Research Journal of Applied Sciences, Engineering and Technology 10(11): 1281-1286, 2015
DOI: 10.19026/rjaset.10.1823
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: December 4, 2014 Accepted: April 1, 2015 Published: August 15, 2015

Research Article

Generating Frequent Patterns from Large Datasets using Improved Apriori and Support Chaining Method

[1]P. Alagesh Kannan and [2]E. Ramaraj
[1]Department of Computer Science, Madurai Kamaraj University College, Madurai, Tamil Nadu, India
[2]Department of Computer Science and Engineering, Alagappa University, Karaikudi, Tamil Nadu, India

Abstract: In this study, generating association rules with improved Apriori algorithm is proposed. Apriori is one of the most popular association rule mining algorithm that extracts frequent item sets from large databases. The traditional Apriori algorithm contains a major drawback. This algorithm wastes time in scanning the database to generate frequent item sets. The objective of any association rule mining algorithm is to generate association rules in a fast manner with great accuracy. In this study, a modification over the traditional Apriori algorithm is introduced. This improved Apriori algorithm searches frequent item sets from the large databases with less time. Experimental results shows that this improved Apriori algorithm reduces the scanning time as much as 67% and this algorithm is more efficient than the existing algorithm.

Keywords: Apriori, ARM, association rule mining, ELCAT, frequent pattern, large datasets, support chaining

INTRODUCTION

Now a days Data mining has been widely used and unifies research in various fields such as computer science, networking and engineering, statistics, databases, machine learning and Artificial Intelligence etc. There are different techniques that also fit in this category including association rule mining, classification and clustering as well as regression Apriori algorithm is the most efficient candidate generation approach proposed by Agrawal *et al*. (1993). To count the support of item sets, it uses breadth-first search strategy and to utilize the downward closure property of support, it uses candidate generation function. Apriori algorithm is an iterative one known as level-wise search and it uses the prior knowledge of frequent item set properties in generating association rules (Agrawal *et al*., 1993). Apriori algorithm works with the following principle.

If an item set is frequent, then all of its subsets must also be frequent.

Based on this principle, the algorithm generates candidate item sets from frequent item sets. The frequency of the item sets are defined by counting their occurrence in transactions. The process of Apriori algorithm is twofold:

- First it determines the set of frequent 1-item sets
- Then these frequent item sets and the minimum confidence are used to generate association rules

Suppose {A, B, C, D and E} is a item set. All the frequent item sets are represented in the below diagram. Apriori principle says that if {C, D, E} is a frequent item set, then any transaction that contains {C, D, E} must also contains its sub sets such as {C, D}, {C, E}, {D, E}, {C},{D} and {E} must also be frequent. This is said to be Monotonicity property of Apriori algorithm (Buehrer *et al*., 2007).

Alternatively, if an item set is infrequent, then all of its super sets must also be infrequent. Suppose {A, B} is infrequent, then {A, B}, {A}, {B} are also infrequent. This reduces the exponential search known as support-based pruning (Buehrer *et al*., 2007). It also filters the candidate item sets generated in the previous iteration. This property is also known as anti-monotone property of Apriori algorithm by Yanbin and Chia-Chu (2006) and Sandhu *et al*. (2010).

The main objective of this study is to reduce the number of frequent item sets. Therefore, in order to improve the efficiency of Apriori algorithm and also reduce the time complexity.

Frequent item set generation: Before beginning the generation process, minimum support must be defined. During the frequent item set generation phase, every item in the data set is taken as candidate 1-item set initially. After counting their supports, the candidate item sets are discarded. Candidate 2-item sets are generated in the next iteration by joining two candidate 1-item sets. Next database transactions are scanned and

Corresponding Author: P. Alagesh Kannan, Department of Computer Science, Madurai Kamaraj University College, Madurai, Tamil Nadu, India

the support count for each candidate 2-item set is collected.

