Université Catholique de Louvain

Pattern Mining : Implementing Apriori

LINGI2364

Group 31
Antoine Geller - DATS2M - 75991900
Trudel Nguepi - DATS2M - 74151900

Professor : Nijssen Siegfried

12 mars 2021

1 Introduction

In the framework of the course of pattern mining, we implemented two different algorithms which aim to find the most frequent item sets of a data set of transaction. The task of frequent item set mining is the task of finding all item sets that are frequent. It is defined by the following formula :

$$\{I \subseteq \mathscr{I} | support_D(I) \geq \theta\}$$

where :
— $\mathscr{I}$ is the set of all items.
— I represents an item set.
— $support_D(I)$ is the frequency of the item set, I.
— θ is the minimum support threshold.

The first algorithm is the Apriori algorithm and the second algorithm implemented is the well-known ECLAT. We performed our analysis on multiple sets such as *retail*, *chess*, *mushroom* and *accident*. We compare both methods by the time and the memory they use to process the task.

2 Apriori algorithm

The first and most basic algorithm is the Apriori algorithm. "Apriori" stands for events based on knowledge prior to the experience. This approach relies on two different properties. First, the level-wise approach, and second, the anti-monotonicity property. We start with the data set of transactions and the number of times they occur as inputs to our analysis. The transactions have a variable size of items and occurrence.

The level-wise approach is the idea in which the process of the algorithm evolves in levels where the item sets of length k are generated before the item sets of length $k+1$. The level zero is the empty level with no item in the set, which is by definition a frequent item set. The first level is of size one, the second level is of size two, etc until we reach the maximum size of the item set possible.
Anti-monotonicity is the property where

$$\forall X, Y; X \subseteq Y \longrightarrow support_D(X) \geq support_D(Y)$$

This property is useful for the generation of candidates as it states that if an item set X is a subset of the item set Y, then the support of X is higher than the support of Y. Therefore, in the search of the frequent item set, we know that if the minimum support is not met on X, it will not match on the item set Y either.

The first step in the implementation of the algorithm is to generate the candidates of item sets composed of one item, $\mathscr{C}_1$, with their respective frequencies via the function `get_items_stage1`. This function returns a dictionary with, as keys, the item sets composed of one item $i_1, i_2, ..., i_n$ and their frequencies $(f_1), (f_2), ..., (f_n)$ for the data set D. The set of frequent items, $\mathscr{F}_1$, is then retrieved from $\mathscr{C}_1$ by comparing the frequencies of each set with the minimum frequency required.
The frequencies of each set is computed as follow : $freq_d(I) = \frac{support_d(I)}{|T|}$.

The second step in the process of frequent item set mining with Apriori is to use the sets of frequent items generated at level 1 to create the candidates of level 2. This is the only optimisation used in this algorithm. We start from the frequent items at level $k-1$ to generate the candidates of level k. In fact, we could have used the candidates of level $k-1$ to generate the candidates at level k but, with the property of anti-monotonicity, we can assume that the non frequent items from level k can not produce frequent items at level $k+1$.
Once the candidates are generated, we can define the frequent item sets by comparing the frequencies with the minimum support. And re-do the process again for each increasing levels.

This algorithm is implemented iteratively until we reach the last level which is the item set composed of all items.

The principal difficulties in the implementation was to take care of the redundancies problems. By joining two different frequent item sets, one can produce the same candidates as two other sets. For example, i_1, i_2, i_3, i_4 and i_1, i_3, i_2, i_4 will produce the same joined candidates. Therefore, it is necessary to flag these types of redundancies. This algorithm is very computationally intensive as we will see in the following sections. It will be compared to the depth-first-search ECLAT.

The simplified pseudo-code for the Apriori algorithm (Book's extract [1] Frequent Pattern Mining) is the following. It is not a copy of the python code as we omit some details.

Algorithm 1 Apriori algorithm

Apriori(Database D, minFrequency f_{min})

Generate frequent 1-patterns from the set of items by counting matches in the transactions data set ;
k :=2 ;
while $k < n_{items}$ **do**
Generate the candidates, $\mathscr{C}_{k+1}$ by using joins on $\mathscr{F}_k$;
Generate $\mathscr{F}_{k+1}$ by counting candidates in $\mathscr{C}_{k+1}$ with respect to the minimum frequency required f_{min} ;
end while

3 Depth First Search : ECLAT

In this part of the report, we describe the ECLAT algorithm as an implementation version of the Depth-first-algorithm for the frequent item set mining in the initial database. The principal strength of the Depth-First Algorithm against the Apriori is the gradual considerations of portions in the database, which allows us to increase the computational efficiency.

