A Performance Study of Apriori Implementation in Data Mining using Vectors

Prasanna M. Kothawale
Department of Computer Science, D.M’S College and Research Centre, Assagao, Bardez-Goa, 403 507.

Abstract: The improvement of performance of the Apriori algorithm by considering vertical representation of database using vector is undertaken. Vertical format supports for fast frequency counting on transaction ids, reduction in the I/O operation and memory requirement. We have presented a complete new vertical format representation for storing item set, pruning and scanning the database for Apriori algorithm. Experimental comparison on both the dense and sparse datasets show better performance in the adopted approach.

1. Introduction

Data mining [1] refers to extracting or “mining” knowledge from large amounts of data. It is the discovery of non-trivial, implicit, previously unknown, and potentially useful information or patterns embedded in databases [2]. Traditionally, business analysts have performed the task of extracting useful information from recorded data, but the increasing volume of data in modern business and science calls for computer-based approaches. Data mining identifies trends within data that go beyond simple analysis and through the use of sophisticated algorithms, non-statistician users have the opportunity to identify key attributes of business processes and target opportunities. It is normally used by large corporations employing Business Intelligence integrated with an ERP system to help make managerial decisions based on the patterns and forecasts generated from the data collected. The data mining research popularly involves development of new efficient mining algorithms for given frequent pattern mining problems. Most of the previous algorithms used today typically employ sophisticated in-memory data structures, where the data is stored into and retrieved from flat files. In present work, we have presented a new vertical data representation called single field representation of transaction ids and item sets using container class “vector”. Vectors are a kind of sequence containers, as such, their elements are ordered following a strict linear sequence.

Vector containers are implemented as dynamic arrays. Just as regular arrays, vector containers have their elements stored in contiguous storage locations, which means that their elements can be accessed not only using iterations but also using offsets on regular pointers to elements. In this approach only the frequent item sets are stored for each transaction. This work is to improve the performance of the Apriori algorithm with vertical representation of database using vector.

2. Frequent Item Set and Problem Formulation

Mining frequent patterns [3] or item sets is a fundamental and essential problem in many data mining applications. The problem is formulated by finding all the frequent item sets from a large database of item transactions. The frequent item set is one that occurs in at least a user-specified percentage of transactions of the database. The objective of frequent item set mining is to find all the item-sets that satisfy the user input minimum support threshold. There have been many different algorithms proposed for frequent item set mining, some of which are Apriori, Eclat and FP-growth.

The frequent item set mining problem can be stated as follows – Problem: Given a set of n transactions \( T_1, T_2, ..., T_n \) in a dataset D, each transaction \( T_i \) consisting of a variable number of items from the item set \( I = \{1, 2, ..., m\} \) and a user input \( min_sup \) representing the minimum support, the problem is to find the set of frequent item sets in the dataset D. An item set is frequent if its support in the dataset D is more than the user input \( min_sup \). The support of an item set \( I_k = \{1, 2, ..., K\} \), is defined as the ratio \( (p/n) \) of the number of transactions \( p \) in the dataset D containing items from \( I_k \) to the total number of transactions \( n \) in the dataset.

3. Frequent Item Set Mining Approaches

Apriori Algorithm: The first algorithm to generate all frequent patterns was the AIS algorithm proposed by Agrawal et al.[4]. To improve the performance, an anti-monotonic property of the support of item sets, called the Apriori heuristic, was identified by Agrawal et al. [4]. Eclat Algorithm: The algorithm Eclat [5] combines Depth First Search
(DFS) with tid list intersections based approach on vertical data layouts. Apriori determines the support of item sets either by testing for each candidate item set which transactions it is contained in, or by traversing for a transaction all subsets of the currently processed size and incrementing the corresponding item set counters. Eclat, on the other hand, traverses the search space in depth first order. That is, it extends an item set prefix until it reaches the boundary between frequent and infrequent item sets and then backtracks to work on the next prefix. Eclat determines the support of item sets by constructing the list of identifiers of transactions that contain the item set. Recently, Zaki [5] proposed a new approach to efficiently compute the support of an item set using the vertical database layout.

**FP-growth Algorithm:** Lately, an FP-tree based frequent pattern mining method called FP-growth, developed by Han et al. [6], achieves high efficiency, compared with Apriori-like approach. The FP-growth method adopts the divide-and-conquer strategy, uses only two full I/O scans of the database, and avoids iterative candidate generation. The frequent pattern mining consists of two steps: 1. Construct a compact data structure, frequent pattern tree (FP-tree), which can store more information in less space and 2. Develop an FP-tree based pattern growth (FP-growth) method to uncover all frequent patterns recursively.

