A Factual Analysis of Improved Python Implementation of Apriori Algorithm

Chapter · September 2018

DOI: 10.1007/978-981-13-2345-4_11

3 authors:

Dr-Kartick Chandra Mondal
Jadavpur University
54 PUBLICATIONS **235** CITATIONS
SEE PROFILE

Biswadeep Deb Nandy
Jadavpur University
3 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Arunima Baidya
2 PUBLICATIONS **3** CITATIONS
SEE PROFILE

A Factual Analysis of Improved Python Implementation of Apriori Algorithm

Kartick Chandra Mondal, Biswadeep Deb Nandy, Arunima Baidya
Department of Information technology
Jadavpur University
{kartickjgec,biswadeep1997,arunimabaidya}@gmail.com

Abstract. Data mining has been proven as a useful tool for data analysis. Association rule mining is one of the important tasks in data mining for finding meaningful and hidden relationship in data. Apriori is the first attempt to do association rule mining using frequent item set mining over transactional databases. Several implementations of the algorithm in various languages have been done so far for performing this task. In this paper, we are dealing with comparative study and critical analysis of various implementations of Apriori algorithm present in different Python packages and implemented another version of the algorithm which is at par with the existing algorithms but without using any existing libraries available in python. A detailed conclusion has been drawn on our empirical study of Apriori algorithm which will help the researcher to understand and use different packages for data analysis.

Keywords: Data Mining; Association Rule Mining; Apriori Algorithm; Frequent Itemsets Mining; Python Packages;

1 Introduction

Data Mining, also known as Knowledge Discovery in Databases (KDD), includes the task to find anomalies, correlations, patterns, and trends to predict outcomes [12, 4]. Association rule mining is one of the most prominent data mining tasks and is gaining much importance in recent years in many application domains. In general, the KDD is a sequence of processes stated [13] as follows:

- Data cleaning which includes the removal of noise and inconsistency from the data.
- Data integration, where multiple data sources are combined and integrated into one.
- Data selection, where data relevant to analysis task are retrieved.
- Data transformation, in which data is transformed into forms appropriate for mining by performing several aggregation operations.
- Data mining, which includes intelligent methods to extract various patterns in data

- Pattern evaluation, where the various patterns are evaluated and the ones truly representing the knowledge are identified.
- Knowledge representation, including techniques to represent the knowledge.

Association Rule Mining (ARM) is a prominent and a well-explored method for determining relations among variables in large databases. ARM is very popular and important, but expensive, task in data mining [15]. An association rule can be represented in the form of Antecedent => Consequent [support, confidence], where support and confidence are the interestingness measure of the rules. The technique can be used in many data types like sequential data, spatial data, and multimedia data along with transactional data. In today's world, ARM is used in various fields like Telecommunication, Weather Analysis, and Banking Sectors etc.

The Apriori Algorithm is the first attempt for mining association rule by finding frequent itemsets from the dataset. The main idea behind this algorithm is to prune the exponential search space using anti-monotonicity. As the name suggests, the algorithm uses prior knowledge (of the support properties) to prune the search space. It is devised to operate on a database containing a lot of transactions. It uses a "bottom up" approach where frequent subsets are extended by one item at a time. It is generally used to operate on transactions containing data [10]. This algorithm has several real world applications such as [14] market analysis to help the customers in purchasing their items with more ease which increases the sales of the markets or the field of healthcare which helps in detecting adverse drug reactions (ADRs). It produces association rules that indicate all combinations of medications and patient characteristics lead to ADRs.

In this paper, we have presented a comparison and analyze some implementations of Apriori algorithm in Python as well as our own implementation of the same. The rest of the paper is organized as follows. Some brief terminologies are described in Section 2. In Section 3, we have presented a very brief explanation of the Apriori Algorithm. Section 4 gives a brief study about different Python packages implementing the algorithm. This section also presents the detail of our own implementation of the algorithm. The Section 5 includes the various parameters on which we have compared different python packages. The experimental results are shown in Section 6, while Section 7 contains the concluding remarks.

