Association Rule Mining Algorithms through Vertical and Horizontal Data Layouts: Implementation and Performance Comparison

Radhika Garg[1], Rakesh Kumar Joon[2], Poonam[3]

[1]Vaish College of Engineering, Rohtak, Haryana, India
[2]Ganga Institute of Technology and Management, Kablana, Haryana, India
UIET, MDU, Rohtak, Haryana, India

Abstract:

Data mining is used to discover interesting and previously unknown patterns from datasets.Association rule mining is a popular and well researched method of data mining for discovering interesting relations between items in the databases. Finding frequent itemsets is an important task in data mining for extracting association rules. In this research, we have taken four association rule mining algorithms that use horizontal and vertical data formats for generation of frequent itemsets. we introduce a new association rule mining algorithm, intersect transaction algorithm that uses purely horizontal database layout and find the frequent itemsets by intersecting the transactions having a no. of items. We have also implemented Apriori and SplitMerge algorithms of association rule mining. Our Apriori implementation is the enhancement of the previous Apriori algorithm but uses the same property in finding the frequent itemsets as the previous Apriori algorithm uses. The Enhanced Apriori implementation is better in execution time as compared to the previous Apriori algorithm. The SplitMerge algorithms uses the split and merge technique to find the frequent itemsets. We have taken the Eclat (Equivalence CLASS transformation) algorithm that uses purely vertical data layout .I tested these algorithms on both real and synthetic datasets and then thoroughly investigate the strengths and weakness of these algorithms by carrying out several runtime experiments. It turns out that the runtime behavior of the algorithms is much more similar as to be expected

Keywords: *Data Mining, Horizontal Layouts, vertical Layouts*

I. INTRODUCTION

Data mining is an essential step in the process of knowledge discovery in databases (KDD), in which intelligent methods are applied in order to extract patterns[7].The knowledge discovered data mining have a variety of different types, the common patterns are: association mode, classification model, class model, sequence pattern and so on [8]. Association rule mining is the current hot.

II. ASSOCIATION RULE MINING

Association rule refers to the rules of certain association relationship between groups of objects in the database. It can be used to find the contact among the different commodities (terms) in the transaction database, and so that the behavior patterns of customer purchases will be found.

A. Association rule conception

Association rule mining can be described as following:
assuming $I = \{ i_1, i_2, \ldots i_n\}$ is n aggregates with different terms, then for a transaction database D , each element T in D is a set composed by some terms in I , $T \subseteq I$. The association rule is expressed as $X \Rightarrow Y$, hereinto, $X \subset I$, $Y \subset I$, and $X \cap Y = \Phi$. The association rule mining is to discover all condition implicative expression meeting the minimum degree of confidence and support users given, that is,

association rules. The confidence and support degree of these rules are all greater than or equal to the minimum degree of confidence and support [9].

The number of items in an itemsets is called length. Rules are defined as extended association rules of the form X → Y, where X and Y are itemsets representing the antecedent and the consequent part of the rule respectively.

Confidence. A rule has confidence c if c% of the transactions in D that contain X also contain Y.

Support. The rule has support s in D if s% of the transactions in D contain both X and Y.

B. Mining steps of association rules

The steps of the mining association rules can be roughly described by a two-step process [10, 6] .

(1) Identify all the frequent term sets. That is, to identify all the term sets whose support degree greater than pre-given support threshold.

(2)To generate strong association rules on the basis of the found frequent term sets. That is, to generate those

association rules whose support and confidence respectively greater than or equal to the pre-given support threshold and confidence threshold.

C. Mining algorithm of association rules

An important consideration in most algorithms is the representation of the transaction database. Conceptually, such a database can be represented by a binary two-dimensional matrix in which every row represents an individual transaction and the columns represent the items. Such a matrix can be implemented in several ways. The most commonly used layout is the horizontal data layout[6]. That is, each transaction has a transaction identifier and a list of items occurring in that transaction, i.e., {TID:itemset}. Another commonly used layout is the vertical data layout[6], in which the database consists of a set of items, each followed by the set of transaction identifiers containing the item, i.e., {item:TID_set}.Hence Existing mining algorithm of association rules can be broadly divided into two main categories:-Horizontal format mining algorithms and Vertical format mining algorithms. Table 1 shows both layouts. For both layouts, it is also possible to use the exact bit-strings from the binary matrix(as shown in Table2 and Table 3).

