ISSN:0975-9646

Incremental Mining of Association Rules: A Survey

Siddharth Shah[*] ,N. C. Chauhan[#,]S. D. Bhanderi[#]

[*]*B.V.M.Engineering College, V.V.Nagar, Gujarat, India*
[#]*A.D.Patel Institute of Technology, New V.V.Nagar, Gujarat, India*

Abstract-The association rule mining has been very useful in many applications such as, market analysis, web data analysis, decision making, knowing customer trends etc. In transactional databases as time advances, new transactions are being added and obsolete transactions are discarded. Incremental mining deals with generating association rules based on available knowledge (obtained from mining of previously stored databases) and incremented databases only, without scanning the previously mined databases again. Several research works have been carried out for deriving the association rules and maintaining them efficiently without re-scanning the complete database. In this paper, a survey on different algorithms designed for incremental mining is presented. The algorithms are discussed into two sub-categories namely, apriori based algorithms and tree based algorithms. The pros and cons of these algorithms are also discussed in brief.

1. INTRODUCTION

Due to the increasing use of large data with high computation required for various applications, the importance of data mining has grown rapidly. From the point of view of business application, analysis of previous transaction data can provide valuable information on behavior of customer, and thus help in making business decisions. Thus it is necessary to collect and analyze a sufficient data properly before making any decisions. Since the amount of data being processed is large, it is important for the mining algorithms to be very computationally efficient. Various data mining algorithms have been explored in the literature [1–6]. Recently many important applications have created the need of incremental mining. This is due to the increasing use of the record-based databases where data is being continuously added e.g., super market data, stock market data, sales data, and weather/traffic records, etc. In the incremental mining, data are not only added but also obsolete data are being deleted. The aim of incremental mining techniques is to re-run the mining algorithm on the only updated database. The overall process of incremental mining is summarized in Fig. 1. However, it is obviously less efficient since previous mining rules are not utilized for discovering new rules while the updated portion is usually small compared to the whole dataset. Consequently, the efficiency and the effectiveness of algorithms for incremental mining are both crucial issues. Algorithms should be such that only updated transactions and previous mined rules to be taken into account for generating new rules. In the next section various algorithms designed for incremental mining have been discussed. The most of algorithms have been classified into majorly two categories: Apriori based algorithms, and Tree based algorithms

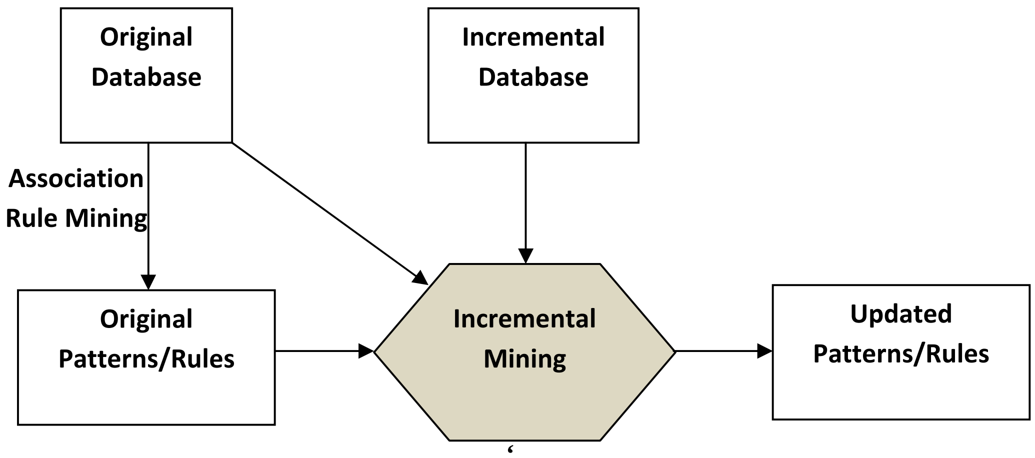

Fig 1: Process of incremental mining[19]

.

2. APRIORI-BASED INCREMENTAL MINING ALGORITHMS

2.1 FUP Algorithm and Its Variations

Algorithm FUP (Fast UPdate) [7] is the first algorithm proposed for incremental mining of association rules. It deals with databases with transaction insertion only, but is not able to solve the same with transaction deletion. Specifically, given the original database D and its corresponding frequent itemsets $L = \{L_1, ..., L_k\}$. the algorithm reuses the information to efficiently obtain the new frequent itemsets $L' = \{L'_i, ..., L'_k\}$ on the new database $D' = D \cup D+$. Here, D is the original database, $\Delta+$ are the transactions added, $\Delta-$ are transactions deleted, D– is set of transactions left after deletion and D' is incremented database. By utilizing the definition of support and the constraint of minimum support S_{min}. The following lemmas are generally used in algorithm FUP.