### 4. Frequent Item Set Mining Using Vectors

The algorithm designed using vectors for the frequent item set mining problem stated in the section 2 is as follows:

**Algorithm:** Find a frequent pattern table (PAT) using an iterative level wise approach based on candidate generation.

**Input:** Transaction table T in Dataset file D, a minimum support threshold, min_support.

**Output:** L, A frequent pattern table PAT in dataset file (D).

**Method:**

1. Read the transaction table T and assign to item-list vector Generate frequent item (FI) and Non frequent item (NFI).
2. \( C_1 = \text{candidate_itemset\_one}(\text{min\_support}, L_1, \text{Sizeof}(L_1)) \) // find frequent 1 itemsets table (FIT)
3. \( \text{for} (k=2; L_{k-1} \neq 0; k++) \)
4. \( C_k = \text{generate\_candidate\_itemset} (\text{min\_support}, L_{k-1}, \text{sizeof}(L_{k-1})) \)
5. \( \text{for every transaction } t \in D \) // scan D for counts
6. \( C_t = \text{subset}(C_k, t) // \text{get the subsets of } t \text{ that are frequent candidate} 
7. \( \text{for each candidate } c \in C_t \)
8. \( \text{count=} \text{support\_count}(c) \)
9. \( L_k = \{c \in C_k | \text{count} > = \text{min\_support} \}
10. \}
11. return \( L_k \);

**procedure generate\_candidate\_itemset**

1. \( \text{min\_support : Minimum support} \), \( L_{k-1} : \text{frequent (k-1) itemsets, Sizeof}(L_{k-1}) : \text{Size of } L_{k-1} \)
2. \( \text{for each itemset } l_1 \in L_{k-1} \{
3. \text{for each itemset } l_2 \in L_{k-1} \{
4. c = l_1 \cup l_2 ; // \text{join } l_1 \text{ and } l_2 \text{ to generate candidates}
5. \text{if } \text{pruning} (L_{k-1}) \text{ then}
6. \text{delete } c; // \text{remove infrequent candidate whose subset is infrequent}
7. \text{else add } c \text{ to } C_k;
8. \}
9. \}
10. return \( C_k \);

**procedure pruning** (L_{k-1} : frequent (k-1) itemset): 1. \( \text{for each (k-1) subset } s \text{ of } c \)
2. \( \text{if } s \notin L_{k-1} \text{ then}
3. \text{return } l_1;
4. \text{return } 0;

**procedure support\_count** (c: candidates item set)
\( // \text{scan } D \text{ for counts}
1. \text{for each candidate } c \in C_t
2. c.count++;
3. \text{return } count;

### 5. Hardware and Software Platform

**Hardware:** Computer System with Intel Core 2 Duo 1.8 GHz Processor, 2 GB RAM, 80GB hard disk, Original Intel Motherboard 915 chip, DVD writer, 2S/2 USB Port, 15 inch colour Monitor, Key board and Optical Mouse. **Software:** Windows XP, Developer C++ Compiler of Borland 4.9.9.2 and MS Office XP.

### 6. Experimental Results


**Experiment 1:** (Input): Accident dataset with 200, 400, 600, 800, 1000 transactions and Min_Support of 0.7, 0.8 and 0.9:
Observations: From figure 1, it is observed that as the Min_Support decreases, the time taken to generate frequent item sets increases for both the approaches for a given number of transactions. However, vertical approach takes less time compared to horizontal approach. It is also seen that vertical approach takes less time till 800 transactions for Min_Support=0.7.

Experiment 2: (Input): Retail dataset with 1000, 2000, 5000 transactions and Min_Support 0.02, 0.025, 0.03, 0.035 and 0.04.

Observations: Figure 2 show that vertical approach algorithm takes less time to generate frequent item sets for higher Min_Support as compared to the horizontal approach algorithm for given number of transactions. However it is found that as transaction number increases beyond 5000, the vertical approach takes less time for Min_Support 0.03 onwards.

Experiment 3: (Input): Mushroom dataset with 200, 400, 600, 800, 1000 transactions and Min_Support 0.7, 0.8 and 0.8.

Observations: Figure 2 show that vertical approach algorithm takes less time to generate frequent item
sets for higher Min_Support as compared to the horizontal algorithm. It is also observed that as Min_Support decreases the time taken by both approaches increase for a given number of transactions. However it is seen that vertical approach takes less time till 800 transactions for Min_Support 0.7.

![Graph](image1.png)

**Figure 3.** Graph of execution time versus number of transactions for various Mini-Support for Mushroom dataset.

7. **Conclusion**

Apriori algorithm generates lot of candidate sets and every candidate set needs to apply pruning criteria followed by a database scan and in turn more time is required to calculate support count for every item set generated. Performance of Apriori algorithm is enhanced by using innovative vertical technique to scan database, generate item set and pruning. The implemented technique of vertical representations using vector for Apriori algorithm showed less execution time depending on selection of threshold compared to Ferenc Bodon’s horizontal approach. The advantages of vertical representation includes fast frequency counting on transaction ids, pruning of irrelevant data, improved performance over horizontal approach and reduces the number of I/O operations in datasets which has long transactions. Future work can lead to use this innovative technique to do the parallel processing for scanning database by partitioning main single column table into different tables, which can concurrently do the processing and then combine their result to get the support count. Also, this technique can be used for prediction purpose in online shopping applications.

8. **Acknowledgements**

I wish to acknowledge, Dr. Jyoti Pawar, Associate Professor, Department of Computer Science, Goa University, Goa and Mr. Sanjay B. Karapurkar, Associate Professor, Department of Computer Science, DM’S College and Research, Assagao Bardez, Goa, for their valuable guidance and suggestions.

9. **References**


