 0.01%

		No. of frequent Items							
No. of transactions	Algorithm	3	4	5	6	7	8	11	17
70K	CBA*	49	76	125	313	620	859	3091	4085
70K	CBA	93	124	161	176	217	267	519	713

CBA* is the algorithm without clustering

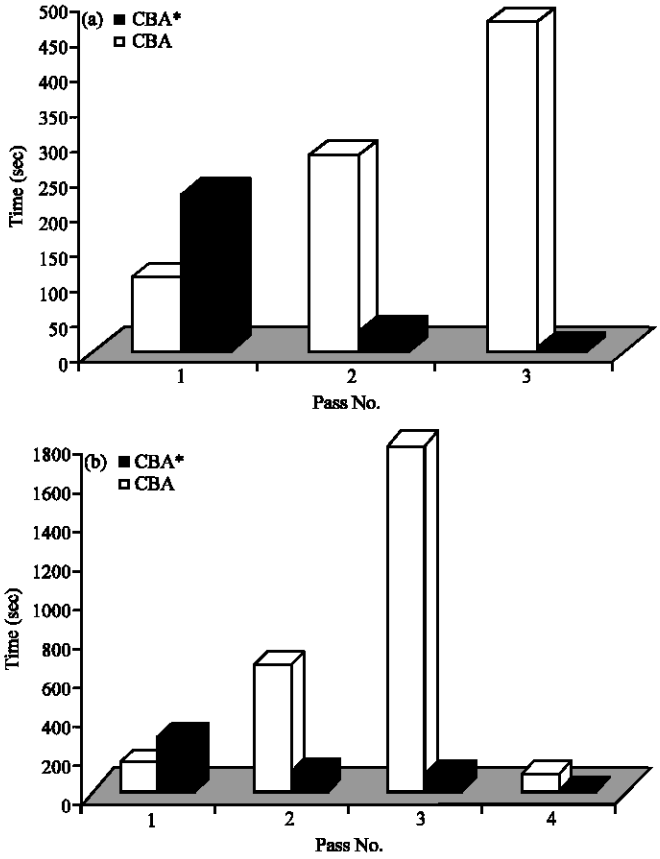

Fig. 1: Time taken by CBA* and CBA in different passes (min_supp = 0.01), (a) T12.18.D70 k and (b) T12.111.070 k

when frequent items are more than 5 CBA becomes more efficient compared to CBA*. Performance gap increases with number of frequent items in the data at the given minimum support. As both CBA* and CBA have used same candidate itemsets generation and testing (counting) techniques, the improvement in the execution times for CBA is mainly due to the reduction in scanned database size during higher passes. It is noticed that CBA* scanned same size of database during each successive

pass while CBA reduced the scanned database size progressively during each higher pass. However, during pass 1 both CBA* and CBA scanned the same size database and time required for CBA* is less than CBA as shown in Fig. 1. It was expected because during this pass CBA also performed some computational efforts for making the clusters. Performance gaps between CBA* and CBA increases with higher passes.

CONCLUSION

An algorithm for discovering all frequent itemsets in a large transactional database has been proposed. Its filters out the insignificant tuples dynamically and scanned reduced data at each higher pass. Experiments have been performed on real databases and the results have been presented. The results show that by removing false frequent items and insignificant transactions dynamically, the performance of association rule-mining algorithms can be improved. It has also been observed that the performance gap increases with the large size of database and/or when there exist prolific size frequent itemset in the database at the given value of minimum support.