2 Terminologies

Definition 1 (Database): A database D is defined as (I, T) where I refers to the transaction ids and T refers to the set of transactions.

Definition 2 (Transaction): A transaction T is defined as a set of items (known as itemset), where each item is denoted by i.

Definition 3 (Itemset): A non-empty finite set of items I where $I \subseteq T$ in D.

Definition 4 (Support): The support of an itemset Q, denoted by sup(Q), is defined as the frequency of occurrence of Q in D.

$$sup(Q) = (|\{q \in T ; X \subseteq q\}|) / |\{T\}|$$

Definition 5 (Frequent Itemset): An itemset Q with support equal to or more than the user-defined threshold of support (minsup) is called a frequent itemset i.e.,

$$Q \subseteq T \text{ is frequent iff } sup(Q) \geq minsup.$$

Definition 6 (Association Rule): The relationship between two itemsets I1 and I2, $I1 \rightarrow I2$, where I1, $I2 \subset L$ and $I1 \cap I2 = \varnothing$ is called an association rule. I1 is known as antecedent of the rule, while I2 is known as consequent of the rule.

Definition 7 (Confidence): The confidence value of a rule $I1 \rightarrow I2$ is the proportion of transactions T that contains I1 as well as I2. It is denoted by conf($I1 \rightarrow I2$) where

$$conf(I1 \rightarrow I2) = sup(I1 \cup I2)/sup(I1)$$

3 Apriori Algorithm

Apriori algorithm was proposed by Agrawal & Srikant in 1993 [6]. It is used originally for mining frequent itemsets in boolean association rules from transaction dataset [1]. The Apriori algorithm helps in pruning the number of itemsets that are infrequent which needs to be examined because, as a principle, if an itemset is infrequent then all its subsets must also be infrequent.

The algorithm is comprised of the following steps [14]:

Step 0: Take the minsup and the itemsets from the user. Let us consider that the minsup is 60% and the transaction database as it is shown in Table 1.

Step 1: Calculate the support of all items. A frequency table of all the items that occur in the transactions is made as shown in Table 2. In this table, each itemset is annotated with its support. The support of an itemset is basically its frequency, i.e., how many times the itemset appears in the transaction database. Here, the itemset {9, 40} has support value 4 because it appears in transactions i2, i3, i4 and i5.

Step 3: We know that only those elements are significant for which the support is greater than or equal to the threshold support (minsup). So, we discard those items whose support is less than minsup. The final result with the significant itemsets and their support is denoted in Table 3.

The diagrammatic representation of the functional flow of the algorithm [5] is provided in Figure 1.

