Hybrid Approach for Improving Efficiency of Apriori Algorithm on Frequent Itemset

Arwa Altameem and Mourad Ykhlef

Dept of Information Systems
Computer and Information Sciences
King Saud University, Riyadh, Saudi Arabia

Summary
Apriori algorithm is one of the most popular algorithms that is used to extract frequent itemsets from large database. It requires multiple scans over the database resulting in a large number of disk reads and placing a huge load on the system. This study proposed an improved version of Apriori Algorithm to overcome the deficiency of the basic one. The improved version is more efficient which takes less time and hence reflects in high efficiency.

Key words: Improved Apriori, Data Mining, Frequent Itemset, Hybrid Apriori.

1. Introduction
The concept of data mining is considered critical in obtaining specified information from a given large database. Data mining is sometimes referred to as knowledge discovery in the database. Implicit knowledge within the databases can afford important patterns like association rules that are helpful in decision support making, medical diagnosis and many other applications. The process of retrieving information from the database involves special procedures that analyze the abstract patterns of the data before making the final decisions (Jeon Y. et al., 2017). It is important to note that pattern evaluation module works with data mining modules to generate the interested patterns. The patterns are then compared with the limits defined by the minimum support to decide the number of entries relevant for extraction (Hadzic, Tan & Dillon, 2011). Association mining is one of the most important functionalities in data mining. Association rule mining is a task of finding interesting association or correlation relationships among large databases. It is considered effective in detecting unknown relationships and thereby produce results (Shi, 2011). There are two distinct phases that are used to mark the association rules, detection of the frequent item sets and generation of the association rules. Apriori is one of the most famous and basic methods in mining association rules and finding the frequent item sets. It will be introduced in the next section.

2. Basic Apriori Algorithm
The history of Apriori algorithm can be traced back to 1994 when it was first proposed by Agarwal and Srikant. The algorithm was first proposed to operate on databases that constituted customer transactions. Later it was developed to assess the association patterns in the items contained in the database (Agrawal et al., 1993). The algorithm makes use of a bottom-up approach which investigates different regular subsets that appear in the database. Every item in the database is evaluated at a time. At times, different groups of items can be investigated against data. The algorithm starts by first identifying the individual items that are frequently sought in the database. It then proceeds and extends to large and larger item sets along the support set by the user. The process of scanning through the database is completed when there are no further successful associations and extensions that are realized from the database. The algorithms work best on items that appear frequently and sufficiently in the database (Tank, 2012). The items identified are then used to determine the association rules which are used to establish trends in the database. The pseudo code of basic Apriori algorithm is illustrated in Figure 1.

Fig. 1 Basic Apriori Algorithm

C_k: Candidate itemset of size k
L_k : frequent itemset of size k
L_1 := {frequent 1-itemsets};
k := 2; //k represents the pass number.
while(L_k-1 != ø)
    C_k := new candidates of size k generated from L_k-1
    for all transactions t ∈ D
        increment count of all candidates in C_k that are contained in t
    L_k := all candidates in C_k with minimum support
    k := k+1
Return U_k L_k as the discovered frequent itemsets
2.1 Example

The Apriori algorithm of searching frequent itemsets is explained with the database of Table 1.

<table>
<thead>
<tr>
<th>Items</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACD</td>
<td>10,50</td>
</tr>
<tr>
<td>BCE</td>
<td>20,30,40</td>
</tr>
<tr>
<td>ABCE</td>
<td>10,20,30</td>
</tr>
<tr>
<td>BE</td>
<td>20,20,40</td>
</tr>
</tbody>
</table>

The algorithm assumes the minimum support threshold to be “2”. First, the frequent 1-itemsets L1 is found. In the algorithm each Lk generated must scan database one time, Ck is generated from Lk-1. L2 is generated from L1 and so on, until no more frequent k-itemsets can be found and then algorithm ceases. An example is illustrated in Figure. 2.