The ECLAT algorithm is mainly based on a vertical representations of the database and uses the concept of incremental projection to calculate item coverage in order to select the most frequent ones :

— The vertical representation of a database consists of associating with each item the set of all the transactions in which it appears.
— The incremental projection consists of starting from a given item, constructing a new table which is the intersection between the current item and the elements which follow it (i.e. in ascending order) ; this is done successively until the addition of a new item no longer makes the set frequent.

The simplified pseudo-code for the ECLAT algorithm (Book's extract [1] Frequent Pattern Mining) is the following. It is not a copy of the python code as we omit some details.

Algorithm 2 ECLAT algorithm

ECLAT(FP , minFrequency f_{min})

for Pi *in* FP **do**
  $FPi = \{\}$
  for Pj *in* FPi, such that j>i **do**
    $Pij = Pi \cup Pj$
    $cover(Pij) = cover(Pi) \cap cover(Pj)$
    $support(Pij) = |cover(Pij)|$
    if $support(Pij) \geqslant f_{min}$ **then**
      Add Pij *in* FPi
    end if
  end for
end for

In this algorithm :

— FP designates the list of current frequent item lists of which only the last items differ (in ascending order) ;

— Pi designates an element of FP ;

— FPi designates the list which will contain the frequent items built from Pi by adding an additional element according to point 3 of the algorithm 2 and which will be used during the next recursive call to ECLAT.

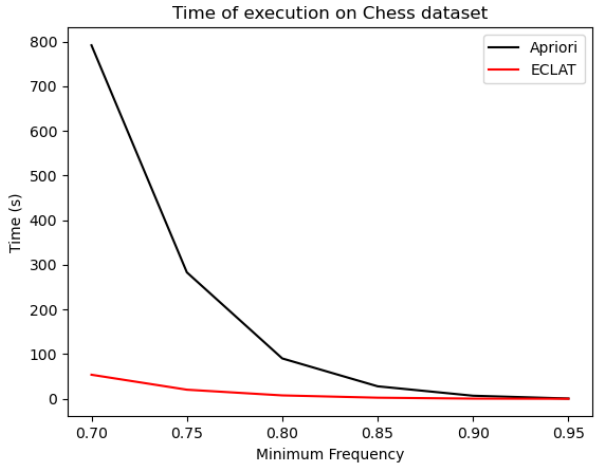
In theory, the Eclat algorithm is much more efficient in terms of computation time compared to the Apriori because it minimizes the computations considering a partial structure of the database and uses this process to browse the database.

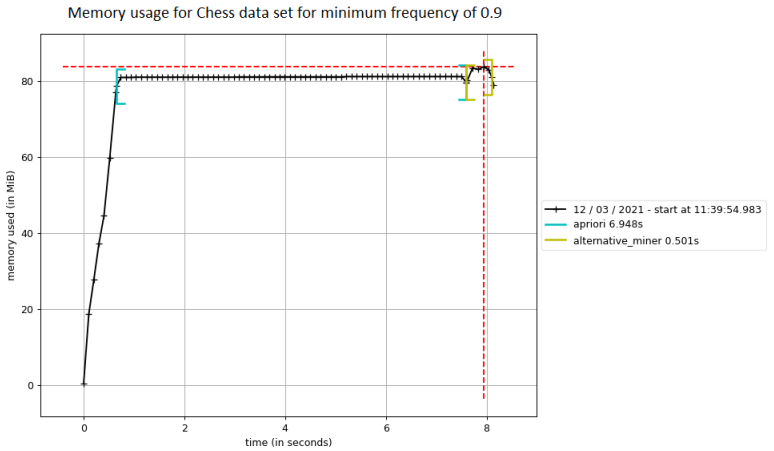
In the rest of this work, we carry out some experiments on different databases to compare these two algorithms and thus verify by experience that the results are in agreement with the theory.

4 Experimental results

Although we present to different experimental results on two data sets, we did our analysis on all the data sets received. The conclusions shown below are recurrent, for our implementations, for all the data sets we tested.