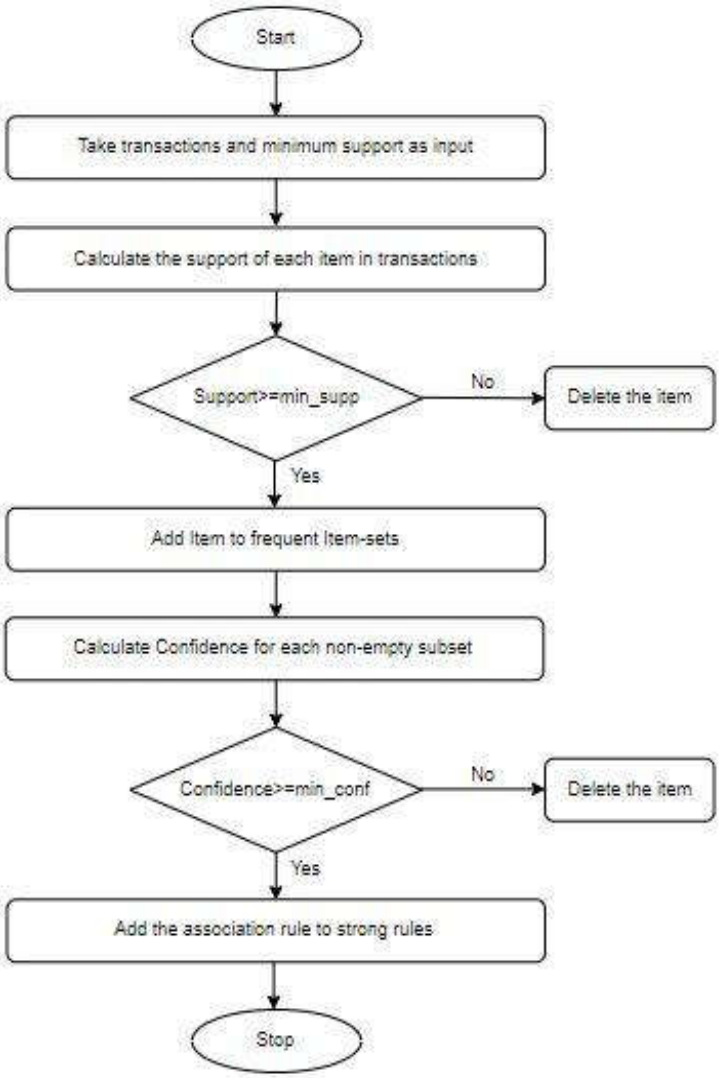

Figure 1: Flowchart of the Apriori algorithm

Transaction id	*Items*
i1	7, 14, 18
i2	9, 14, 40
i3	7, 9, 14, 40
i4	9, 40
i5	7, 9, 14, 40

Table 1: Data containing all transactions

Itemsets	*Support*
7	3
14	4
7, 14	3
9	4
40	4
9, 14	3
9, 40	4
14, 40	3
9, 14, 40	3

Table 3. Final result after running Apriori algorithm on the dataset.

Itemsets	Support	Itemsets	Support
7	3	9,40	4
14	4	14,40	3
18	1	9,14,40	3
7,14	3	7,9	2
7,18	1	7,40	2
14,18	1	7,9,14	2
7,14,18	1	7,9,40	2
9	4	7,14,40	2
40	4	7,9,14,40	2
9,14	3		

Table 2: The intermediate table containing all itemsets and their respective supports

4 Python Implementations of Apriori Algorithm

In this section, we have provided an outline of various packages that are implemented and widely used in Python which have implemented the Apriori Algorithm. At the end of this section, we have also explained our implementation of the apriori algorithm.

4.1 Explanation of existing packages

1. **akapriori 0.1.0:** It works for both Python 2.x and 3.x. It does not support file input. Input is given by a program consisting of a list of transactions and, this input is passed to the package. The resultant data is contained in a frozen set corresponding to their support and confidence.
2. **apriori-1.0.0:** It imports numpy package for faster computation. Input data is set and returned from function loadDataSet(). Mapping is done between frozen set and Ck, candidate k-tuples. It returns candidate generation with support in list Lk and, association rules are also generated by calculating the confidence and pruning of rule is done by comparing with the minimum confidence.
3. **apyori-1.1.1:** It imports json and csv packages. It is an implementation of Apriori algorithm with Python 2.7 and 3.3 - 3.5, provided both as APIs and as command line interfaces. Input to this implementation is a .tsv file (transaction values are tab separated). Output can be generated in.tsv or .json file format.

Basic usage: python apyori.py < test.tsv
TSV Output: python apyori.py -f tsv < test.tsv
Specify details: python apyori.py -s 0.5 -c 0.5 < test.tsv

4. **myapriori-0.0.22:** It imports apriori functionalities from the apyori package. In addition to this, it requires the support of numpy as well as pandas library. Transaction data can be passed as a pandas.DataFrame or list of lists.

4.2 Explanation of our package

The originality of our implementation is, no external libraries have been imported or used in it. Input data is passed in the form of a file consisting

space separated values as transaction. We have done our implementation in Python 3.6.5. Command line input pattern for running our implementation is: python3 apriori.py -f “path/filename” -s “minsupport” -l “number of transactions”.

Eg: python3 apriori.py -f datasets/set1.dat -s 0.2 -l 100

Default value of minsup is set to 0.25 and the default value of the number of transactions is set to -1. We have created a class Apriori and, we have defined the following functions in it:

- createC1(self): This function creates C1 candidate k tuples.
- filterCk(self, Ck): This function filters Ck candidate tuples and returns Lk and its supports.
- createCk(self, Lk, k): This function creates a set of Ck candidate k tuples.