3. Problem Description

Even though effective algorithms like Apriori are reported for Association mining, they still inherit the issue of scanning the whole database many times and most of time has been consumed in accessing the database. The Apriori algorithm makes many searches in database to find frequent itemsets. Moreover, basic Apriori Algorithm generates a large set of candidate sets if database is large. Hence, it still suffers from the cost of producing a huge number of candidate sets and scanning the database continually to produce a large set of candidate itemsets. In another words, it is required to produce 2100 candidate itemsets to obtain frequent itemsets of size 100 (Figure. 3). Another drawback is the tedious work of support counting for each itemset. These limitations and other related issues are the motives of this research. A number of conducted surveys in many researches indicates that more consideration is required to address the issues related to Apriori (Yuan, X., 2017; Suneetha and Krishnamoorti, 2011). There is a need to diminish the number of database scan, and also to reduce memory space since Apriori results in a large number of disk reads (Yu, S., & Zhou, Y., 2016).

4. Approaches of Improving Apriori

Various improved algorithms have been proposed by many researchers to overcome the weaknesses of Apriori algorithm:

- Hash-based: This approach decreases the size of candidate set through filtering any k-item set whose corresponding hashing count below the threshold. It shows improvement in execution time and utilization of space.
- Bhardwaj et. Al. (2015) used a data structure that directly represents a hash table. This algorithm proposes overcoming some of the weaknesses of the Apriori algorithm by reducing the number of candidate k-itemsets.
- Transaction reduction: A transaction that does not contain any frequent k-item set is useless in subsequent scans. Mohammed Al- Maolegi and Bassam Arkok (2014) indicated the limitation of the original Apriori algorithm of wasting time for scanning the whole database searching on the frequent item-sets, and presented an improvement on Apriori by reducing that wasted time depending on scanning only some transactions.
Fig. 3 The huge number of possible itemsets (given d items, there is 2d possible itemsets)

- Partitioning: A partitioning can be used to distribute the entire dataset into smaller subsets. The dataset is divided into n non-overlapping partitions. Each partition is mined separately. Any item set that is potentially frequent in DB must be frequent in at least one of the partitions of DB. As Albshiri (2013) made use of a vertical data partitioning or a horizontal data segmentation technique to distribute the input data amongst a number of agents.

- Sampling: Sampling is mining a random sampled small subset from the entire dataset. Since there is no guarantee that we can find all the frequent itemsets, a lower support threshold should be used (Aggarwal and Sindhu, 2015).

However each one of these approaches has its own drawbacks like high computational requirement, high memory utilization, data deviation and less accuracy. Hence, to overcome the negatives and to combine the positives, a hybrid approach is being introduced in the next section.

5. Improved Apriori Approach

This study proposes an improved Apriori algorithm with a hybrid approach. This algorithm shrinks original database by integrating user’s domain knowledge, eliminating any non-related items and then mining k-frequent itemsets using the new database. Therefore, it is necessary to delete the useless transactions in the database in order to reduce the scale of database and reduce itemsets generated from Ck by the join procedure.

Furthermore, the proposed algorithm reduces time consuming for candidates itemset generation. This accomplished by:

- Generating Ck without scanning the whole DB \( \Rightarrow \) Ck will be generated fast.
- Add attribute called (Tindex) to store positions of items.
- Common transactions= Support count.

The pseudo code of improved algorithm is illustrated in Figure. 4. The approach reduces number of transactions when focusing on preferable items and thus, reduces number of comparisons. Further, there is no need to scan database every time, the approach demands only one scan to generate first candidate set. It also facilitates support counting of itemset as it is not necessitate matching every candidate against every transaction to count support.

\( D' = \) subset of D containing preferences specified by the user.

