International Journal of Mathematics Research.
ISSN 0976-5840 Volume 12, Number 1 (2020), pp. 23-32

https://dx.doi.org/10.37624/IJMR/12.1.2020.23-32

Extraction of itemsets frequents

Y.Fakir*, C. Esaili, M.Fakir, R. Elayachi

Universiy Sultane Moulay Slimane, Faculty of Sciences and Tchnics, Beni mellal, morocco.

Abstract

Frequent model extraction is the most important step in association rules. The time required for generating frequent itemsets plays an important role. This paper provides a comparative study of algorithms Eclat, Apriori and FP-Growth. The performance of these algorithms is compared according to the efficiency of the time and memory usage

Keywords: Apriori, Fp-growth, Eclat, itemsets, association rule.

I. INTRODUCTION

In the field of data mining, the search for association rules is a popular method studied in a thorough way whose aim is to discover relations of interest to the statistician between two or more variables stored in very large databases.

The association rules, being an unsupervised learning method, make it possible to discover, from a set of transactions, a set of rules that expresses a possibility of association between different items. A transaction is a succession of items expressed in a given order; similarly, the transaction set contains transactions of different lengths.

An association rule is an implication expression of the form X =>Y, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in a transactions that contain X. The formal definitions of these metrics are :

$$\text{Support}(X \Rightarrow Y) = \text{support}\ (X \cup Y)$$

$$\text{Confidence}(X \Rightarrow Y) = sup\ (X \cup Y)/support\ (X)$$

Intuitively, a set of items that appears in many baskets is said to be “frequent.” To be formal, we assume there is a number S, called the support threshold. If I is a set of items,

the support for I is the number of baskets for which I is a subset. We say I is frequent if its support is equal to S or more.

In this paper, a comparison between the algorithms Apriori, Fp-growth, and Eclat, is done.

The remainder of this paper is organized as follows:

Section 2 deals with the Eclat algorithm while section 3 describes the Apriori algorithm. Section 4 present the FP-Growth algorithm. In section 5 we present the results of experiment carried out on the database. Section 6 conclude the paper.

II. ECLAT ALGORITHM

The Eclat algorithm is used to perform itemset mining. let us find frequent patterns in data like if a consumer buys milk, he also buys bread. This type of pattern is called association rules and is used in many application domains[1,6].

The basic idea for the Eclat algorithm is use tidset intersections to compute the support of a candidate itemset avoiding the generation of subsets that does not exist in the prefix tree. The algorithm Eclat is as follows :

Algorithm 1 : Eclat

Input: Fk = {I1, I2, ..., In} // cluster of frequent k-itemsets.

Output: Frequent l-itemsets.

```
Bottom-Up (Fk) {
for all Ii € Fk
   Fk +1 = ∅;
   for all Ij € Fk , i < j
    N = Ii ∩ Ij;
    if N .supı>= min_sup then
     Fk + 1 = Fk +1 U N;
    end
  end
 end
 if Fk +1 ≠ ∅ then
Bottom-Up (Fk +1);
 end
}
```

In this algorithm, numbers of items are stored in Fk as a input. Output is frequent itemsets which are frequently occurred. Searching of elements starts from bottom to top. First take Fk +1 as empty datasbase. In next step find support of individual items. Compare support of all items with minimum support. And that put all those items in Fk +1. Fk +1 contains all frequent items. Again check that Fk +1 is empty or not. If it is not empty then bottom up approach will apply on Fk + 1.

III. APRIORI ALGORITHM

Apriori principle:

- If an itemset is frequent, then all of its subsets must also be frequent [3,4]

Apriori principle holds due to the following property of the support measure :

- Support of an itemset never exceeds the support of its subsets
- This is known as the anti-monotone property of support

The basic steps to mine the frequent elements are as follows:

Generate and test: In this first find the 1-itemset frequent elements L1 by scanning the database and removing all those elements from C which cannot satisfy the minimum support criteria.

Join step: To attain the next level elements Ck join the previous frequent elements by self join i.e. Lk-1*Lk-1 known as Cartesian product of Lk-1.

i.e. This step generates new candidate k-itemsets based on joining Lk-1 with itself which is found in the previous iteration. Let Ck denote candidate k-itemset and Lk be the frequent k-itemset.

Prune step: This step eliminates some of the candidate k-itemsets using the Apriori property. A scan of the database to determine the count of each candidate in Ck would result in the determination of Lk (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to Lk). Step 2 and 3 is repeated until no new candidate set is generated.