The output generated is a list of frequent itemsets. Figure 2 shows the activity diagram of our implementation.

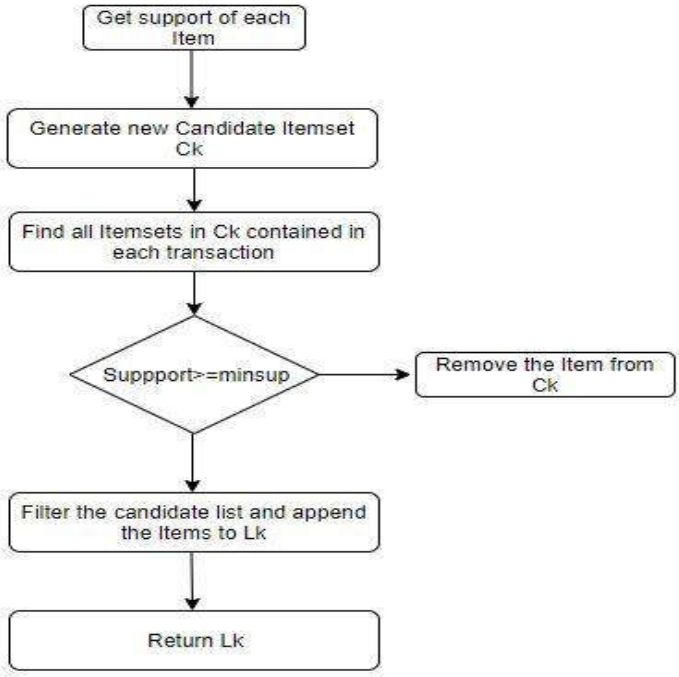

Figure 2: Activity diagram of our Apriori implementation

4.3 Comparison between packages

Following are the characteristics which we have used for comparison on various packages:

1. **Imports**: These depict the various Python libraries used by the apriori packages.
2. **Input pattern**: This denotes the type of input data that the program requires.
3. **Data Structures**: This signifies the various data structures used in different packages.
4. **Time Complexity**: It refers to the implementation complexity that estimates the time taken for an algorithm to run for that particular implementation.
5. **Space Complexity**: Space Complexity of an algorithm is total space taken by the implementation with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

For the time and space complexities, we have studied different implementations and used Big O notation to show the analysis of the same. The table 4 given below shows the comparison between the different packages, according to their features mentioned above:

Packages	Imports	Input Pattern	Data Structures	Time	Space
akapriori 0.1.0	collection	List of sets	Defaultdict, Frozenset, Set, List	O(n^2)	O(n^2)
apriori-1.0.0	numpy	List of lists	List, Dictionary	O(n^3(log n))	O(n^2)
apyori-1.1.1	collection	Diction ary	List, Dictionary, Set, Frozenset, Namedtuple	O(n^d)	O(n^2)
myapriori-0.0.22	apyori	Type flexible	List	O(n^d)	O(n^2)
Our Implementation	collection	List of lists	List	O(n^2)	O(n^2)

Table 4: Comparison of various packages on the basis of features where d=unique items in entire set of transactions**.**

5 Experiments and Critical Analysis

In this section, we have described about our experimental setup, the kinds of dataset we have used as well as the statistical and experimental results of our experiment.

5.1 Experimental Setup

This sort of experimental study can be very sensitive to hardware configuration and the possibility of other programs or tasks running at the same time in a multiprocessing and multiprogramming environment. So, we have used a single machine for carrying out all the experiments, whose configuration is stated as follows:

- **Processor**: 6th Generation Intel Core i5-6 300HQ Quad Core (6M Cache, up to 3.2 GHz)
- **Memory**: 8GB 1 DIMM (1x8GB) DDR3L 1600M hz
- **Disk/Hard Drive**: 1TB (5400rpm) Hybrid HDD with 8GB Flash
- **System Type**: 64 bit Operating System, x64-based Processor
- **Operating System**: Windows 10 Home Single Language

5.2 Dataset Preparation

Conceptually, there are two types of databases, namely, dense and sparse. Sparse databases refer to the databases where most of the cells are filled with null values [7]. Dense databases, on the other hand, have fewer null values and, are filled with very useful and significant information. According to many studies done in literature where the Apriori algorithm is applied to a dense dataset, its performance declines because of the long patterns that emerge. When the parameters are low and the database is dense, it searches through all possible itemset combination and becomes computationally infeasible. So, for our experiments, we have considered sparse databases [2, 3, 14].