\( C_k: \) Candidate itemset of size k

\( [Tx]= \) transactions that contain \( c \in C_k \) \( // \) new attribute to hold transactions' numbers

\( C'_k: \) modified candidate itemset of size k after adding Tx attribute

\( L_k: \) frequent itemset of size k

\( L_1: \) [frequent 1-itemsets];

\( k:=2; \)

while ( \( L_{k-1} \)= \( \emptyset \) )

\( C'_k: = \) new candidates of size k generated from \( L_{k-1} \)

For each \( c \in C'_k \)

\( [Tx]= \) common transactions between items \( i1 \wedge i2 \mid i1 \neq i2 \)

\( c.cound= \) num of elements in \( [Tx] \) //counting

\( \text{min}\_\text{support} \)

\( L_k: =\{c\in C_k' | c.\text{count} \geq \text{minsup}\} \)

\( k := k+1 \)

Return \( U_k, L_k \) as the discovered frequent itemsets

Fig. 4 Improved Apriori Algorithm

5.1 Stepwise Description of the Pseudocode

1- Scan the database to get C1, and store for each itemset \( c \subseteq C1 \), the transaction numbers in Tx.

2- Make sure the proper minsupport to get frequent itemset \( L_1 \).

3- Generate Ck \( (k=2) \) though joining two frequent itemsets that belong to \( L_{k-1} \). The mth item in the first \( L_{k-1} \) should join with the \( m+1 \)th item of the second \( L_{k-1} \). To count minsupport for each \( c \), get the number of common \( tx \subseteq Tx \) between \( i1 \wedge i2 \mid i1 \neq i2 =c \) (parents of \( c \)).
4. Generate \( L_k \) by adding items of \( C_k \) to \( L_k \) if the support of the item is greater than minsupport.

5. If \( L_{k+1} = \emptyset \), the algorithm terminates. Otherwise, \( k = k+1 \), continue to the second step in circle until \( L_{k+1} = \emptyset \).

5.2 Example

The performance of the new algorithm is compared with that of Apriori algorithm with the help of an example (Figure 5). The example used a new database \( D' \) (Table 2) after eliminating useless items according to user preferences (user is interested in B). The minimum support threshold is the same as previous which is “2”.

Table 2: Database \( D' \)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCE</td>
<td>30</td>
</tr>
<tr>
<td>ABCE</td>
<td>20</td>
</tr>
<tr>
<td>BE</td>
<td>20</td>
</tr>
</tbody>
</table>

As shown below, figure 5 illustrates step-wise working of the proposed approach.

6. Theoretical Comparison of the Performance

The theoretical estimation of the performance is presented in this section. The total run time for basic Apriori algorithm is defined as in equation (1) (Suneetha and Krishnamoorti, 2011):

\[
\sum_{i=1}^{n} \left( t_s \times m_k + t_c + l_{k+1} \times k+1 / 2 \times t_s \times m_k / A \right)
\]

where, \( t_s \) be the time cost of a single scan of the database, \( t_c \) be the time cost of generating \( C_{k+1} \) from \( L_{k} \), \( m_k \) is set to be the amount of item sets in \( C_k \), the variable \( l_{k+1} \) is set to be the amount of item sets in \( C_{k+1} \) and the variable \( n \) denotes the amount of the records in the database and \( A \) denotes the dimension of the data.

The total estimated run time for improved version of Apriori algorithm is indicated in equation (2):

\[
t_s \times m'_k + \sum_{i=2}^{K+1} \left( t_c + 2 \times t_s \times l_{k+1} \right)
\]

7. Experiment

For the purpose of performance evaluation, both improved Apriori algorithm and basic Apriori have been run on the same platform under same conditions --windows RAM etc. The programming language used for the implementation algorithms is java (jdk 1.5.0).

The experimental runs have been conducted with two different sized datasets. It has been found that the proposed algorithm always takes less time than basic Apriori and the interesting information can be mined in a shorter time.

The time comparison of both the algorithms is shown in Figure 6.
we proposed an improved algorithm. Our proposal helps to decrease the times of scanning database, optimize the process that generates candidate itemsets and facilitate support counting. Theoretical and practical experiments results show that the proposed algorithm outperforms the Apriori algorithm in computational time.

Acknowledgements

This work was supported by Deanship of Scientific Research and Research Center of College of Computer and Information Sciences, King Saud University. The authors are grateful for this support.

References