Care must be taken such that candidate 2-item sets are generated with only frequent candidate 1-item sets. This is because of the Apriori principle which defines that "if an item set is frequent, then all of its sub sets must also be frequent". Agrawal *et al*. (1993) describes the Set of frequent 2-item sets from the set of candidate 2-item sets are determined in the next step. This is done by selecting those candidate 2-item sets that are having minimum support.

Apriori property is used only from generating candidate 3-item sets. In order to generate candidate 3-item sets, two frequent candidate 2-item sets are joined and resulting set is the candidate 3-item set. Pruning is performed to reduce the size of candidate 3-item set by Fang *et al*. (2009). It also helps us to avoid heavy computation due to large candidate item set by Buehrer *et al*. (2007).

The above procedure is repeated till the algorithm generates empty candidate item sets i.e., having found all the frequent items. In the next phase, these generated frequent item sets are used to get strong association rules. El-Hajj and Zaiane (2006) described the meaning of strong association rules is that the rules that satisfy both minimum support and minimum confidence.

METHODOLOGY

Generating association rules from frequent item sets: For each frequent item set, generate all non-empty sub sets of the frequent item sets. Calculate count of item set divided by count of subsets. If this value is greater than or equal to minimum confidence threshold defined by the user, include the new rule else reject. For example, let sc is the subset confidence, let (I_1, I_2,… I_k) be the item set. Let ms be the minimum support. Let the new rule R represented as R: $I_1 \hat{} I_2 \Rightarrow I_5$. The confidence of rule R is calculated as:

$$confidence(R) = \frac{sc\{I_1, I_2, I_5\}}{sc\{I_1, I_2\}}$$

If the output value is greater than minimum support, accept the rule else reject i.e.:

```
If confidence (R) ≥ms
        accept
else
        reject
```

Algorithm apriori:

```
Procedure Apriori (D, ms)
    // D is the transactional database and ms is the minimum support

    k = 1
$F_k = \{i / i \in I \wedge \sigma(\{i\}) \geq N \times ms\}$
// To find all the frequent 1-item sets
        repeat
        $k = k+1$
        $C_k$ = apriori_gen ($F_{k-1}$)
// To generate candidate item sets
for each transaction t in D do
// Increment the count of all candidates in Ck that are contained in t
$C_t$ = subset ($C_k$, t)
//Identify all candidates that belong to t
for each candidate item set belonging to $C_t$ Do
        σ (c) = σ (c) +1
            // Increment support count
        end for
        end for
$F_k = \{c / c \in C_k \wedge \sigma(\{c\}) \geq N \times ms\}$
// To extract frequent k-item sets
        Until $F_k = \varphi$
        Return $U_k F_k$
```

The subset function which is described in the above procedure determines all the candidate item sets in C_k that are available in each database transaction. The algorithm eliminates all candidate item sets whose support count is less than minimum support (Li *et al*., 2008). The algorithm stops when no more frequent item sets are generated. Apriori algorithm is a level wise algorithm since it traverses the item set lattice one level at a time beginning from frequent 1-item sets to frequent k-item sets, where k is the maximum size of frequent item sets by Li *et al*. (2008). Moreover, it adopts generate and test strategy since new candidate item sets are generated from frequent item sets found in the previous iteration.

Candidate item set generation and pruning: The process of generating new candidate k-item sets from frequent (k-1) item sets generated in the previous iteration is said to be candidate item set generation process. Pruning eliminates some of the candidate k-item sets using support-based pruning strategy. The effectiveness of the candidate item set generation procedure is analyzed by the following features:

- The algorithm must not generate unnecessary candidate item sets.
- It must generate complete candidate set, partial or incomplete candidate item set are of no use for generating association rules.
- There should not be any repetition in candidate item set generation process. The process should be unambiguous.

Support counting: It is the process of determining the frequency of occurrence of a particular candidate item set that is used in candidate pruning step. One way of doing this is to compare each transaction against every candidate item set and update support counts. Another way to perform support counting is to calculate the item sets contained in each transaction and update the support counts of their respective candidate item sets by Fakhrahmad *et al*. (2007).