We have divided our data into four sets with respect to the number of transactions. Set 1, Set 2, Set 3 and Set 4 are comprised of a set of 5, 10, 100 and 1000 transactions respectively. Additional features about the datasets are given in Table 5. To check the performances of the packages, we have considered three different minsup values which are 20%, 40% and 60%, respectively.

Set No.	Minimum # of items in a transaction	Maximum # of items in a transaction	Total # of distinct items
Set 1	2	4	5
Set 2	2	7	19
Set 3	2	10	101
Set 4	1	9	869

Table 5: Table signifying the features of the datasets used for experiments.

5.3 Result Generation

We have generated the results by investigating the total runtime and the memory consumed by the different implementations where *Runtime* denotes the total time for which a program runs and *Memory consumption* signifies the amount of memory used by the program while running. Now, we have considered these two parameters for our experimental study as these two are vital benchmarks for stating the efficiency of any algorithm. The complete experimental result sets are shown in Table 6 and Table 7.

PACKAGES DATASETS	Minsup (in %)	akapriori-0.1.0	apriori-1.0.0	apyori-1.1.1	myapriori-0.0.22	Our implementation
SET 1	20 40 60	0.001505 0.001503 0.002023	0.003071 0.002618 0.000789	0.015994 0.012031 0.011034	0.019281 0.018418 0.017692	0.006554 0.006017 0.007064
SET 2	20 40 60	0.002007 0.002006 0.002005	0.001162 0.000202 0.000187	0.017602 0.013995 0.011017	0.026016 0.025351 0.019815	0.006023 0.006016 0.005982
SET 3	20 40 60	0.004018 0.003973 0.003008	0.002013 0.001904 0.001919	0.018584 0.015036 0.013035	0.032692 0.028407 0.025493	0.013034 0.013579 0.011539
SET 4	20	0.297486	0.128429	0.321865	0.475892	0.466642

	40 60	0.180798 0.180479	0.125685 0.131158	0.315902 0.202961	0.454569 0.382356	0.448878 0.342496

Table 6: Runtime (in seconds) for four different datasets for different minsup values (20%, 40% and 60% respectively).

PACKAGES DATASETS	Minsup (in %)	akapriori-0.1.0	apriori-1.0.0	apyori-1.1.1	myapriori-0.0.22	Our implementation
SET 1	20 40 60	14454784 14401536 14352384	15425536 15560704 15589376	15454208 15417344 15433728	15734314 15685931 15689892	15114240 15081472 15101952
SET 2	20 40 60	14430208 14508032 14393344	15491072 15560704 15429632	15470592 15470592 15482880	15891652 15886073 15797435	15110144 15044608 15077376
SET 3	20 40 60	14587552 14528512 14516224	15474688 15572992 15388672	15566496 15540224 15450112	15725689 15725645 15656892	15159296 15134720 15073280
SET 4	20 40 60	15609856 15577088 15458304	17137664 16879616 16637952	15765504 15536128 15716352	18561245 17959871 17936814	16764928 16633856 16478208

Table 7: Memory requirement (in bytes) for four different datasets for different minsup values (20%, 40% and 60%, respectively).

5.4 Result Analysis

In this paper, we deal with comparison between the different packages of Apriori algorithm. The analysis of results generated from above experiments for different packages are given below on different parameters.

Runtime. Figure 3 showcases the runtime (in seconds) for four different data sets of data (Set1 (Figure 3 (a)), Set2 (Figure 3 (b)), Set3 (Figure 3 (c)) and Set4 (Figure 3 (d)), respectively) for different minsup values (20%, 40% and 60%, respectively). The data corresponding to minsup value = 60% is marked with grey, minsup value = 40% is marked with dark grey, and the one with minsup value = 20% is marked with black.

