© 2016 IJSRSET | Volume 2 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Using Hyper-structure mining to Ascertain Recurrent Patterns in Large Dataset

R. D. Priyanka, Dr. R. Sabitha, T. Mythili

Department of Information Technology, Info Institute of Engineering, Kovilpalayam, Coimbatore, Tamilnadu, India

ABSTRACT

Data mining (DM) is the process of exploration and analysis, by automatic or semiautomatic means, of large quantities of data in order to discover meaningful patterns and rules.One of the important problems in data mining is discovering association rules from databases of transactions where each transaction consists of a set of items. Frequent itemsets play an important role in many data mining tasks. The task of finding the frequent itemsets is a fundamental problem in data mining. In this paper we examine the problem of finding frequent itemsets using Hyper-structure mining of frequent patterns in large databases (H-Mine) algorithm on Mushroom Dataset. The performance of the algorithm is evaluated with respect to execution time and frequent pattern generation. The Mushroom dataset contains characteristics of various species of mushrooms, and was originally obtained from the UCI Repository of Machine Learning Databases. H-Mine is one of the efficient algorithm for mining frequent itemsets in a set of transactions. H-mine uses the advantage of the H-struct data structure and dynamically adjusts links in the mining process. A distinct feature of this method is that it has very limited and precisely predictable space overhead and runs really fast in memory-based setting. . In this paper we describe the implementation of this algorithm with respect to both execution time and frequent pattern generation.

Keywords : Data Mining, Association Rule Mining, Frequent Itemsets, H-mine

I. INTRODUCTION

Data mining refers to extracting or mining knowledge from large amounts of data [3]. Discovery of interesting association relationships among huge amounts of data will help marketing, decision making and business management [4]. The goal of the technique is to detect relationships or associations between specific values of categorical variables in large data sets. These powerful exploratory techniques have a wide range of applications in many areas of business practice and also research - from the analysis of consumer preferences or human resource management, to the history of language. These techniques enable analysts and researchers to uncover hidden patterns in large data sets.

Association rule mining is a twostep process. In the first step, all frequent itemises that occur at least as frequently as a predetermined minimum support count are found. In the second step, strong association rules that must satisfy minimum support and minimum confidence, from the frequent itemsets are generated. The overall performance of mining association rule is determined by the first step [3]. Therefore, efficient discovery of frequent patterns from large databases is the fundamental problem in data mining. In this paper we examine the problem of finding frequent itemset on mushroom dataset using H-mine algorithm [1]. The mushroom dataset contains characteristics of various species of mushrooms, and was originally obtained from the UCI Repository of Machine Learning Databases. [2]

II. METHODS AND MATERIAL

.

2. Problem statement

2.1 Basic concepts

The association mining task can be stated as follows:

Let *I* be a set of items and D be a database of transactions, where each transaction has a tid and contains a set of items. A set of items is also called an itemset. An itemset with k items is called a k-itemset. The support of an itemset X, denoted as σ(X) is the number of transactions in which it occurs as a subset. The input data (D) for most ARM algorithms comprises N columns describing a binary valued set of attributes A, and M transactions such that each transaction describes some subset of A. The k length subset of an itemset is called a k-subset. An itemset is maximal if it is not a subset of any other itemset. An itemset is frequent if its support is more than a user-specified minimum support (min_sup) value. The set of frequent k-itemsets is denoted as F_k. If the support of an item set is greater than a given support threshold, "min_sup", the itemset is said to be large or frequent [4].

The data mining task is to generate all association rules in the database, which have a support value greater than min_sup. These association rules are frequent. This task can be broken into two steps:

- **Find all frequent itemsets.** Given m items, there can be potentially 2m frequent itemsets. Efficient methods are needed to traverse this exponential itemset search space to enumerate all the frequent itemsets.
- **Generate confident rules.** This step is relatively straightforward to generate association rules of the form X - > Y, for all frequent itemsets X and Y. [8]

3. H-MINE: Hyper-Structure Mining

H-mine takes advantage of the data structure H-Struct and dynamically adjusts links in the mining process. A distinct feature of this method is that it has very limited and precisely predictable space overhead and runs really fast in memory-based setting. Moreover, it can be scaled up to very large databases by database partitioning, and when the data set becomes dense, (conditional) FP-trees can be constructed dynamically as part of the mining process. H-mine has high performance in various kinds of data, and is highly scalable in mining large databases. [1]

3.1 General Idea of H-Mine

Let Table 1 be the transaction database *TDB*. Let the minimum support threshold be *min sup* = 2. The frequent item projection contains items which are frequent, which have a support value greater than min_sup.