For Apriori algorithm, support counting is performed by using hash table. For this, candidate item sets are partitioned into different buckets and stored in a hash tree. Item sets that are contained in each transaction are hashed according to their buckets. Candidate item sets are matched with other candidate item sets within the same bucket.

COMPUTATIONAL COMPLEXITY

The Apriori algorithm's efficiency can be affected by the following factors.

Low support value: Low support threshold values often results more frequent data items. Since frequent item sets increases, algorithm needs more passes over the database resulting poor computational efficiency.

Dimensions of database: If the dimensionality of the data increases, the computation and I/O costs will increase.

Number of transaction: Efficiency of the Apriori algorithm decreases with large number of transactions. Average transaction width is also having impact on Apriori algorithm efficiency. As average transaction width increases, more candidate item sets must be examined.

Limitations of apriori algorithm: Though Apriori algorithm is clear and simple, it contains certain drawbacks such as:

- Needs several iterations of the data
- It uses uniform minimum support threshold
- Difficult to find rarely occurring events
- Poor focus on partition and sampling strategies

The main disadvantage of Apriori algorithm is wastage of time to hold a vast number of candidate item sets by Liu *et al*. (2007). Apriori algorithm is inefficient and slow when the memory capacity is limited and transactions are numerous.

Improved apriori algorithm: There are so many ways by which efficiency of the Apriori algorithm can be enhanced. For counting item set, hash function can be used because an item set whose corresponding hashing bucket count is below the threshold cannot be frequent. If frequent item sets are minimal, Apriori algorithm's efficiency increases by Ye and Chiang (2006). Kessl and Tvrdk (2007) introduce a method by which efficiency can be increased is by reducing number of transactions. A database transaction is useless if it does not contain any frequent k-item set.

To sum up, all the efficiency enhancing methods for Apriori algorithm concentrate mainly on reducing the number of frequent item sets. Therefore, in order to improve the efficiency of Apriori algorithm, we must reduce the time spent for searching frequent item sets from the database.

Our improved Apriori algorithm works by scanning all transactions to generate frequent item set. The contents are items, corresponding support and their transaction IDs in the database. Next, construct candidate item set and identify the target transaction to generate candidate item sets with minimum support. Suppose T is a set of transactions such that T = {T_1, T_2, T_3 …. T_m} where m≥1. In each transaction, set of items are defined as Ti = {I_1, I_2, ….. I_n} where n≥1. For an item set, support count which is represented by σ, is the frequency of occurrence of an item set in transactions. Suppose C_k is the candidate item set of size k and L_k represents frequent item set, then our improved Apriori algorithm works as follows.

First we scan all the Transactions (T) and generate L1 containing items, support count and transaction IDs where the items are found. Next we generate candidate 2-item sets C_2, by joining $L_1 \times L_1$. Before performing remaining scan, use L_1 to get the transaction IDs of those transactions which have minimum support count between x and y. Here x and y are the items belonging to candidate 2-item sets. Scanning is one in C_2 for only these specific transactions.

The above procedure is repeated for C_3 i.e., candidate 3-item set with three items such as x, y and z. These three items belongs to C_3 and to get transaction IDs of minimum support count between x, y and z, L_1 is used:

```
Improved_Apriori (T, I, minsupport, TID)
// Generate items, items support, their transaction ID
$L_1$ = find_frequent_1_itemsets (T);
for ($k$ = 2; $L_{k-1} \neq \varphi$; $k$++)
{
// Generate candidate item set $C_k$ from $L_{k-1}$
        $C_k$ = candidates generated from $L_{k-1}$;
        // is Cartesian product $L_{k-1} \times L_{k-1}$ and eliminating
any $k$-1 size
        // item set that is not frequent
        $x$ = get_item_min_sup ($C_k$, $L_1$);
        // Get the target transaction IDs that contain $x$
        t = get_transaction_ID ($x$);
for (each transaction t in D) do
{
// Increment the count of all candidates in $C_k$ that are
contained in $t$
      $L_k$ = candidates in $C_k$ with min support;
} // end for
} // end for
return $U_k$ $L_k$;
```

