Efficient Frequent Itemset Mining Mechanism Using Support Count

1Neelsh Kumar Kori, 2Ramratan Ahirwal, 3Dr. Yogendra Kumar Jain

1Department of C.S.E, Samrat Ashok Technological Institute, Vidisha, M.P., India
2Asst. Prof. Department of C.S.E, Samrat Ashok Technological Institute, Vidisha, M.P., India
3HOD of Computer Science & Engineering, Samrat Ashok Technological Institute, Vidisha, (M.P.), India

ABSTRACT

Data mining, which is the exploration of knowledge from the large set of data, generated as a result of the various data processing activities. Frequent Pattern Mining is a very important task in data mining. Frequent pattern mining has been a focused theme in data mining research for over two decades. In past literature has been dedicated to this research and tremendous progress has been made, ranging from efficient and scalable algorithms for frequent itemset mining in transaction databases to numerous research frontiers, such as sequential pattern mining, structured pattern mining, correlation mining, associative classification, and frequent pattern-based clustering, as well as their broad applications. In this paper we present a new technique which is combination of present maximal Apriori (improved Apriori) and Frequent Pattern-tree techniques (FP-tree) that guarantee the better performance than classical apriori algorithm. The aim of our proposed techniques are analyze the mining frequent itemsets and evaluate the performance of new techniques and compare with the existing classical Apriori and Frequent Pattern-tree algorithm with support count.

Keywords: Data Mining, Frequent Itemset, Apriori, Frequent Pattern-Tree, Support Count.

1. INTRODUCTION

Data mining techniques achieved wide recognition during the last years despite the fact that the concepts underpinning them were around for much longer. The need for getting information on customer’s habits was always a ‘hot’ issue. Moreover, the introduction and growth of the World Wide Web and, as a consequence of that, e-commerce made techniques that would give information about the new type of customer- the 'online customer' of great value to companies. This new category has some special characteristics associated with it such as that there is no room for personal contact and we need to infer conclusions about their likes and dislikes through the 'traces' they leave when navigating around the company's web pages. The latter should be combined with the latest marketing trend, widely known as direct marketing, which advocates personalized marketing.

Data mining requires identification of a problem, along with collection of data that can lead to better understanding and computer models to provide statistical or other means of analysis. This may be supported by visualization tools, that display data, or through fundamental statistical analysis, such as correlation analysis [1] and [3].

Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases, such as association rules, correlations, sequences, episodes, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems [2] and [4].

The aim of our proposed techniques are analyze the mining frequent itemsets and evaluate the performance of new techniques and compare with the existing classical Apriori and Frequent Pattern-tree algorithm with support count.

2. BACKGROUND TECHNIQUES

2.1 Association Rules:

The process of finding association rules is also known as market basket analysis. This is the case since it is widely applied in retail stores; however it might as well have applications in many other cases.

Association Rule Mining finds the set of all subsets of items/attributes that frequently occur in database records /transactions and extracts rules on how a subset of items influences the presence of another subset.

2.2 Association Rules and Frequent Itemsets

The market-basket problem assumes we have some large number of items, e.g., "bread," "milk." Customers will their market baskets with some subset of the items, and we get to know what items people buy together, even if we don't know who they are. Marketers use this information to position items, and control the way a typical customer traverses the store.

In addition to the marketing application, the same sort of question has the following uses:
1. Baskets = documents; items = words. Words appearing frequently together in documents may represent phrases or linked concepts. It can be used for intelligence gathering.

2. Baskets = sentences, items = documents. Two documents with many of the same sentences could represent plagiarism or mirror sites on the Web [4].

2.3 Goals for Market-Basket Mining

1. Association rules are statements of the form \((X_1; X_2; \ldots; X_n) \Rightarrow Y\), meaning that if we find all of \(X_1; X_2; \ldots; X_n\) in the market basket, then we have a good chance of finding \(Y\). The probability of finding \(Y\) for us to accept this rule is called the confidence of the rule. We normally would search only for rules that had confidence above a certain threshold. We may also ask that the confidence be significantly higher than it would be if items were placed at random into baskets.

2. Causality. Ideally, we would like to know that in an association rule the presence of \(X_1; \ldots; X_n\) actually 'causes' \(Y\) to be bought. However, 'causality' is an elusive concept. For market-basket data, the following test suggests what causality means. If we lower the price of diapers and raise the price of beer, we can lure diaper buyers, who are more likely to pick up beer while in the store, thus covering our losses on the diapers. That strategy works because 'diapers causes beer.' However, working it the other way round, running a sale on beer and raising the price of diapers, will not result in beer buyers buying diapers in any great numbers, and lose money.