1. An original frequent itemset X, i.e., $X \in L$, becomes infrequent in D' if and only if X.support$D' < S_{min}$.
2. An original infrequent itemset X, i.e., X /∈ L, may become frequent in D_0 only if X.support$\Delta+ \geq S_{min}$.
3. If a k-itemset X whose (k-1)-subset(s) becomes infrequent, i.e., the subset is in L_{k-1} but not in L'_{k-1}, X must be infrequent in D'.

FUP can update the association rules in a database when new transactions are added to the database, contains a number of iterations [13, 14]. The candidate sets at each iterations are generated based on the frequent itemsets found in the previous iteration. At the k-th iteration of FUP, $\Delta+$ is scanned exactly once. For the original frequent itemsets, they only have to be checked against the small increment $\Delta+$. To discover the new frequent itemsets, the set of candidate itemsets C_k is firstly extracted from $D+$, and then be pruned according to the support count of each candidate itemset in $\Delta+$. Moreover, the pool for candidate itemsets can be further reduced by discarding itemsets whose (k–1)-subsets are becoming infrequent.

Cheung, et. al. [8] proposed a new algorithm FUP_2 which is an extension of FUP algorithm. The FUP updates the association rules in a database when new transactions are added to the database whereas FUP_2 updates the existing association rules when transactions are added to and deleted from the database. FUP_2 is similar to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. A very feature is that the old frequent k itemsets L_k from the previous mining result is used for dividing the candidate set C_k into two parts: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. In other words, P_k and Q_k are the sets of candidate itemsets that are previously frequent and infrequent with respect to D. For the candidate itemsets in Q_k, their supports are unknown since they were infrequent in the original database D, posing some difficulties in generating new frequent itemsets. It is noted that if a candidate itemset in Q_k is frequent in $\Delta-$, it must be infrequent in $D-$. This itemset is further identified to be infrequent in the updated database D' if it is also infrequent in $\Delta+$. This technique helps on effectively reducing the number of candidate itemsets to be further checked against the unchanged portion $D-$ which is usually much larger than $\Delta-$ or $\Delta+$.

2.2 UWEP (Update With Early Pruning)

In [9], algorithm UWEP has been proposed which uses the technique of update with early pruning. The advantage of algorithm UWEP over other FUP-based algorithms is that it prunes the supersets of an originally frequent itemset in D as soon as it becomes infrequent in the updated database D', rather than waiting until the k-th iteration. In addition, only itemsets which are frequent in both $\Delta+$ and $D' = (D \cup \Delta+)$ are taken to generate candidate itemsets to be further checked against $\Delta+$. If a k-itemset is frequent in $\Delta+$, but infrequent in D', it is not considered when generating C_{k+1}. This significantly reduces the number of candidate itemsets in $\Delta+$. Consequently, these early pruning techniques can enhance the efficiency of FUP-based algorithms.

2.3 *Algorithm Utilizing Negative Borders*

The concept of negative borders [10] has been utilized in [11] to improve the efficiency of FUP-based algorithms on incremental mining. Given a collection of frequent itemsets L, the negative border $B_d-(L)$ of L consists of the minimal itemsets $X \subseteq R$ not in L where R is the set of all items. In other words, the negative border consists of all itemsets that were candidates of the level-wise method which did not have enough support. That is, $B_d-(L_k) = C_k - L_k$ where $B_d-(L_k)$ is the set of k-itemsets in $B_d-(L)$. The intuition behind the concept is that given a collection of frequent itemsets, the negative border contains the "closest" itemsets that could be frequent, too.

The algorithm proposed in [11] first generate the frequent itemsets of the increment portion $\Delta+$. A full scan of the whole dataset is required only if an itemset outside the negative border gets added to the frequent itemsets or its negative border. Even in such cases, it requires only one scan over the whole dataset. The drawback is that to compute the negative border closure may increase the size of the candidate set. However, a majority of those itemsets would have been present in the original negative border or frequent itemset. Only those itemsets which were not covered by the negative border need to be checked against the whole dataset. As a result, the size of the candidate set in the final scan could be much smaller as compared to algorithm FUP.