TABLE 1
HORIZONTAL AND VERTICAL DATABASE LAYOUT

TID	**List of item_IDs**
T10	Coke,Slice,Kurkure
T20	Coke, Kurkure
T30	Slice, pizza

Itemsets	**TID_set**
Coke	T10,T20
Slice	T10,T30
Kurkure	T10,T20
Pizza	T30

TABLE 2
HORIZONTAL DATA LAYOUT IN BINARY FORMAT

	Coke	Slice	Kurkure	Pizza
T10	1	1	1	0
T20	1	0	1	0
T30	0	1	0	1

TABLE 3
VERTICAL DATA LAYOUT IN BINARY FORMAT

	Coke	Slice	Kurkure	Pizza
T10	1	1	1	0
T20	1	0	1	0
T30	0	1	0	1

To count the support of an itemsets X using the horizontal database layout, we need to scan the database completely, and test for every transaction T whether X ⊆T. An important misconception about frequent pattern mining is that scanning the database is a very I/O intensive operation. However, in most cases, this is not the major cost of such counting steps. Instead, updating the supports of all candidate itemsets contained in a transaction consumes considerably more time than reading that transaction from a file or from a database cursor.

A number of vertical mining algorithms have been proposed recently for association mining, which has shown to be very effective and usually outperform horizontal approaches. The main advantage of the vertical format is support for fast frequency counting via intersection operations on transaction ids (tids) and automatic pruning of irrelevant data. The main problem with these approaches is when intermediate results of vertical tid lists become too large for memory, thus affecting the algorithm scalability.

III. HORIZONTAL AND VERTICAL DATA FORMAT ALGORITHMS

The algorithms that use horizontal data format are:-

- Apriori
- Split and Merge
- Intersecttransaction

.The main algorithm that uses the vertical format is:-

- Eclat

A. Apriori Algorithm

We present an extension of the Apriori algorithm[1,2,4] called enhancedApriori without additional computation time.The enhancedApriori uses a data structure that directly represents a prefix tree.This tree is grown top-down level by level, pruning those branches that cannot contain a frequent itemset. The prefix tree is not only an efficient way to store the counters, it also makes processing the transactions very simple, especially if we sort the items in a transaction ascendingly w.r.t. their identifiers. Then processing a transaction is a simple doubly recursive procedure: To process a transaction for a node of the tree, (1) go to the child corresponding to the first item in the transaction and process the remainder of the transaction recursively for that child and (2) discard the first item of the transaction and process it recursively for the node itself (of course, the second recursion is more easily implemented as a simple loop through the transaction). In a node on the currently added level, however, we increment a counter instead of proceeding to a child node. In this way on the current level all counters for item sets that are part of a transaction are properly incremented. the recursion is a very expensive procedure and therefore it is worthwhile to consider how it can be improved. One approach is based on the fact that often there are several similar transactions, which lead to a similar program flow when they are processed. By organizing the transactions into a prefix tree transactions with the same prefix can be processed together.

Advantages

- “Perfect” pruning of infrequent candidate item sets

Disadvantages

- Can require a lot of memory since all frequent itemsets are represented.
- Support counting takes very long for large transactions.

B. Split and Merge Algorithm

The split and merge algorithm[5] uses only a single transaction list stored as an array. This array is processed with a simple split and merge scheme, which computes a conditional database, processes this conditional database recursively, and finally eliminates the split item from the original (conditional) database.

The main steps are:-

a) Take the transaction database in its original form
b) Then the frequencies of individual items are determined from this database in order to be able to discard infrequent items immediately.
c) The items in each transaction are sorted according to their frequency in the transaction database .
d) The transactions are sorted lexicographically into descending order. Here the item with the higher frequency precedes the item with the lower frequency.
e) The data structure on which Split and merge algorithm operates is built by combining equal transactions and setting up an array, in which each element consists of two fields: an occurrence counter and a pointer to the sorted transaction. This data structure is then processed recursively to find the frequent item sets.

The basic operations of recursive processing are:-

a) Split Step:
   - Move all transactions starting with the same item to a new array.
   - Remove the common leading item (advance pointer into transaction).
b) Merge Step:
   - Merge the rest of the transaction array and the copied transactions.
   - The merge operation is similar to a mergesort phase.

If the transaction database is sparse, the two transaction arrays to merge can substantially differ in size. In this case SaM can become fairly slow, because the merge step processes many more transactions than the split step.

Advantages

- Very simple data structure and processing scheme.
- Easy to implement for operation on relational databases.

Disadvantages

- Can be slow on sparse transaction databases due to the merge step.

C. Intersecttransaction Algorithm

We developed the Intersecttransaction algorithm[11] to find frequent item sets by intersecting transactions, which is based on the insight that an item set is closed if it is the intersection of all transactions that contain it. This approach can be highly competitive in special cases, namely if the data set is sparse or very sparse ,that is, the sum of the transaction sizes divided by the product of the number of items and the number of transactions is very small or the number of transactions is fairly small (a few thousand to a few tens of thousands) and the user-specified minimum support is small or very small. For large minimum support values and dense data sets, however, it is not a recommendable approach.