3. Frequent itemsets. In many (but not all) situations, we only care about association rules or causalities involving sets of items that appear frequently in baskets. For example, we cannot run a good marketing strategy involving items that no one buys anyway. Thus, much data mining starts with the assumption that we only care about sets of items with high support; i.e., they appear together in many baskets. We then find association rules or causalities only involving a high-support set of items (i.e., \((X_1; X_2; \ldots; X_n; Y)\)) must appear in at least a certain percent of the baskets, called the support threshold [4] and [6] and [7].

2.4 Framework for Frequent Itemset Mining

Use the term frequent itemset for a set \(S\) that appears in at least fraction \(s\) of the baskets," where is some chosen constant, typically 0.01 or 1%. We assume data is too large to fit in main memory. Either it is stored in a RDB, say as a relation Baskets (BID; item) or as a file of records of the form (BID; item1; item2; \ldots; item n). When evaluating the running time of algorithms:

\(\Rightarrow\) Count the number of passes through the data. Since the principal cost is often the time it takes to read data from disk, the number of times we need to read each datum is often the best measure of running time of the algorithm. There is a key principle, called monotonicity or the a-priori trick that helps us find frequent itemsets:

\(\Rightarrow\) If a set of items \(S\) is frequent (i.e., appears in at least fraction \(s\) of the baskets), then every subset of \(S\) is also frequent.

2.5 Algorithms for mining association rules

The 'classical' associations rule problem deals with the generation of association rules by defining a minimum level of confidence and support that the generated rules should meet. This is the case since first of all we want the rule to contain items that are purchased often - if we know that whoever buys product A byes product B as well but product A occurs only in two out of ten million transactions then it is of low interest unless the profit margin is high. Furthermore, within the set of transactions that contain item A, we want to know how often they contain product B as well; this is the role of rule's confidence.

If we introduce the term frequent for an itemset \(X\) that meets the criterion that its support is greater than the minimum value set- for example, we might say that we want all items or set of items that were bought by more than 70% of our customers- then our problem is restricted to finding all frequent itemsets from the database. If we know these, then we can derive all association rules by following a simple strategy. This strategy involves the calculation for every frequent itemset \(X\) and very subset \(Y\) of it- which is neither the null neither set nor \(X\) - of the confidence level of all rules of the form \(X\Rightarrow Y\); the latter is essential so that we will comply with the part of the definition that demands the intersection set to be the null. As an example, if they consider an itemset that consists of pork steaks, coke and beer, the previous definition suggests that we should look for the confidence level of, say, the rule 'pork steak => coke and beer'. They, then, drop those that do not meet the minimum confidence level criterion. The problem, however, is that a small number of items is able of generating a large search space. This space, on the other hand, has a very interesting property that facilitates our work; there is a border that separates the frequent itemsets from the infrequent ones- thus, the problem is restricted on finding that border.

This is done through a mapping procedure between the set items and the set of natural numbers and the use of special classes. Each algorithm presented below will be characterized by how it looks for the border between frequent and infrequent itemsets and how it calculates the support value for each of them. The first can be done either by using the breadth-first search (BFS) or the depth-first search (DFS). In the breadth-first algorithm given a tree and a goal state they try initially all paths- ways of reaching our goal state - of length one, then all paths of length two and so on.
till they reach the goal state. In the depth-first search they try a path first till they get to a dead-end; then they return to the top and look for alternatives. As far as the calculation of support value is concerned, it can be done using either the number of the subset’s occurrences in the database or by using set intersections [5] and [8] and [10].

2.6 Breadth-first search and subset’s occurrences

In this category, they can identify one major algorithm, Apriori, and a number of others that, in essence, are variations of it. Apriori scans the database looking for all the candidate itemsets with k items in them. A hash-tree structure-, which is a tree structure for identifying the candidate subsets - is introduced in order to look for those sets throughout the database. In each transaction, they go down the tree and once they have reached its leaves it means that they have found some sets with common items that form the same prefix. They, then, look up in the database for these candidate sets and if they find them we increase the counter by one.

The second algorithm is Apriori TID; the difference between that algorithm and the previous one is that it encodes each transaction by the candidate sets it contains. In this category, they might also classify SETM – its advantage being that it is readily executable in SQL. Moreover, DIC is another algorithm that starts generating additional candidate itemsets once a set has reached the minimum support threshold [2] and [7] and [9].

2.7 Breadth-first search and set intersections

In this category, they can find the partition algorithm. In order for it to work, it requires the use of tid lists (i.e. the set of identifiers for transactions that contain a specific subset; in a super market case it might be the transaction ID) of all candidate frequent itemsets with cardinality, say, (k-1) to determine the tid lists of those with cardinality k. One problem is that these can be very big size and not fit in the main computer’s memory; this can be overcome, though, by dividing the database in smaller parts and adjusting the size so that all tid lists will fit into the main memory.