Transaction	Items	Frequent item Projection
100	c,d,e,f,g,i	c,d,e,g
200	a,c,d,e,m	a,c,d,e
300	a,b,d,e,g,k	a,d,e,g
400	a,c,d,h	a,c,d

Table 1. The transaction database *TDB* used

Following the *Apriori* property [5], only frequent items play roles in frequent patterns. By scanning *TDB* once, the complete set of frequent items {*a* : 3, *c* : 3, *d* : 4, *e* : 3, *g* : 2} can be found and output, where the notation *a* : 3 means item *a*'s support (occurrence frequency) is three. Let *freq* (*X*) (the *frequent-item projection* of *X*) be the set of frequent items in itemset *X*. Following the alphabetical order of frequent items1 (called an *F-list*) *a-c-d-e-g*, the complete set of frequent patterns can be partitioned into five subsets as follows:

(i) those containing item a;
(ii) those containing item c but not item a;
(iii) those containing item d but no item a nor item c;
(iv) those containing item e but no item a nor item c nor item d; and
(v) those containing only item g, as shown in Figure 1

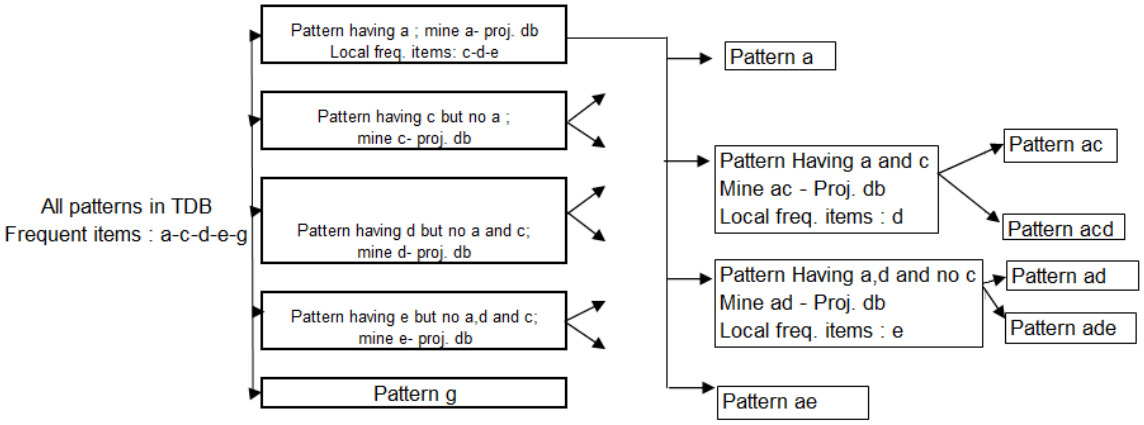

Figure 1. Divide and Conquer Frequent Patterns

If the frequent-item projections of transactions in the database can be held in the main memory, then they can be organized as shown in Figure. 2. All items in frequent-item projections are sorted according to the *F-list*. For example, the frequent-item projection of transaction 100 is listed as *cdeg*. Every occurrence of a frequent item is stored in an entry with two fields: an *item-id* and a *hyper-link*. A *header table H* is created, with each frequent item entry having three fields: an *item-id*, a *support count*, and a *hyperlink*. When the frequent-item projections are loaded into the memory, those with the same first item (in the order of the *F-list*) are linked together by the hyper-links into a queue, and the entries in header table *H* act as the heads of the queues.

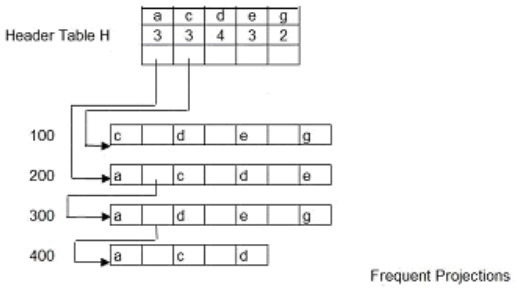

Figure 2. H-struct, the hyper-structure to store frequent-items projections.

Clearly, it takes one scan (the second scan) of the transaction database *TDB* to build such a memory structure (called the H-struct).

Then the remaining mining is performed on the H-struct only, without referencing any information in the original database. After that, the five subsets of frequent patterns can be mined one by one. To find the set of frequent patterns in the first subset, i.e., all the frequent patterns containing item *a*. This requires to search all the frequent item projections containing item *a*, i.e., the *a-projected database*, denoted as $TDB|_a$. The frequent item projections in the a-projected database are linked in the a-queue, which can be traversed efficiently.