APPENDIX

Nomenclature

D	:	Database
|D|	:	Number of transactions in the database
I	:	Set of items
t_i	:	Transaction I in database
|t|	:	Size of transaction
X, Y	:	Itemset
S	:	Actual support
C	:	Actual confidence
k-itemset	:	An itemset having k items
min_supp	:	Minimum support
min_conf	:	Minimum confidence
ξ	:	Concentrator
β_i	:	Bit vector corresponding to item I
$\beta_{i(1)}$	:	Count of 1s in bit vector of item I
C_k	:	Set of candidate k-itemsets
L_K	:	Set of frequent k-itemsets
L	:	Set of frequent itemsets of all size
L_{max}	:	Set of maxpatterns
L_{maxk}	:	Set of k-maxpatterns
l	:	A single frequent itemset
γ	:	Maximum size of frequent itemset(s) that may present in data (worst case) at given minimum support
X⇒Y	:	Association rule having itemset X in antecedent and Y in consequent

Dataset Description

TX-X	:	Average number of items in transactions
IY-Y	:	Number of frequent items present in the data
DZK-Z	:	Number of transactions in data and K-represents thousand

REFERENCES

Agrawal, R., T. Imielinski and A. Swami, 1993. Mining association rules between sets of items in large databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, May 25-28, New York, USA., pp: 207-216.

Agrawal, R. and R. Srikant, 1994. Fast algorithm for mining association rules in large databases. Proceedings of the 20th International Conference on Very Large Data Bases, Sept. 12-15, San Francisco, CA, USA., pp: 487-499.

Aggarwal, C.C. and P.S. Yu, 1999. Data mining techniques for associations, clustering and classification. Proceedings of the 3rd Pacific-Asia Conference, PAKDD-99, Beijing, China, pp: 13-23.

Agarwal, R.C., C. Aggarwal and V.V.V. Prasad, 2000. A tree projection algorithm for generation of frequent item sets. J. Parallel Distributed Comput., 61: 350-371.

Bjorvand, A.T., 1998. Object mining: A practical application of data mining for the construction and maintenance of software components. Proceedings of the 2nd European Symposium, PKDD-98, Nantes, France, pp: 121-129.

Coenen, F., G. Graham and L. Paul, 2001. Computing association rules using partial totals. Proceedings of the 5th European Conference, PKDD 2001, Freiburg, Germany, pp: 141-149.

Han, J., J. Pei and Y. Yin, 2000. Mining frequent patterns without candidate generation. Proceedings of ACM SIGMOD International Conference on Management of Data,, 2000, Dallas, TX., pp: 1-12.

Houtsma, M. and A. Swami, 1995. Set oriented mining for association rules in relational databases. Proceedings of the 11th IEEE International Conference on Data Engineering, 1995, Twenty Univ., Enschede, pp: 25-33.

Hunzikar, P., M. Andreas, N. Alex, T. Markus, W. Douglas and Z. Peter, 1998. Data mining at a major bank: Lessons from a large marketing application. Proceedings of the 2nd European Symposium, PKDD-98, Nantes, France, pp: 345-351.

Landau, D., R. Feldman, O. Zamir, Y. Aumann, M. Fresko, Y. Lindell and O. Lipshtat, 1998. Text vis: An integrated visual environment for text mining. Proceedings of the 2nd European Symposium, PKDD-98, Nantes, France, pp: 56-64.

Mannila, H., H. Toivonen and A. Inkeri Verkamo, 1994. Efficient algorithms for discovering association rules. Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, (KDD-94), IEEE, pp: 181-192.

Park, J.S., S. Ming and S. Philips Yu, 1997. Using a hash-based method with transaction trimming for mining association rules. IEEE Trans. Knowledge Data Eng., 9: 813-825.

Savasere, A., E. Omieccinski and S. Navathe, 1995. An efficient algorithm for mining association rules in large databases. Proceedings of the 21st International Conference on Very Large Databases, 1995, Zurich, Switzerland, pp: 432-443.

Tiwari, A., R.K. Gupta and D.P. Agrawal, 2008. Mining frequent item sets using prime number based approach. Proceedings of the 3rd International Conference on Advanced Computing and Communication Technologies (ICACCT), Nov. 08-09, India, pp: 138-141.

Toivonen, H., 1996. Sampling large databases for association rules. Proceedings of International Conference on Very Large Databases, 1996, Bombay, India, pp: 134-145.

Yen, S.J. and L.P. Arbee Chen, 2001. A graph-based approach for discovering various types of association rules. IEEE Trans. Knowledge Data Eng., 13: 839-845.