2.8 Depth-first search and subset’s occurrences

In this case they encounter problems with the size of the candidate itemsets, since they need to scan the database for each one of them; because of the cost this introduces it is very difficult to be put in practice. However, lately a new algorithm was introduced that overcomes this problem by aiming directly at some itemsets in the search space [3] and [6].

2.9 Depth-first search and sets intersections

The most widely used algorithm of this category is Eclat. The ‘innovation’ with Eclat is that once they take the intersection of two tid lists then the result is only of interest if its cardinality is at least equal to the value of the minimum support. In the case of the super market, for example, if they assume that tid lists consist of the transactions ID, then when they have to intersect the tid lists X and Y these are the transaction Ids of the transactions that contained the X and Y items – then they will accept the result – the transaction ID of the transactions containing both items – only if it reaches the minimum support. In other words, they want the number of transactions that contain both X and Y to occur at least as many times as they have set in the minimum support.

The first, and maybe, one of the biggest advantages of association rules is that the results produced are very easy to be understood. This is the case since they are only statements in plain English, which combine one or many items with other items from the database; for example a customer who buys chips buys Coke as well. This way, the end user can get a string insight of his business and decide what actions he should take.

Furthermore, it is a technique that can be used when we do not really know where to start from but they want to examine the trends in our business. It is probably the best technique to use when they do not have something particular in mind; when they do not want to predict the value of a specific attribute.

The computations that need to be performed are not very complex; that is another strong point of the method. In the classical association rule problem that was described above all they need to calculate is the support and the confidence of a number of rules.

On the other hand, it has drawbacks associated with it. The first is that despite the fact that the computations involved are simple, the search space can be very large with the presence of a relatively small number of items; in fact the relationship between the number of items and the search space is of exponential magnitude.

Moreover, it does not support ‘rare’ items; these are items that are not purchased very often. As a result of that, the support of any rule, which incorporates those items, will be very low resulting in it being thrown out of the class of frequent itemsets; this is the case since the support of the rule would be very low. Finally, the implementation of the algorithms tends to lead to a very big number of association rules. The most likely case is that they will be confusing and we will have to examine each one of them very carefully to determine if it is of any interest to us [4] and [9] and [10].
3. PROPOSED TECHNIQUES

A new algorithm based upon the improved Apriori and the Frequent Pattern-tree structure with support count is present. In a large transactional database like retailer database it is common that multiple items are selling or purchasing simultaneously therefore the database surly contains various transactions which contain same set of items. Thus by taking advantage of these transactions trying to find out the frequent itemsets and prune the database as early as possible without generating the candidate itemset and multiple database scan, results in efficiently usage of memory and improved computation.

This proposed algorithm is based upon the Apriori property i.e. all non empty subsets of the frequent itemsets are frequent. Algorithm has two procedures. In first procedure, find all those maximal transactions which are repeating in the database equal to or greater than min user defined support also known as maximal frequent itemset. Then get all nonempty subsets of those maximal frequent itemset as frequent according to Apriori property. Scan the database to find 1-itemset frequent elements. There may be many items found which are 1-itemset frequent but not include in maximal frequent transactions. Therefore prune the database by just considering only those transactions from the database which contain 1-itemset frequent elements, but not include in the maximal frequent itemsets. Now this pruned database is smaller than the actual database in the average cases and no item left in best case.

For the second procedure, pruned database is taken as input and scan the pruned database once find 1-itemset frequent and delete those items from transaction which are not 1-itemset frequent. Then construct the Frequent Pattern-tree only for pruned transactions. In this way it reduces the memory problem for Frequent Pattern-tree because the database is reduced in most of cases. In best case no need to build Frequent Pattern-tree because all elements are found in first procedure. In the worst case if there is no maximal frequent transaction exist, then only second procedure run and also computational performance is same as Frequent Pattern-tree. The key of this idea to prune the database after finding the maximal frequent itemsets and formation of Frequent Pattern-tree for a pruned database thus reduce memory problem in FP-tree and make the mining process fast. The more detail step as follows:

Procedure1:

**Input:** Database D, minimum support

**Step 1:** Take a 2-dimensional array; put the transaction into 2-dimensional array with their count of repetition.

**Step 2:** Arrange them in increasing order on the basis of the pattern length of each transaction.

**Step 3:** Find maximal transactions (k-itemset) from the array whose count is greater than or equal to the minimum support known as maximal frequent itemsets or transactions. If k-itemsets count is less than minimum support then look for k-itemsets and (k-1)-itemsets jointly for next (k-1) maximal itemsets and so on until no itemsets count found greater than minimum support. If no such transaction found then go to Procedure2.

**Step 4:** Once the maximal frequent transactions found, than according to Apriori property consider all its non empty subsets are frequent.