Rule generation: Once frequent item sets are generated, efficient association rules are extracted from this set. For each frequent k-item set, 2^k-2 association rules can be generated. From this, rules that have empty antecedents or consequents can be ignored. Suppose if we have a frequent k-item set Y, then the association rules are generated by partitioning the item set Y into two non-empty sub sets, X and Y-X. The rules that are represented as $\varphi \Rightarrow Y$ or $Y \Rightarrow \varphi$ are ignored.

To generate association rules using Apriori algorithm, level-wise approach is used. Here level refers to the number of items belonging to the rule consequent. All the high-confidence rules that have only one item are extracted initially. Suppose if {a, c, d} → {b} and {a, b, d} → {c}, then these two high-confidence rules used to generate {a, d} → {b, c} candidate rule simply by merging consequents of two rules. Algorithm for generating association rules based on Apriori method is given below:

```
        Rule Generation _Apriori
for each frequent k-item set $f_k$ $k \geq 2$ do
        $H_1 = \{i \mid i \in f_k\}$
// 1-item set consequents of the rule
        Call ap_gen_rules ($f_k$, $H_1$)
// calling function to generate association rules
        end for
```

The above procedure calls a function that generates association rules and passes the rules to this function. In the algorithm, we use minconf as the minimum confidence threshold value described by the user. The procedure which generates association rules are as follows:

```
    Procedure ap_gen_rules ($f_k$, $H_m$)
    K = $|f_k|$
    //size of the frequent item set
    $m = |H_m|$
    //size of the rule consequent
    if $k > m+1$ do
        $H_{m+1}$ = Aprior_gen ($H_m$)
    for each $h_{m+1} \in H_{m+1}$ do
        conf = $\sigma(f_k) / \sigma(f_k - h_{m+1})$
        if conf $\geq$ minconf then
// Checking for minimum confidence and selecting / deleting the // rule based the value
        output the rule $(f_k - h_{m+1}) \rightarrow h_{m+1}$
        else
        delete $h_{m+1}$ from $H_{m+1}$
        end if
end for
call ap_gen_rules ($f_k$, $H_{m+1}$)
end if
```

RESULT ANALYSIS

A large super market tracks the sales data for each item. The stock keeping unit stores the information which gives the details of items purchased together. The transaction details are as in Table 1.

Table 1: The transactions

TID	Items
T_1	I_1, I_2, I_5
T_2	I_2, I_4
T_3	I_2, I_4
T_4	I_1, I_2, I_4
T_5	I_1, I_3
T_6	I_2, I_3
T_7	I_1, I_3
T_8	I_1, I_2, I_3, I_5
T_9	I_1, I_2, I_3

Table 2: The candidate 1-itemset

Items	Support
I_1	6
I_2	7
I_3	5
I_4	3

Table 3: Frequent 1-itemset

Items	Support	TIDs
I_1	6	$T_1, T_4, T_5, T_7, T_8, T_9$
I_2	7	$T_1, T_2, T_3, T_4, T_6, T_8, T_9$
I_3	5	T_5, T_6, T_7, T_8, T_9
I_4	3	T_2, T_3, T_4

Table 4: Frequent 2-itemset

Items	Support	Min	TIDs
I_1, I_2	4	I_1	$T_1, T_4, T_5, T_7, T_8, T_9$
I_1, I_3	4	I_3	T_5, T_6, T_7, T_8, T_9
I_1, I_4	1	I_4	T_2, T_3, T_4
I_2, I_3	3	I_3	T_5, T_6, T_7, T_8, T_9
I_2, I_4	3	I_4	T_2, T_3, T_4
I_3, I_4	0	I_4	T_2, T_3, T_4

We can define minimum support as 3. To generate all frequent 1-item set, we scan all transactions and we select transaction IDs that contain these items. We also eliminate the candidates that are in frequent/less than minimum support. The candidate 1-item set generated are as in Table 2.