2.4 MAAP (Maintaining Association rules with Apriori Property) and PELICAN

Several other algorithms, including the MAAP algorithm [12], and the PELICAN algorithm [13], are proposed to perform incremental mining. Algorithm MAAP firstly finds the frequent itemset(s) of the largest size based on previously discovered frequent itemsets. If a *k*-itemset is found to be frequent, then all of its subsets are also frequent and are thus added to the new set of frequent itemsets *L'*. This eliminates the need to compute some frequent itemsets of shorter sizes. The other frequent itemsets are then identified by following the levelwise itemset generation. Both algorithms MAAP and PELICAN are similar to algorithm FUP_2, however, their main goal is to maintain maximum frequent itemsets when the database is updated. The algorithms do not consider non-maximum frequent itemsets, and therefore, the counts of non-maximum frequent itemsets cannot be calculated. The difference of these two algorithms is that MAAP calculates maximum frequent itemsets by Apriori-based framework, while PELICAN calculates maximum frequent itemsets based on vertical database format and lattice decomposition. Since these two algorithms maintain maximum frequent itemsets only, the storage space and the processing time for performing each update can be thus reduced.

3. TREE-BASED INCREMENTAL MINING ALGORITHMS

3.1 DB-tree and PotFp-tree Algorithms

In [14], DB-tree and PotFp-tree have been proposed in order to achieve incremental mining. The algorithm DB-tree, stores all the items in an FP-tree rather than only frequent 1-itemsets in the database. Besides, the construction of a DB-tree is exactly the same way as that of a FP-tree. Consequently, the DB-tree can be seen as an FP-tree with minimum support = 0. When new transactions are added, corresponding branches of the DB-tree could be adjusted or new branches may be created. On the other hand, when old transactions are deleted, corresponding branches are also adjusted or removed. This retains the flexibility to accommodate the FP-tree to database changes when performing incremental mining. However, since the whole dataset being considered could be quite large, a much more space could be needed to maintain this DB-tree structure even a high compression is made by the nature of tree projection. This drawback may cause the problem of insufficient memory even more severe when the size of the DB-tree is far above the memory capacity. The other algorithm proposed in [14] is the PotFp-tree, which stores only some potentially frequent items in addition to the frequent 1-itemsets at present. A tolerance parameter *t* is used to decide if an item is potentially frequent. Therefore, the need to scan the whole old database in order to update the FPtree when updates occur is likely to be effectively reduced. The PotFp-tree is seeking for the balance of required extra storage and possibility of re-scanning the dataset. Since FP-tree is a subset of either the DB-tree or the PotFptree, for mining frequent itemsets, the FP-tree is firstly projected from either the DB-tree or the PotFp-tree. The frequent itemsets are then extracted from the FP-tree in the way described in [15].

3.2 FELINE (Frequent /Large patterns mining with CATS tree)

In [16], the CATS tree (compressed and arranged transaction sequences tree) has been proposed which has several common properties of FP-tree. Also, the CATS tree and the DB-tree are very alike since they both store all the items no matter they are frequent or not. This feature enables the CATS tree to be capable of avoiding re-scans of databases when updates occur. However, the construction of the CATS tree is different to that of an FP-tree and a DB-tree. Specifically, the FP-tree is built based on the ordering of global supports of all frequent items, while the CATS-tree is built based on the ordering of local supports of items in its path. Consequently, the CATS-tree is sensitive to the ordering of input transactions, making the CATS-tree not optimal since no preliminary analysis is done before the tree construction. This in turns can reduce the data scan required to only once, which is the advantage of this algorithm.

3.3 CAN Tree (Canonical – order Tree)

In [17] a novel tree structure, called CanTree (**Can**onical-order **Tree**) has been proposed which captures the content of the transaction database and orders tree nodes according to some canonical order. The construction of the CanTree only requires one database scan as compared to an FP-tree which requires two database scans. In CanTree, items are arranged according to some canonical order, which can be determined at runtime during the mining process. Specifically, items can be arranged in lexicographic order or, items can also be arranged according to some specific order depending on the item properties (e.g., price, validity, etc.) which are frequency independent ordering. Items can also be arranged according to some fixed frequency-related ordering (e.g., in descending order of the global frequency of the "original" database *DB*). Once the ordering is determined, items will follow this ordering in CanTrees for subsequently updated database even the frequency ordering of items in these updated databases is different from *DB*. With this canonical ordering of items, some properties [17], are described below.

