East West University

Final Project Report: Implementation of FP-growth and Apriori algorithm using python programming language

Course Title	**: Data Mining**
Course Code	**:** CSE 477
Section	**:** 01

Submitted To

Jesan Ahammed Ovi
Lecturer,
Department of Computer Science and Engineering

Submitted By

Nasif Wasek Fahim
2017-1-60-037
Md. Jubayer Hossain Abir
2017-1-60-084
Syed Md. Asif Hossain
2017-1-60-086
Mueem Nahid Ibn Mahbub
2017-1-60-089
Tahmina Ahmed Prima
2017-1-60-105

Submission Date

03/06/2021

In data mining finding the frequent patterns from the large dataset is a great task. To conduct this study two of the prominent data mining algorithms were used. One is the Apriori algorithm and the other one is FP - growth. Both these algorithms generate same frequent items from any given dataset. But in practice, the approach to generate those frequent items is quite different. Like in Apriori, candidate set generation is required but in FP - growth no candidate set generation is required.

Apriori and FP – Growth Algorithm

Apriori and FP - growth is both data mining algorithms that can be used to discover the frequent pattern from any data set. They are quite efficient in making association analysis. Both these algorithms use the downward closure property to fasten the search. Many mining algorithms use support and confidence to discover the association in a pattern. Apriori and FP - growth is such kind of algorithms where support is used to prune out the frequent pattern. Although both these algorithms have many things in common and they generate the same output, there are some fundamental differences in the procedure. For example, the problem with Apriori is that if the frequent pattern gets large like 100 then the database must be scanned 100 times. If the frequent pattern's length is n, then it must be scanned n times which is a very costly operation. But with FP - growth it does not matter how big the length is database needs to be scanned only two times which saves a lot of memory and processing.

Datasets

Apriori and FP - growth algorithms have been applied on the chess and korsarak dataset for the requirement of this project and one-third of the dataset has been used. In the chess dataset, there are 3196 transactions and for each transaction, there are 37 items. In the kosarak dataset, there are 990002 transactions. Also, the number of items is different in each transaction. We have compared the time difference between these two datasets after applying the Apriori algorithm and FP - growth algorithm. The given two datasets are in numeric form. Every row is considered a transaction, and, in every transaction, there are numerical elements that are considered as items. So, for example in chess dataset {1, 3, 5, 7} are items that can be found in many of the transactions of chess dataset. So, to find the support of 1 or 3 or 5 these items must be scanned in all the used rows as we have worked on one-third of the dataset.

For example, let us consider a simple dataset to understand the given dataset more comprehensively.

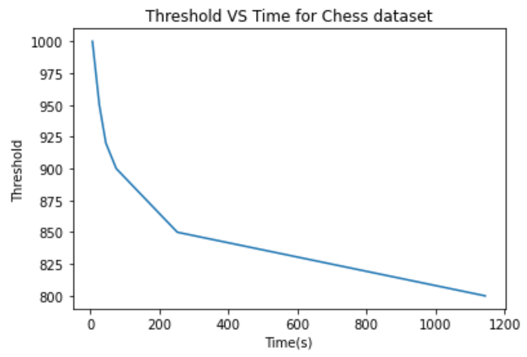
TID	Items
T1	Bread, Butter
T2	Chicken
T3	Milk, Bread, Butter

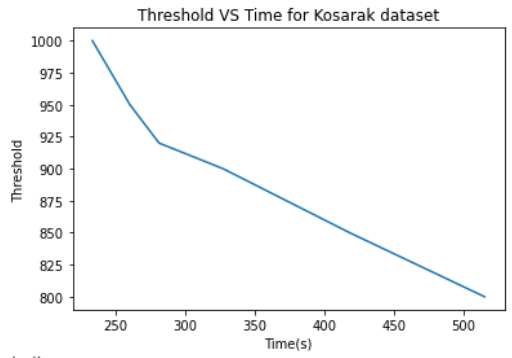
From the above dataset we can see that there are three transactions, and, in each transaction, there are different items. It can be also noticed that the length of each itemset is different in each transaction just like korsarak dataset. To generate frequent patterns from the given dataset apriori or FP – growth can be used. Here the support of bread and butter is two. So, on a given absolute threshold of two bread and butter can be considered as a frequent pattern meaning if we consider these items are from a super shop then it can be excluded that there is an association between bread and butter and there is a higher chance of them being sold together. This was a very simple dataset to understand which has a header and items which are properly named. Bur in chess or korsarak dataset there is no header neither the items are properly named. But still the both the given datasets will work similarly to the above dataset. On the given dataset the given numeric numbers are considered as items and by applying data mining algorithms there can also be a generation of frequent items, which have been done in our project accordingly.

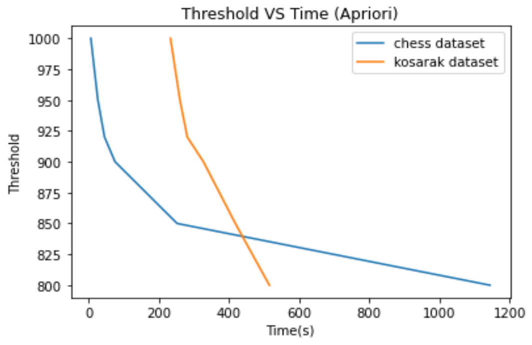
Apriori Algorithm on the Datasets

We have applied the Apriori algorithm on both datasets for different thresholds and monitored the time differences for different thresholds. For the chess dataset, first, we set the minimum threshold at 800 and then run the program. It took 1144.5 seconds and then stopped executing the program because of data rate exceeded. Then we have run the program in different thresholds. For thresholds 850, 900, 920, 950, 1000 we got the time 251.65, 75.3, 45.5, 26, 5.9 seconds respectively and the program executed successfully.