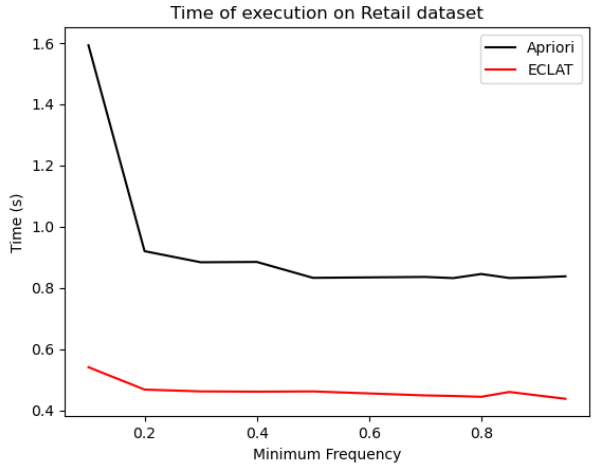
The first results is for the `chess` data set where we have 3196 total transactions and 75 different items. The Apriori and the ECLAT algorithms are compared over 6 different level of minimum frequencies, going from 0.7 to 0.95. As we can observe on the left graphic, the lower the minimum support the higher the time

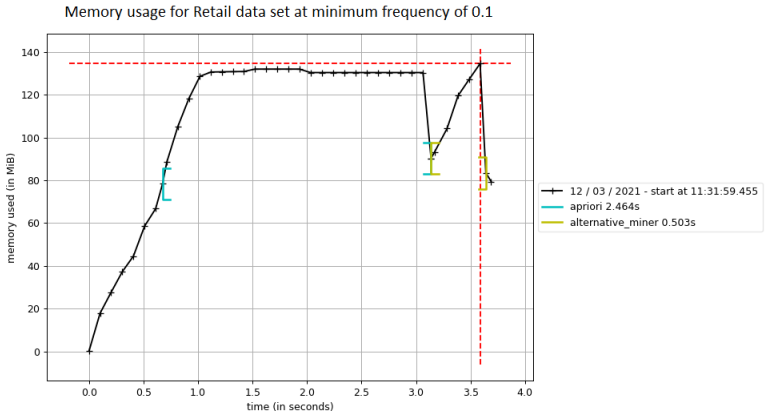




of execution. This is due to a high number of frequent items for lower minimum frequencies. But the speed at which each algorithm time increases is very different. The Apriori algorithm increases exponentially with the decrease of the minimum frequency, while ECLAT is more linear. As we reach higher minimum frequencies, the time of execution tends to reach the same value but with an advantage for ECLAT, this might be due to the fact that for higher minimum frequencies values, we have a small number of frequent items sets. This might be due to the fact that for higher values of minimum frequencies, we have a very small number of items. On the other side, for the right graphic, we can observe the memory usage with respect to time. The black line is the evolution of our `Main.py` file that calls first the Apriori algorithm and then the ECLAT algorithm, both with a minimum support of 0.9. Between the blue brackets is the evolution of memory usage which is constant around 80 MiB. Then, the evolution of the memory usage for ECLAT is represented between the green brackets. We can clearly see that the ECLAT algorithm uses more memory but over less time.

For the second analysis, we used the `retail` data set. This set is composed of 88162 transactions with 16470 different items. Although it is bigger than the chess data set, the time of computation is lower. The Apriori and the ECLAT algorithms are compared over 11 different levels of minimum frequencies, going from 0.1 to 0.95.





The lower time of computation is due to the fact the size of the transactions sets is higher for the chess data set than for the retail data set. The mean size of the transactions sets is 37 while for the retail data set, the mean size is between 7 and 11 (Paper's extract [3] Retail market Basket Data set). Therefore the computation time is higher to check a match in all the transactions sets. Concerning the memory usage, the maximum value is represented by the crossing between the two red dotted line. It is still the ECLAT which takes up more memory but over a shorter period of time.

5 Conclusion

The performance of the ECLAT algorithm is significantly higher than the Apriori algorithm as described in the experimental result section. The performance stands out on two different points. First the overall time of execution is lower for any minimum support and for any data set. Moreover, the slope of the Apriori function is higher than for ECLAT. For the same decrease in minimum frequency, the increase in execution time is higher for the Apriori than for ECLAT. These results are in good agreement with the theory.
One of the disadvantages of the Apriori Algorithm is that it performs a double calculation at each level. A calculation to generate the candidates and a calculation for the most frequent selection, this being due to the fact that this algorithm performs a single pass on the tree (from top to bottom). While ECLAT, although passing several times at a level, it only performs the calculations once by considering a sub-part of the initial tree and gradually building the frequent items from a frequent item. The ECLAT algorithm, although being efficient in terms of computation time, it can sometimes require a lot of memory as shown in the experimental results.

A Annexes

A.1 References

[1] Charu C. Aggarwal (2014). *Frequent Pattern Mining*. Springer

[2] Nijssen Siegfried. *Slide of the course of Pattern Mining*

[3] Tom Brijs. *Retail Market Basket Data Set*