Steps:-

a. Remove items of short transactions then sort the transaction lexicographically.
b. Reduce transactions to the unique ones.
c. Create an item frequency array.
d. Create the prefix tree.
e. Traverse the transactions and then intersect the transactions and the tree.
f. If itemsets could be pruned then count newly prunable items and prune the prefix tree.
g. Find the frequent itemsets and then print them.

Advantages

- Fast speed.
- Useful only for datasets of small size i.e. for sparse datasets.

Disadvantages

- Not successful for dense datasets.

D. Eclat Algorithm

Eclat represents the set of transactions as a sparse bit matrix and intersects rows to determine the support of item sets. The item sets in Eclat[11] are checked in depth-first traversal of the prefix tree. The search scheme is the same as the general scheme for searching with canonical forms having the prefix property and possessing a perfect extension rule (generate only canonical extensions) Eclat generates more candidate item sets than Apriori, because it does not store the support of all visited item sets. As a consequence it cannot fully exploit the Apriori property for pruning. Eclat uses a purely vertical transaction representation. No subset tests and no subset generation are needed to compute the support. The support of item sets is rather determined by intersecting transaction lists. The transaction lists can be compressed by combining consecutive transaction identifiers into ranges. Exploit item frequencies and ensure subset relations between ranges from lower to higher frequencies, so that intersecting the lists is easy. In a conditional database, all transaction lists are “filtered” by the prefix. Only transactions contained in the transaction list for the prefix can be in the transaction lists of the conditional database. This suggests the idea to use diffsets to represent conditional databases

Basic Processing Scheme

- Depth-first traversal of the prefix tree.
- Data is represented as lists of transaction identifiers (one per item).
- Support counting is done by intersecting lists of transaction identifiers.

Advantages

- Depth-first search reduces memory requirements.
- Usually (considerably) faster than Apriori.

Disadvantages

- With a sparse transaction list representation (row indices) eclat is difficult to execute for modern processors.

IV. EXPERIMENTAL RESULTS

We ran experiments with all four programs on four datasets , which exhibit different characteristics, so that the advantages and disadvantages can be observed. The data sets are: Agaricus-lepiota(contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family), Connect-4(contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced), Accidents(contain a rich source of information on the different circumstances in which the accidents have occurred) and T10I4D100K ([IBM10],a synthetic data set generated with IBM’s data generator). Usually the synthetic datasets are sparse when compared to the real datasets These data sets are downloaded from UCI machine learning repository. Some characteristics of these data sets are shown in Table 4.

TABLE 4
DATABASE CHARACTERISTICS

Datasets	#Items	Avg. Length	#Records	#Size
Agaricus-lepiota	120	23	8,124	557 KB
Connect-4	130	43	67,557	8.82 MB

Accidents	468	45	340,183	33.8 MB
T10I4D100k	1000	10	100,000	3.83 MB

All experiments were performed on a Dell Intel (R) Core(TM) 2 Duo CPU ,2.09GHz with 2GB of memory, running Microsoft Windows XP. All times shown include time for outputting all the frequent itemsets. The results are presented in Fig. 1, Fig. 2, Fig.3, and Fig. 4.

TABLE 5
RUNTIME(S) FOR AGARICUS-LEPIOTA DATA

	Execution Time(s)			
Support	Enhanced-Apriori	SplitMerge	Intersect-transaction	Eclat
0.1	1.65	1.23	4.77	2
0.2	0.72	0.28	1.29	0.42
0.3	0.24	0.19	0.4	0.26
0.4	0.23	0.24	0.26	0.26
0.5	0.24	0.21	0.25	0.26
0.6	0.24	0.2	0.24	0.24
0.7	0.25	0.23	0.24	0.24
0.8	0.25	0.2	0.24	0.24
0.9	0.25	0.2	0.23	0.22
1	0.24	0.2	0.23	0.22

TABLE 6
RUNTIME(S) FOR CONNECT-4 DATA

	Execution Time(s)			
Support	Enhanced-Apriori	Split-Merge	Intersect-transaction	Eclat
0.1	1.65	1.23	4.77	2
0.2	0.72	0.28	1.29	0.42
0.3	0.24	0.19	0.4	0.26
0.4	0.23	0.24	0.26	0.26
0.5	0.24	0.21	0.25	0.26
0.6	0.24	0.2	0.24	0.24
0.7	0.25	0.23	0.24	0.24
0.8	0.25	0.2	0.24	0.24
0.9	0.25	0.2	0.23	0.22
1	0.24	0.2	0.23	0.22