Since the minimum support is 3, the item I_5 is removed from the candidate 1-item set. Next to generate frequent 1-item sets, scan all the transactions which contains items and their support count and the transaction IDs that contains these items. In this step, candidates that are infrequent or their support is less than minimum support threshold are eliminated. Thus the resulting frequent 1-item set is as in Table 3.

In order to generate candidate 2-item set from the above frequent 1-item set, we have to group the item with two items in each group and find the support for all possible combinations of items. In our example, we are using four items that are frequent are combined and candidate 2-item set is generated (Table 4).

Now determine the transactions where we can find the item set L1 instead of searching the whole database. For example, (I1, I2), the original Apriori algorithm will search all the 9 transactions. But in our improved Apriori algorithm, we split the item set into I1 and I2 and find the minimum support. So we search for item set (I1, I2) only in the transactions T1, T4, T5, T7 and T9.

Since the item sets (I1, I4) and (I3, I4) is having support level less than the minimum support value,

Table 5: Frequent 3-itemset

Items	Support	Min	TIDs
I_1, I_2, I_3	2	I_3	T_5, T_6, T_7 T_8, T_9
I_1, I_2, I_4	1	I_4	T_2, T_3, T_4
I_1, I_3, I_4	0	I_4	T_2, T_3, T_4
I_2, I_3, I_4	0	I_4	T_2, T_3, T_4

Table 6: Number of transactions

	Total scans	
Transaction set	Traditional apriori	Our improved apriori
T_1	8.325	4.995
T_2	13.500	8.100
T_3	18.450	11.070
T_4	35.400	21.240
T_5	44.516	26.867

Table 7: Comparison of transactions in time

	Time consumed (in sec)	
Transaction set	Traditional apriori	Our improved apriori
T_1	2.867	1.874
T_2	11.286	4.507
T_3	13.608	5.851
T_4	21.397	9.271
T_5	29.568	12.714

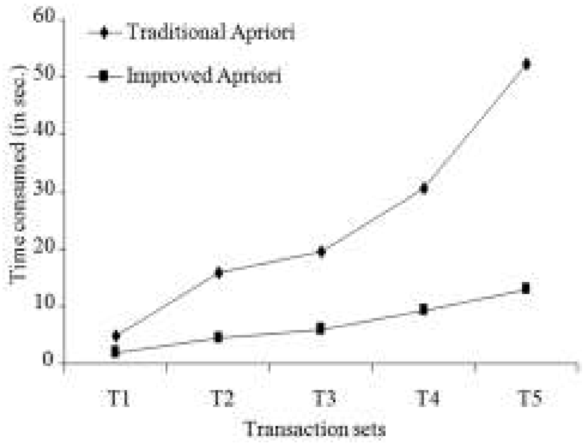

Fig. 1: Representation of transactions and time

these two frequent item sets are deleted for further processing.

The process is repeated for generating candidate 3-item set and from that frequent 3-item set are generated. The resultant frequent 3-item set are as in Table 5.

Since no more frequent item set generation is possible, we can stop the algorithm and start generating association rules.

Performance comparison: If we count the number of database scans done by traditional Apriori algorithms and improved Apriori algorithm, we observe that there is a huge reduction in database scans in the proposed method. Since frequent 1-item set generation is common for the traditional Apriori algorithm and improved Apriori algorithm, both the method needs the same number of transactions. From frequent 2-item set generation, our improved method outperforms the traditional algorithm.

Total number of transactions needed for the traditional Apriori algorithm is 143 and our improved Apriori algorithm consumes only 89 transactions to generated association rules. The efficiency of improved Apriori algorithm is 63% higher than the traditional method.