Apriori algorithm, in spite of being simple, has some limitation. They are:

- It is costly to handle a huge number of candidate sets.
- It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.
- In order to overcome the drawback inherited in Apriori, an efficient FP-tree based mining method, FP-growth, which contains two phases, where the first phase constructs an FP tree, and the second phase recursively Researches the FP tree and outputs all frequent patterns.

IV. FP-GROWTH ALGORITHM

FP-Growth: allows frequent itemset discovery without candidate itemset generation. Two step approach:

Step 1: Build a compact data structure called the FP-tree built using 2 passes over the data-set.

Step 2: Extracts frequent item-sets directly from the FP-tree.

FP-tree: is a compact data structure that stores important, crucial and quantitative information about frequent patterns.

The main components of FP tree are:

- It consists of one root labeled as "null", a set of item prefix sub-trees as the children of the root, and a frequent-item header table.
- Each node in the item prefix sub-tree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP- tree carrying the same item-name.
- Each entry in the frequent-item header table consists of two fields, (1) item-name and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

The tree construction algorithm is listed in Algorithm 2 [5].

Algorithm 2: FP-tree_construction

Input: A transaction database DB and a minsupcount ξ.

Output: The frequent pattern tree F

(1) 1. Scan the DB to get the list L of frequent items, and sort it in support descending order.

(2) Create a FP-tree F by:

(3) Create the header table, and set all the headof-node-links to null.

(4) Create the root node T of the tree having the item-name of null.

(5) Set the parent-link and node-link of T to null.

(6) 2. Scan the DB again

(7) For each transaction Tran in DB do

(8) Get the list of frequent items.

(9) Sort it according to the order L.

(10) Let this list be [p|P], where p is the first item and P is the remaining items.

(11) Call insert_tree([p|P], T).

where the insert_tree(.) procedure is defined in Algorithm 3.

Algorithm 3: insert_tree

Input: the ordered list [p|P] of frequent items, and a node T of a FP-tree.

Output: the updated FP-tree.

(1) if T has a child node N such that the item name of N and p is the same then

(2) Increase the count of N by 1

(3) else

(4) Create a new node N.

(5) Set the item_name of N to p.item_name.

(6) Set the count of N to 1.

(7) Link the parent-link of N to T.

(8) Set the node-link of N to null.

(9) if the head-of-node-links of the item h in the header table having the same name as p is null then

(10) Set head-of-node-links of h to p;

(11) else

(12) Traverse through the head-of-node-links of h to the end of the list, and link the node-link of the end-node to p.

(13) if P is not empty then

(14) Let P=[p1|P1]

(15) Call insert_tree([p1|P1], N)

Advantages of an FP-Tree structure:

The most significant advantage of the FP-tree is that the algorithm scans the tree only twice. Apart from this major advantage, the others include:

- Completeness:

  The FP-tree contains all the information related to mining frequent patterns (given the min_support threshold)

- Compactness:
  - The size of the tree is bounded by the occurrences of frequent items
  - The height of the tree is bounded by the maximum number of items in a transaction

Three major steps performed are starting the processing from the end of list L:

- Construct conditional pattern base for each item in the header table
- Construct conditional FP-tree from each conditional pattern base
- Recursively mine conditional FP-trees and grow frequent patterns obtained so far. If the conditional FP-tree contains a single path, simply enumerate all the patterns.

The algorithm to extract frequent patterns from a FP-tree is given in Algorithm 4.

Algorithm 4: FP_growth

Input: FP-tree constructed based on Algorithm 1, using DB and a minsupcount ξ, and a pattern prefix α

Output: The complete set of frequent patterns. Procedure FP-growth (Tree, α)

(1) if Tree contains a single path P then

(2) for each combination (denoted as β) of the nodes in the path P do

(3) Generate pattern $\beta \cup \alpha$ with support =minimum support of nodes in β;

(4) else for each ai in the header of Tree do

(5) Generate pattern β = ai $\cup \alpha$ with support = ai .support;

(6) Construct β 's conditional pattern base and then β 's conditional FP-tree Treeβ with respect to minsupcount ξ ;

(7) If Tree$\beta \neq \emptyset$ then call FP-growth (Treeβ, β) To extract patterns from a FP-tree F, we call FP-growth(F, null).

V. EXPIRIMENTAL RESULTS

For the experimental study, the same dataset will be used for tree algorithms in order to compare the efficiency and rapidity of the algorithms.