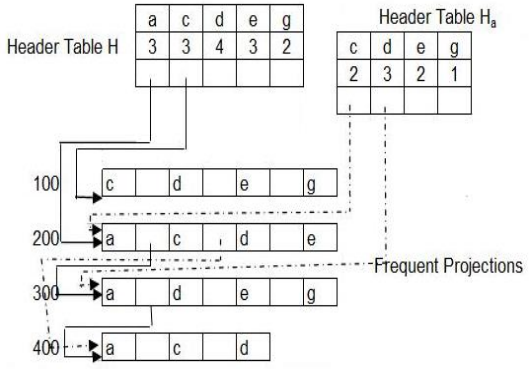

Figure 3. Header table Ha and the ac-queue

To mine the *a*-projected database, an *a-header table Ha* is created, as shown in Figure. 3. In H_a, every frequent item, except for *a* itself, has an entry with the same three fields as *H*, i.e., *item-id*, *support count* and *hyper-link*. The support count in H_a records the support of the corresponding item in the *a*-projected database. By traversing the *a*-queue once, the set of locally frequent items, i.e., the items appearing at least two times, in the *a*-projected database is found, which is {*c* : 2, *d* : 3, *e* : 2} This scan gives the frequent patterns {*ac* : 2, *ad* : 3, *ae* : 2} as output and builds up links for the H_a header as shown in Figure 3.

Similarly, the process continues for the ac-projected database by examining the c-queue in Ha, which creates an ac-header table H_{ac}, as shown in Figure 4.

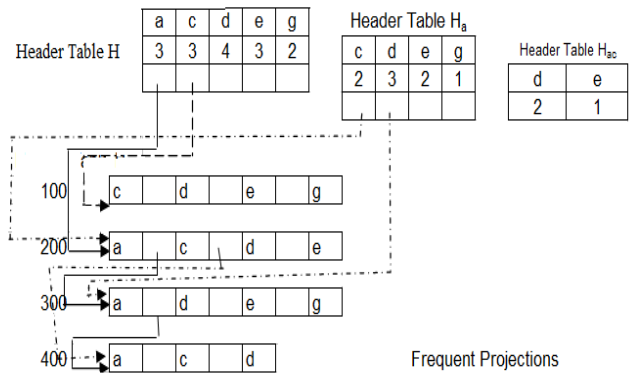

Figure 4. Header Table H_{ac}

Since only item d : 2 is a locally frequent item in the ac projected database, only acd : 2 will be the output, and the search along this path completes. Then the recursion backtracks to find patterns containing a and d but not c. Since the queue started from d in the header table Ha, i.e., the ad-queue, links all frequent item projections containing items a and d (but excluding item c in the projection), one can get the complete ad projected database by inserting frequent-item projections having item d in the ac-queue into the ad-queue. This involves one more traversal of the ac-queue. Each frequent item projection in the ac-queue is appended to the queue of the next frequent item in the projection according to *F-list*. Since all the frequent-item projections in the *ac*-queue have item *d*, they are all inserted into the *ad*-queue, as shown in Figure 5. After the adjustment, the *ad*-queue collects the complete set of frequent-item projections containing items *a* and *d*.

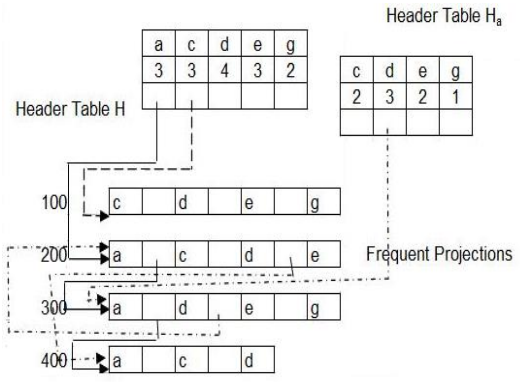

Figure 5. Header table *Ha* and the *ad*-queue.

Thus, the set of frequent patterns containing items *a* and *d* can be mined recursively. Even though item *c* appears in frequent-item projections of the *ad*-projected database, it is not considered as a local frequent item in any recursive projected database since it has been considered in the mining of the *ac*-queue. This mining generates only one pattern *ade* : 2. For the search in the *ae*-projected database, since *e* contains no child links, the search terminates, with no patterns being generated. After the frequent patterns containing item *a* are found, the *a*-projected database, i.e., *a*-queue, is no longer needed for the remaining mining processes. Since the *c*-queue includes all frequent-item projections containing item *c* except for those projections containing both items *a* and *c*, which are in the *a*-queue.