TABLE 7
RUNTIME(S) FOR ACCIDENTS DATA

	Execution Time(s)			
Support	Enhanced-Apriori	Split-Merge	Intersect-transaction	Eclat
0.1	1.65	1.23	4.77	2
0.2	0.72	0.28	1.29	0.42
0.3	0.24	0.19	0.4	0.26

0.4	0.23	0.24	0.26	0.26
0.5	0.24	0.21	0.25	0.26
0.6	0.24	0.2	0.24	0.24
0.7	0.25	0.23	0.24	0.24
0.8	0.25	0.2	0.24	0.24
0.9	0.25	0.2	0.23	0.22
1	0.24	0.2	0.23	0.22

TABLE 8
RUNTIME(S) FOR T10I4D100K DATA

	Execution Time(s)			
Support	Enhanced-Apriori	Split-Merge	Intersect-transaction	Eclat
0.1	1.65	1.23	4.77	2
0.2	0.72	0.28	1.29	0.42
0.3	0.24	0.19	0.4	0.26
0.4	0.23	0.24	0.26	0.26
0.5	0.24	0.21	0.25	0.26
0.6	0.24	0.2	0.24	0.24
0.7	0.25	0.23	0.24	0.24
0.8	0.25	0.2	0.24	0.24
0.9	0.25	0.2	0.23	0.22
1	0.24	0.2	0.23	0.22