Input:

Transaction id	**Items**
t1	{1, 3, 4}
t2	{2, 3, 5}
t3	{1, 2, 3, 5}
t4	{2, 5}
t5	{1, 2, 3, 5}

Output:

Apriori :

------- FREQUENT ITEMSETS -------

L0

L1

pattern 0: 1 support : 3

pattern 1: 2 support : 4

pattern 2: 3 support : 4

pattern 3: 5 support : 4

L2

pattern 4: 1 2 support : 2

pattern 5: 1 3 support : 3

pattern 6: 1 5 support : 2

pattern 7: 2 3 support : 3

pattern 8: 2 5 support : 4

pattern 9: 3 5 support : 3

L3

pattern 10: 1 2 3 support : 2

pattern 11: 1 2 5 support : 2

pattern 12: 1 3 5 support : 2

pattern 13: 2 3 5 support : 3

L4

pattern 14: 1 2 3 5 support : 2

The **output** is defined as follows; each line represents a frequent itemset. On each line, the items of the itemset are listed after the pattern number. Each item is represented by an integer and followed by a single space. After, all the items, the keyword "support:" appears, which is followed by an integer indicating the support of the item-set.

========== APRIORI - STATS ========

Candidates count: 15

The algorithm stopped at size 5

Frequent itemsets count: 15

Maximum memory usage: 1.282684326171875 mb

Total time ~ 17 ms

==========

The statistics above contain the number of generated patterns, the pattern size at which the algorithm sttopped wich in this case is five, then the memory used for the algorithm's variables and finally, the runtime.

FP-Growth :

------- FREQUENT ITEMSETS -------

L0

L1

pattern 0: 5 support : 5

pattern 1: 3 support : 5

pattern 2: 2 support : 5

pattern 3: 1 support : 5

L2

pattern 4: 1 5 support : 3

pattern 5: 3 5 support : 4

pattern 6: 2 5 support : 5

pattern 7: 2 3 support : 4

pattern 8: 1 3 support : 4

pattern 9: 1 2 support : 3

L3

pattern 10: 1 3 5 support : 3

pattern 11: 2 3 5 support : 4

pattern 12: 1 2 5 support : 3

pattern 13: 1 2 3 support : 3

L4

pattern 14: 1 2 3 5 support : 3

==========FP-GROWTH- STATS ==========

Transactions count from database : 7

Max memory usage: 1.2811813354492188 mb

Frequent itemsets count : 15

Total time ~ 12 ms

========== ==========

Eclat :

------- FREQUENT ITEMSETS -------

1 #SUP: 3

2 #SUP: 4

3 #SUP: 4

5 #SUP: 4

1 2 #SUP: 2

1 2 3 #SUP: 2

1 2 5 #SUP: 2

1 2 3 5 #SUP: 2

1 3 #SUP: 3

1 3 5 #SUP: 2

1 5 #SUP: 2

2 3 #SUP: 3

2 5 #SUP: 4

2 3 5 #SUP: 3

3 5 #SUP: 3

========== ECLAT - STATS ==========

Transactions count from database: 5

Frequent itemsets count: 15

Total time ~ 5 ms

Maximum memory usage: 0.6226730346679688 mb

==========

It is found that:

FP-tree: a novel data structure storing compressed, crucial information about frequent Patterns, compact yet complete for frequent pattern mining.

- FP-growth: an efficient mining method of frequent patterns in large Database: using a highly compact FP-tree, divide-and-conquer method in nature.
- The Eclat algorithm searches for the basic elements as the first in-depth search.

VI. CONCLUSION

Frequent itemset mining is an important task in association rule mining. It has been found useful in many applications like market basket analysis, financial forecasting etc. In this paper three algorithms are used to extract itemsets frequents.

After this study, we find that Apriori and FP-Growth are aiming to find out complete set of patterns but, FP-Growth is more efficient than Apriori in respect to long patterns, and Eclat is more efficient than FP-Growth.

VII. REFERENCES

[1] Hahsler, M .; Gruen, B. & Hornik, K. arules - Un environnement informatique pour les règles de l'Association minière et les ensembles d'articles fréquents . Journal of Statistical Software, 2005, 14, 1-25

[2] M. Kaur, U. Garg, et S. Kaur, « Advanced Eclat Algorithm for Frequent Itemsets Generation », p. 19.

[3] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," Knowledge and Information Systems, vol. 14, no. 1, pp. 1–37, Dec. 2007.

[4] Garima Jain, Diksha MauryaExtraction of Association Rule Mining using Apriori algorithm with Wolf Search Optimisation in R ProgrammingInternational Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8, Issue-2S7, July 2019

[5] Tri Thanh Nguyen , A Compact FP-tree for Fast Frequent Pattern Retrieval, 27th Pacific Asia Conference on Language, Information, and Computation, 2013.

[6] Manjit kaur , Urvashi Grag , ECLAT Algorithm for Frequent Itemsets Generation, International Journal of Computer Systems , Volume 01– Issue 03, December, 2014