**Step 5:** There are itemsets remaining which are not included in maximal frequent itemset but they are frequent. Therefore find all frequent 1-itemset and prune the database just considering only those transactions which contain frequent 1-itemset element but not include in maximal frequent transaction.

**Output:** some or all frequent itemsets, Pruned database D1.

Procedure2:

**Input:** Pruned database D1, minimum support

**Step 1:** Find frequent 1-itemset from pruned database; delete all those items which are not 1-itemset frequent.

**Step 2:** Construct FP-tree for mine remaining frequent itemset by following the procedure of FP-tree algorithm

**Output:** Remaining frequent itemsets

**Example:**

Suppose table 1 is a database of retailer, D. There are 10 transactions. Suppose the minimum support is 2.

<table>
<thead>
<tr>
<th>Tid</th>
<th>List of Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T2</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T3</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T4</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T5</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T6</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T7</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T8</td>
<td>I1, I2, I3, I5</td>
</tr>
</tbody>
</table>
Procedure 1:

Input: Database D, Minimum support = 2

Step 1: After scanning a database put items in 2-dimensional array with the count of repetition.

<table>
<thead>
<tr>
<th>Table: Support Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 item set</td>
</tr>
<tr>
<td>{I1, I2, I4}</td>
</tr>
<tr>
<td>{I1, I2, I3}</td>
</tr>
<tr>
<td>{I1, I2, I4}</td>
</tr>
</tbody>
</table>

Step 2: Find maximal itemset (4-itemset). Check whether its count is greater or equal to specified support, its count is 2 in our case which is equal to given support therefore this transaction is considered as maximal frequent. If its count is less than support value then we scan k-1 and k-itemset in array for k-1 maximal itemset jointly and so on until finding all maximal frequent itemset from a array. i.e. 3-itemset and 4-itemset for checking 3-itemset maximal.

Step 3: According to Apriori property, subset of maximal frequent itemset is also considered as frequent i.e. Maximal frequent itemset: {I1, I2, I3, I5}. All subsets are frequent (Apriori Property). i.e. {I1, I2, and I3}, {I1, I2, I5}, {I2, I3, I5}, {I2, I3}, {I2, I5}, {I1, I2}, {I1, I3}, {I1, I5}, {I3, I5}, {I1}, {I2}, {I3}.

Step 4: Scan the database for finding the above mined support.


Procedure 2:

Input: Pruned database, minimum support = 2

Step 1: Find frequent 1-itemset from pruned database with support = 2. It is found I1 is not frequent therefore delete it.

Step 2: Construct FP-tree for remaining transaction in pruned database.

In this case I2 and I4 have the same frequency, therefore no need to rearrange in L order (descending order of their frequencies).

A new branch is created for each transaction. In this case single branch is created because of the same set of transactions.

Construct Conditional pattern base and FP – tree only for item I4.

Item → I4; Pattern base→ I2: 2; Frequent Pattern tree→ I2:2, Frequent Itemset→ I4, I2 :2

Output: remaining itemsets ({I4, I2})

Thus the remaining frequent itemsets which are not mined by only maximal frequent itemsets are mined by the FP and also in efficiently usage of memory because after pruning whole database is easily fit.
4. RESULTS ANALYSIS

The experimental results show the performance of our new technique the Apriori and Frequent pattern-tree algorithm with support count. The run time is the time to mine the frequent itemsets. The experimental result of time is shown in Figure 1 describes that the proposed scheme outperforms the FP-tree and the Apriori approach. It is clear from the graph (fig -2) that our proposed new algorithm performs well for the low support value with compare to Frequent Pattern tree and Apriori algorithm.

For the large dataset which contains the maximal frequent itemset in large amount shows better result with our new approach with respect to FP-tree and Apriori algorithm. In the dataset there are various transactions consider which occur repeatedly in the database and some transactions occur greater than the minimum support.

![Figure 2: Comparison Graph of Execution Time](image)

The itemset remains for mining frequent itemset are mined with the help of second procedure whose complexity equals to the FP-Growth algorithm but due to procedure 1 the overall complexity reduce and become efficient.

5. CONCLUSION

Mining frequent itemsets for the association rule mining from the large transactional database is a very crucial task. There are many approaches that have been discussed; nearly all of the previous studies were using Apriori approach and FP-Tree approach for extracting the frequent itemsets, which have scope for improvement. It is found that for a transactional database where many transaction items are repeated many times as a super set in that type of database maximal Apriori (improvement over classical Apriori) is best suited for mining frequent itemsets. The itemsets which are not included in maximal super set is treated by FP-tree for finding the remaining frequent itemsets. Thus this algorithm produces frequent itemsets completely. The aim of our proposed techniques are analyze the mining frequent itemsets and evaluate the performance of new techniques and compare with the existing classical Apriori and Frequent Pattern-tree algorithm with support count.

REFERENCES