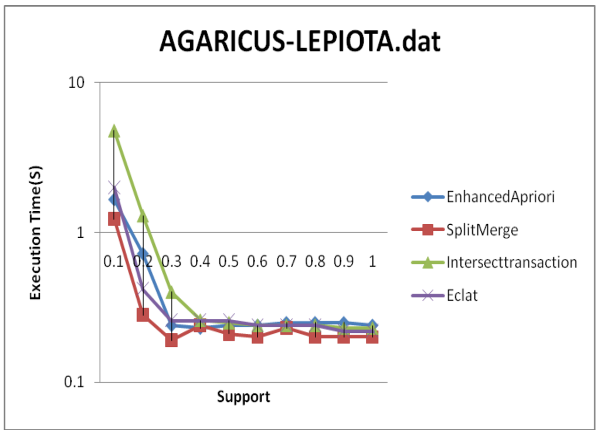

FIG. 1. Runtime for Agaricus-lepiota data

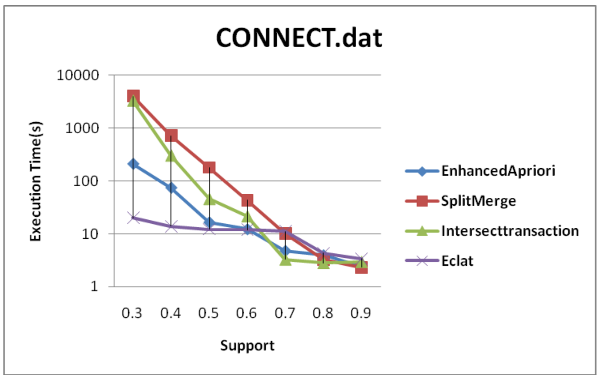

FIG. 2. Runtime for Connect-4 data

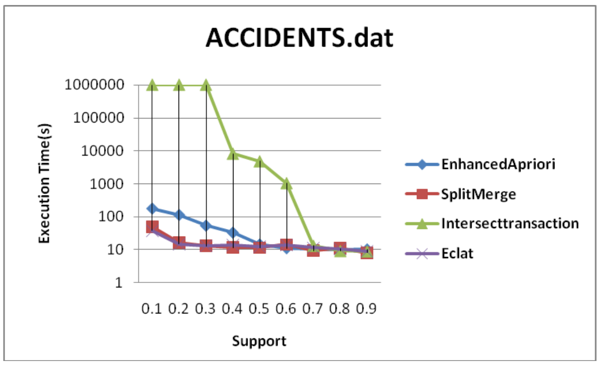

FIG. 3. Runtime for Accidents data

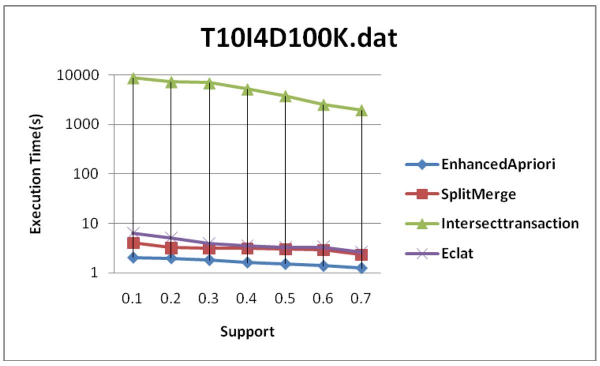

FIG. 4. Runtime for T10I4D100k data

EnhancedApriori algorithm gives good results for Agaricus-Lepiota and T10I4D100K datasets. It takes a long time to generate the frequent itemsets in case of Accidents dataset, but results are satisfactory. In case of Connect dataset, EnhancedApriori was not able to generate all the frequent itemsets at low supports even after running for long time. Performance of the algorithm not only depends on the size of the dataset, but it also depends on the average number of items per transactions. As the average number of items per transaction increases, frequent itemsets generated for each pass also increases. Hence support counting requires more time for that.

SplitMerge algorithm gives very good results for Agaricus-Lepiota and T10I4D100K datasets, satisfactory results for Accidents dataset but it takes a very long time in case of Connect dataset. So its overall performance is satisfactory.

Intersecttransaction algorithm performs good for Agaricus-Lepiota dataset , somewhat better than SplitMerge in case of Connect dataset, but it performance degraded a lot for T10I4D100K dataset and it runs out of memory for Accidents dataset and was not able to generate frequent itemsets at low support values due to the intersect operation.

Eclat algorithm outperforms other algorithms for Connect dataset . Its performance is satisfactory in Accidents, Agaricus-Lepiota and T10I4D100K datasets as that of SplitMerge algorithm.

Overall we can say that SplitMerge and Eclat outperforms other algorithms in taking less execution time, less memory consumption and less CPU utilization. Performance of EnhancedApriori is satisfactory in all cases.

MEMORY CONSUMPTION AND CPU UTILIZATION

At lower support values, more memory is consumed as well as more CPU (approx. more than 50%) is utilized because more frequent items are generated. Whereas at higher supports less memory as well as CPU is utilized.

V. CONCLUSIONS AND FUTURE WORK

In this paper we compared the main horizontal and vertical data format algorithms in detail with experiments. Vertical data format algorithms are faster than horizontal data format algorithms but sometimes become more memory consuming; also they are relatively complex in structure but simple to understand. Also they show different results on different data sets, i.e., sometimes horizontal data format algorithms shows better results than verticals .All this depend on -the kind of dataset used, i.e., sparse or dense, size of the dataset etc. intersect transaction show better result for T10I4D100k datasets (synthetic) on higher supports whereas for real datasets éclat and SplitMerge performed better and almost take less execution time. Overall The performance of éclat is better on larger datasets than other algorithms. Hence for large and real datasets, vertical data format algorithms performed well whereas for synthetic datasets and small datasets, Horizontal data format algorithms performed well. The future work is to study these variations in results of these algorithms and to make efficient them in all kind of datasets.

REFERENCES

[1] Agrawal R., Imielienski T., and Swami A. Mining Association Rules between Sets of Items in Large Databases. Proc. Conf. on Management of Data, 207–216. ACM Press, New York, NY, USA 1993
[2] Agrawal R., Srikant R. , “Fast Algorithms for Mining Association Rules,” in Proc. 20th Intl. Conf. on Very Large Data Bases (VLDB’94), Santiago de Chile, Chile, pp. 487-499, 1994
[3] Blake C.L. and Merz C.J. UCI Repository of Machine Learning Databases. Dept. of Information and Computer Science, University of California at Irvine, CA,USA 1998
[4] Bodon F.. A Fast Apriori Implementation. Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90, Aachen, Germany 2003.
[5] Borgelt C. Simple Algorithms for Frequent Item Set Mining. Advances in Machine Learning II 2010: 351-369
[6] Goethals B., “Survey on Frequent Pattern Mining,” manuscript, 2003.
[7] Han J W, Kamber M. Data Mining: Concepts and Techniques. SanFrancisco: CA. Morgan Kaufmann Publishers. 2001:2~4.
[8] Jay-Louise Weldon. Data mining and visualization. Database Programming and Design, May 1996, 9(5):21~24.
[9] Kamran Parsaye. Surveying decision support. Database Programming and Design. Apr1996, 9(4):27~33.
[10] Mannila Heikki, Data mining: machine learning, statistics and database. In Proceedings of the 8th International Conference on Scientific and Statistical Database Management, Stockholm, June 18~20, 1996.1~8.
[11] Zaki M.J. and Gouda K., “Fast Vertical Mining Using Diffsets,” Proc. Ninth ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, pp. 326-335, (2003)