Then we have applied the algorithm on the kosarak dataset for 800, 850, 900, 920, 950, 1000 thresholds and found the execution time 514.7, 418, 327.4, 280.5, 259.5, 233 seconds, respectively.





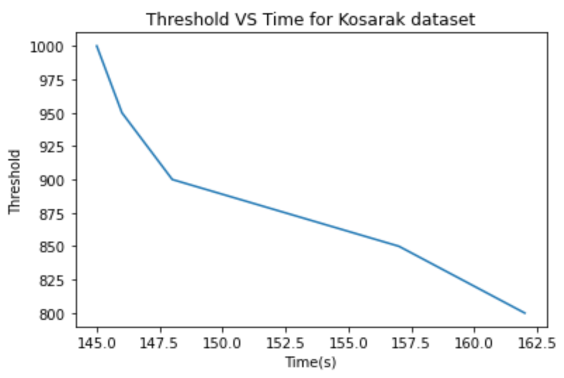


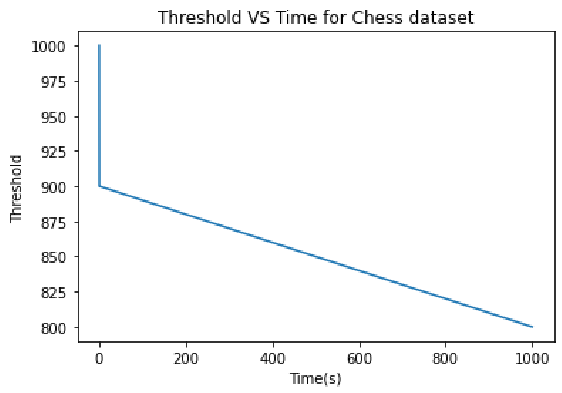
As can be seen, the lower the minimum support level, the longer the execution duration. All the candidate sets are generated by the Apriori Algorithm, and the sets grow. We can observe for the chess dataset in the graph above,

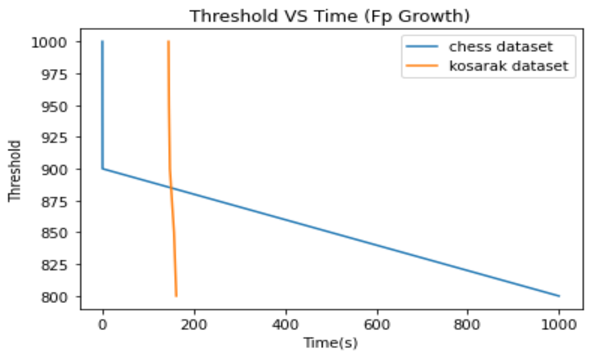
It took far too long to complete for thresholds of 800 or less. Even the generated data grows too large to fit in the ram. The Kosarak dataset, on the other hand, had a more constant execution time than the Chess dataset. Though the minimum threshold of 1000 took longer to run than the Chess dataset, the minimum threshold of 800 took less time. Despite the fact that the Kosarak dataset has more transactions than the chess dataset, threshold 800 required less time. Because the number of itemset in the Kosarak dataset is lower than the number of itemset in the chess dataset. In the Chess dataset, the itemset density is greater. The number of iterations conducted increases as the number of instances increases. Candidate sets will be larger if there are more attributes. The program does more data scans as the size of the data is larger. As a result, threshold 800 took longer for the Chess dataset. The Algorithm will dramatically increase for less than threshold 800 on the chess dataset, resulting in frequent crashes and erroneous results.

FP - Growth on the Datasets

The program took 162, 158, 148, 146, 145 seconds to execute for the Kosarak dataset's minimum thresholds of 800, 850, 900, 950, and 1000, respectively. It took too long to perform the Chess dataset's thresholds of 800 and 850. The program eventually crashed due to a data rate overflow. It took 0.470, 0.25, and 0.112 seconds to successfully perform thresholds 900, 950, and 1000, respectively.







From the above graph, we can observe that, when the minimum support threshold is lesser, it takes more time to execute. In the kosarak dataset, the time vs threshold was consistent. On the other hand, in the chess dataset, for threshold 900-1000, it took only a few moments to execute the program. But for a threshold less than 900, it took so long, and the program crashed because the data rate exceeded. Because FP - growth works badly with a large dataset. FP - growth algorithm must build a tree and store it in the memory. As the chess dataset is large and the density of the itemset is high, so it is possible that the FP - tree could not fit in the memory. That is why the program crashed.

Note

The execution time may vary in a different machine. The configuration of this machine where the algorithms were tested is processor – Intel® core™ i7-7500U CPU @ 2.70GHz – 2.90 GHz, memory – 8.00 GB.

We did not run any heavily weighted application in the background to fully utilize the ram.