Next, comparison is done based on time consumed by the original Apriori algorithm and the improved version. For the above example, the traditional Apriori algorithm consumed 1.486 sec to generate frequent item sets whereas the improved versions consumed only 0.861 sec which is again less than 62% of time consumed by traditional version.

We continued our experiment with five different transaction sets namely T_1, T_2, T_3, T_4 and T_5 consisting of 555, 900, 1230, 2360 and 3000 transactions, respectively. The experiment was conducted on both traditional Apriori algorithm and improved Apriori algorithm.
The overall performance is given in the Table 6.

In the table shown that improved Apriori algorithm performs better than existing original Apriori algorithm. Next we estimate the time consumed to generate association rules by both the methods. The results are shown in the Table 7.

All the above data are plotted in a graph. The time consumed in improved Apriori in each group of transactions is less than the traditional Apriori and the difference increases more and more as the number of transactions in a set increases (Fig. 1).

CONCLUSION

In this study, generating frequent patterns using improved Apriori algorithm and support chaining is presented which reduces the number of scans as well as the time needed to generate efficient association rules. Improved Apriori algorithm uses large item set property, easy to implement, but it repeatedly scan the database. Apriori takes more time to scan the large Frequents patterns. Here the improvement is done in generating candidate item sets and this reduces the number of transactions to be scanned as well as the time. Whenever the item set increases, the gap between traditional and improved versions of Apriori algorithm increases in terms of performance and time consumption. This improved Apriori algorithm is faster than the existing algorithm.

REFERENCES

Agrawal, R., T. Imielinski and A. Swami, 1993. Mining association rules between sets of items in large databasesm. Proceeding of the ACM SIGMOD International Conference on Management of Data. ACM Press, New York, pp: 207-216.

Buehrer, G., S. Parthasarathy, S. Tatikonda, T. Kurc and J. Saltz, 2007. Toward terabyte pattern mining: An architecture-conscious solution. Proceeding of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp: 2-12.

El-Hajj, M. and O.R. Zaiane, 2006. Parallel leap: Large-scale maximal pattern mining in a distributed environment. Proceeding of the 12th International Conference on Parallel and Distributed Systems, pp: 128-136.

Fakhrahmad, S.M., M.H. Sadreddini and M.Z. Jahromi, 2007. Mining frequent itemsets in large data warehouses. Proceeding of the 8th International Conference on Intelligent Data Engineering and Automated Learning, pp: 517-526.

Fang, W., M. Lu, X. Xiao, B. He and Q. Luo, 2009. Frequent itemset mining on graphics processors. Proceeding of the 5th International Workshop on Data Management on New Hardware, pp: 97-105.

Kessl, R. and P. Tvrdk, 2007. Toward more parallel frequent itemset mining algorithms. Proceeding of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, pp: 317-328.

Li, H., Y. Wang, D. Zhang, M. Zhang and E.Y. Chang, 2008. Pfp: Parallel fp-growth for query recommendation. Proceeding of the ACM Conference on Recommender Systems, pp: 107-114.

Liu, L., E. Li, Y. Zhang and Z. Tang, 2007. Optimization of frequent itemset mining on multiple-core processor. Proceeding of the 33rd International Conference on Very Large Data Bases (VLDB, 2007), pp: 1275-1285.

Sandhu, P.S., D.S. Dhaliwal, S.N. Panda and A. Bisht, 2010. An improvement in apriori algorithm using profit and quantity. Proceeding of the 2nd International Conference on Computer and Network Technology (ICCNT, 2010), pp: 3-7, April 23-25.

Yanbin, Y. and C. Chia-Chu, 2006. A parallel apriori algorithm for frequent itemsets mining. Proceeding of the 4th International Conference on Software Engineering Research, Management and Applications, pp: 87-94, August 9-11.

Ye, Y. and C.C. Chiang, 2006. A parallel apriori algorithm for frequent itemsets mining. Proceeding of the 4th International Conference on Publication Software Engineering Research, Management and Applications, pp: 87-94.