To mine all the frequent patterns containing item *c* but not *a*, and other subsets of frequent patterns, we need to insert all the projections in the *a*-queue into the proper queues. The *a*-queue is traversed one more time. Each frequent-item projection in the queue is appended to the queue of the next item in the projection following *a* in the *F-list*, as shown in Figure.6.

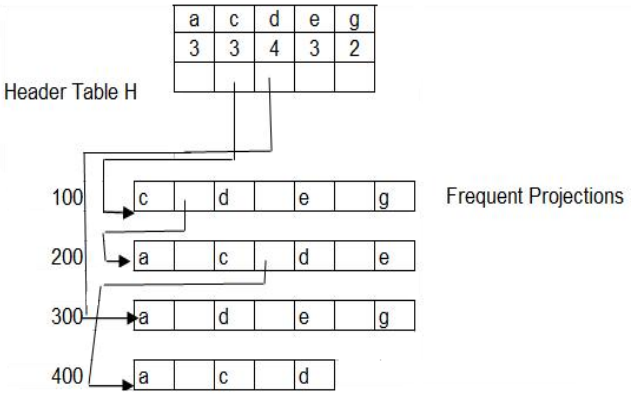

Figure 6. Adjusted hyper-links after mining the a-projected database

The efficiency of H-mine comes from the following aspects.

- H-mine avoids candidate generation and test by adopting a frequent-pattern growth methodology. H-mine absorbs the advantages of pattern growth.
- H-mine confines its search in a dedicated space. Unlike other frequent pattern growth methods, it does not need to physically construct memory structures of projected databases. It fully utilizes the

information well organized in the H-struct, and collects information about projected databases using header tables, which are light-weight structures. That also saves a lot of efforts on managing space.

- H-mine does not need to store any frequent patterns in memory. Once a frequent pattern is found, it is output to disk. [1]

3.2 Partitioning in H-mine:

H-mine mines frequent-patterns in large data sets that cannot fit in main memory. H-mine is efficient when the frequent-item projections of a transaction database plus a set of header tables can fit in main memory. When they cannot fit in memory, a database partitioning technique can be developed as follows.

Let *TDB* be the transaction database with *n* transactions and *min_sup* be the support threshold. By scanning *TDB* once, one can find *L*, the set of frequent items. Then, *TDB* can be partitioned into *k* parts, $TDB_1, \ldots, TDB_k$, such that, for each *TDBi* ($1 \leq i \leq k$), the frequent-item projections of transactions in TDB_i can be held in main memory, where *TDBi* has *ni* transactions, and $\Sigma^{k}_{i=1} ni = n$. We can apply H-mine to *TDBi* to find frequent patterns in *TDBi* with the minimum support threshold $min\_sup_i = [min\ sup \times ni/n]$.

Let *Fi* ($1 \leq i \leq k$) be the set of (locally) frequent patterns in *TDBi*. *P* cannot be a (globally) frequent pattern in *TDB* with respect to the support threshold *min sup* if there exists no *i* ($1 \leq i \leq k$) such that *P* is in *Fi*. Therefore, after mining frequent patterns in the *TDBi*, we can gather the patterns in *Fi* and collect their (global) support in *TDB* by scanning the transaction database *TDB* one more time. Thus we can extend H-mine to handle datasets whose frequent patterns cannot be held in main memory. [1]

III. RESULTS AND DISCUSSION

EXPERIMENTAL ANALYSIS

In this section, we describe the experimental results of H-Mine algorithm for generating frequent patterns.

DATASET DESCRIPTION:

Title : Mushroom Database

No of instances : 8124
No of attributes : 22 (all nominally valued)

Attribute information:

(classes: edible=e, poisonous=p) cap_shape, cap-surface, cap color, bruishes, odor, gill-attachment, gill-spacing, gill-size, gill-color, stalk-shape, stalk, root, stalk _surface _along ring, stalk color_above_ring, stalk_color_ below ring, veil type, veil color, ring_number, ring_type spore_print_colour, population, habitat.

Class distribution:

Edible : 4208(51.8%)
Poisonous: 3916(48.2%)

The main advantage of the H-Mine algorithm is that it provides the flexibility by avoiding candidate generation and test by adopting a frequent-pattern growth methodology. H-mine absorbs the advantages of pattern growth. Algorithms were coded in JAVA. The algorithm was tested for its efficiency in finding the frequent patterns. It was tested for different values of minimum support. The sample output result screens are shown in figure 7 and figure 8.The frequent pattern generation and the execution time graphs are shown in figure 9 and figure 10.