Property 1**)** The ordering of items is unaffected by the changes in frequency caused by incremental updates.Property 2**)** The frequency of a node in the CanTree is at least as high as the sum of frequencies of its children.Now with these properties of CanTree, transactions can be easily added to the CanTree without searching for merge-able paths. As canonical order is fixed, any changes in frequency caused by incremental updates (e.g., insertions, deletions, and/or modifications of transactions) will not affect the ordering of items in the CanTree at all. Also, swapping of tree nodes which often leads to merging and splitting of tree nodes is *not* required. Once the CanTree is constructed, mining frequent patterns from the tree is similar to FP-growth. Since items are consistently arranged according to some canonical order, the inclusion of all *frequent* items using just upward traversals is guaranteed. There is no worry about possible omission or doubly-counting of items. Hence, for CanTrees, there is no need for having both upward and downward traversals which significantly reduces computation.

4 Discussions

Mining of association rules can provide very valuable information, and improve the quality of business decisions. Many incremental mining algorithms have been proposed by different researchers in accordance with the need of applications which uses record based database and where database grows rapidly. The overall approach towards incremental mining is to make use of previously mined knowledge and scan only incremented database. Most of the algorithms try to reduce the number of scans of database and maintain the association rules efficiently. Apriori based algorithms like FUP and FUP_2 requires two complete scans of database and are computationally less efficient due to candidacy generation. Tree based algorithms, like CAN tree and FELINE, require only single scan of database. The FELINE algorithm requires swapping, merging and splitting of tree nodes, since it uses frequency dependent ordering and this drawback has been overcome in CAN tree. FELINE also takes large computation time in finding merge-able paths and needs downward traversals during mining. In all, many algorithms have contributed to achieve incremental mining, however yet there are scopes to improve the efficiency of algorithms, development of new algorithms, and to reduce number of scans of databases.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pages 207—216, May 1993.

2. R. Agrawal and R. Srikant. Mining Sequential Patterns. Proceedings of the 11th International Conference on Data Engineering, pages 3—14, March 1995.

3. J. M. Ale and G. Rossi. An Approach to Discovering Temporal Association Rules. Proceedings of the 2000 ACM Symposium on Applied Computing, pages 294—300, March 2000.

4. M.-S. Chen, J. Han, and P. S. Yu. Data Mining: An Overview from Database Perspective. IEEE Transactions on Knowledge and Data Engineering, 8(6):866— 883, December 1996.

5. M.-S. Chen, J.-S. Park, and P. S. Yu. Efficient Data Mining for Path Traversal Patterns. IEEE Transactions on Knowledge and Data Engineering, 10(2):209— 221, April 1998.

6. X. Chen and I. Petr. Discovering Temporal Association Rules: Algorithms, Language and System. Proceedings of the 16th International Conference on Data Engineering, 2000.

7. D. Cheung, J. Han, V. Ng, and C. Y. Wong. Large Databases: An Incremental Updating Technique. Proceedings of the 12th International Conference on Data Engineering, pages 106—114, February 1996.

8. D. Cheung, S. D. Lee, and B. Kao. A General Incremental Technique for Updating Discovered Association Rules. Proceedings of the Fifth International Conference On Database Systems for Advanced Applications, pages 185—194, April 1997.

9. N. F. Ayan, A. U. Tansel, and M. E. Arkun. An Efficient Algorithm to Update Large Itemsets with Early Pruning. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 287—291, August 1999.

10. H. Toivonen. Sampling Large Databases for Association Rules. Proceedings of the 22th International Conference on Very Large Data Bases, pages 134—145, September 1996.

11.S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases Proceeding of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 263—266, August 1997.

12. Z. Zhou and C. I. Ezeife. A Low-Scan Incremental Association Rule Maintenance Method. Proceedings of the 14th Canadian Conference on Artificial Intelligence, June 2001.

13. A. Veloso, B. Possas, W. M. Jr., and M. B. de Carvalho. Knowledge Management in Association Rule Mining. Workshop on Integrating Data Mining and Knowledge Management (in conjuction with ICDM2001), November 2001.

14. C. I. Ezeife and Y. Su. Mining Incremental Association Rules with Generalized FP-Tree. Proceedings of the 15th Canadian Conference on Artificial Intelligence, May 2002.

15. J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data, May 2000.

16. W. Cheung and O. R. Zaiane. Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint. Proceedings of the 7th International Database Engineering and Application Symposium, July 2003.

17. C. K. Leung, Q. I. Khan and T. Hoque. CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns, Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05), 2005.

18. C.-H. Lee, C.-R. Lin, and M.-S. Chen. Sliding-Window Filtering: An Efficient Algorithm for Incremental Mining. Proceeding of the ACM 10th International Conference on Information and Knowledge Management, November 2001.

19. V. Pudi. Data Mining: Concepts and Techniques, Oxford University Press, Jan-2009