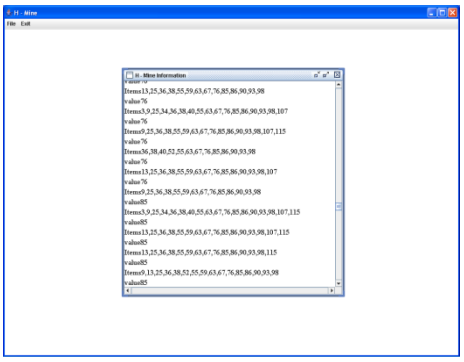

Figure 7. Frequent itemsets Generation

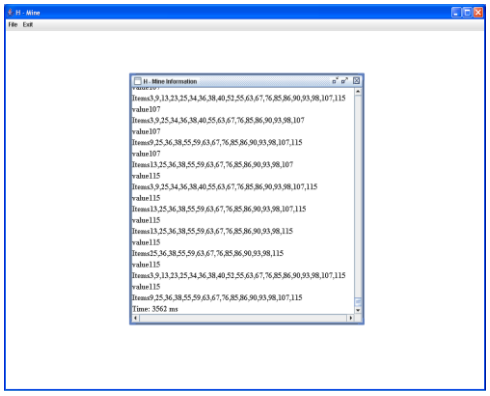

Figure 8. Execution time for generation of different frequent itemsets

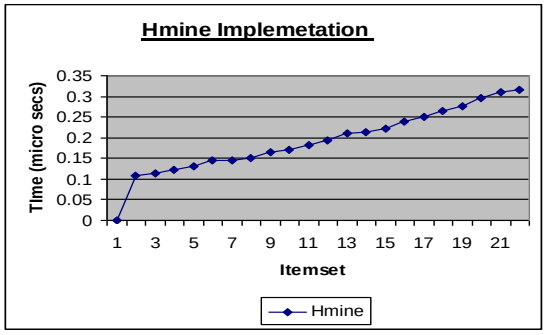

Figure 9. Frequent itemsets Generation

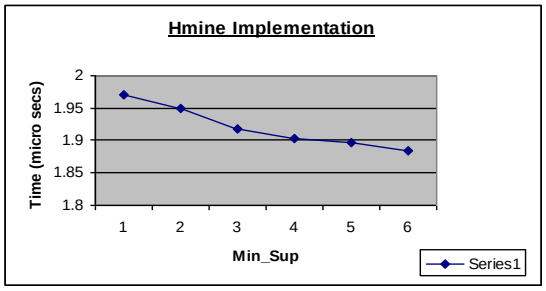

Figure 10. Execution time for generation of different frequent itemsets

IV. CONCLUSION

The overall performance of mining association rule is determined by finding all frequent itemsets. From the experiment which we have conducted we were able to find frequent itemsets from the mushroom dataset depending upon support values which were given as input by the user. Our experimental results show that the performance of the algorithm is determined by finding all frequent itemsets and is influenced by various support factors. The results show that the algorithm takes advantage of the H-struct data structure and dynamically adjusts links in the mining process.

V. REFERENCES

[1] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. Hmine: Hyper-structure mining of frequent patterns in large databases. In ICDM, pages 441--448, 2001.

[2] UCI Repository of Machine Learning databases, University of California, Irvine, Department of Information and Computer Science.http://www.ics.uci.ed/~mlearn/MLRepository.html

[3] Jaiwei Han and Micheline Kamber, "Datamining: Concepts and Techniques", Morg Kaufman Publishers, 2001.

[4] R.Agrawal, T.Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases", Proc.1993. ACM SIGMOD Int'l. Conf. Management of Data, pp.207-216, Washington, D.C., May, 1993.

[5] R.Agrawal and R.Srikant, "Fast Algorithms for Mining Association Rules", In Proc.1994 Int'l. Conf. Very Large Databases, pp.487-499.Santiago, Chile, Sep, 1994.

[6] Jiawei, Han and Yongjian Fu," Discovery of Multiple-level Association rules from Large Databases". In Proc.1995, Int'l. Conf. Very Large Databases, pp.420-431, Zurich, Switzerland, Sep, 1995.

[7] M. J. Zaki, "Scalable algorithms for association mining", IEEE Transactions on Knowledge and Data Engineering, 12(3):372-390, May-June 2000.

[8] Yanbo Wang – "Categorization of Association Rule Mining Algorithms", In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, 2003.

[9] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules", In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, August 1997.

[10] R. J. Bayardo, "Efficiently mining long patterns from databases", In ACM SIGMOD Conf. Management of Data, June 1998.