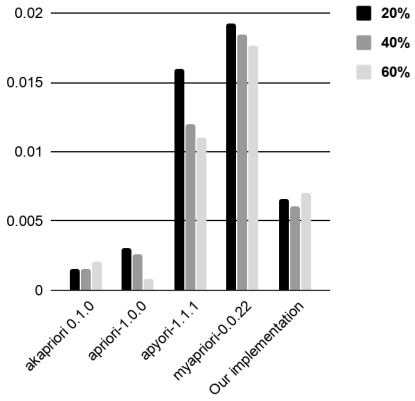

Figure 3 (a): Runtime (in Second) for dataset 1

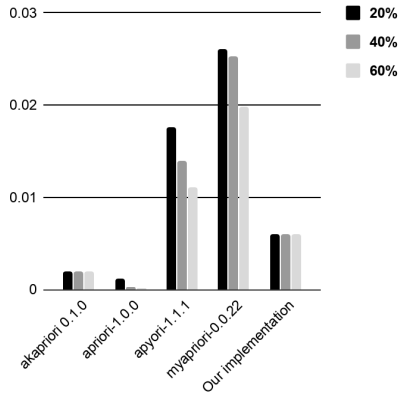

Figure 3 (b): Runtime (in Second) for dataset 2

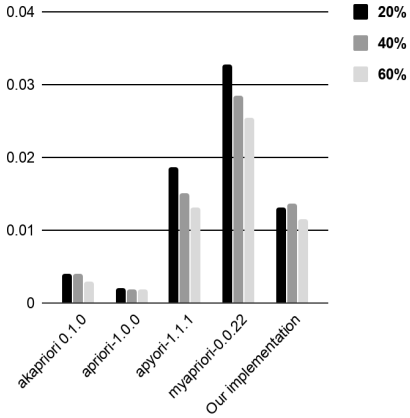

Figure 3 (c): Runtime (in Second) for dataset 3

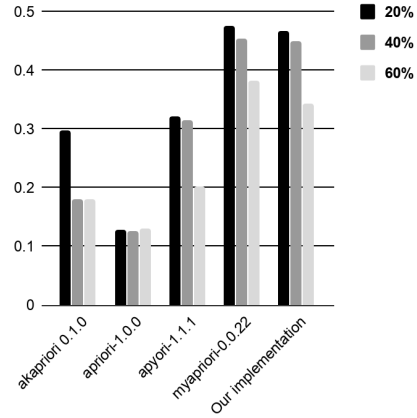

Figure 3 (d): Runtime (in Second) for dataset 4

Figure 3: Runtime analysis of different implementations of Apriori Algorithm

Memory Consumption Figure 4 showcases the memory consumption (in bytes) for four different datasets of data (Set1 (Figure 4 (a)), Set2 (Figure 4 (b)), Set3 (Figure 4 (c)) and Set4 (Figure 4 (d)), respectively) for different minsup values (20%, 40% and 60%, respectively). The data corresponding to minsup value 60% is marked with grey, minsup value 40% is marked with dark grey, and the one with minsup value 20% is marked with black.

Akapriori implements candidate item pruning using the help of numpy package. From the resultant graphs it is clear that for large datasets, packages apriori and akapriori are almost at par.

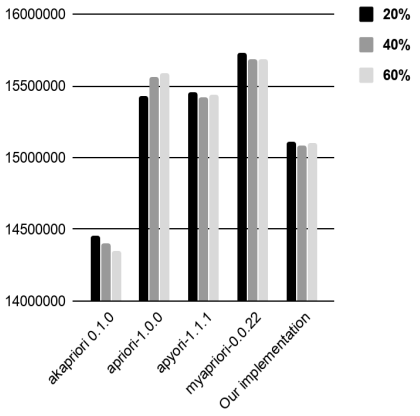

Figure 4 (a): Memory Consumption (in Bytes) for dataset 1

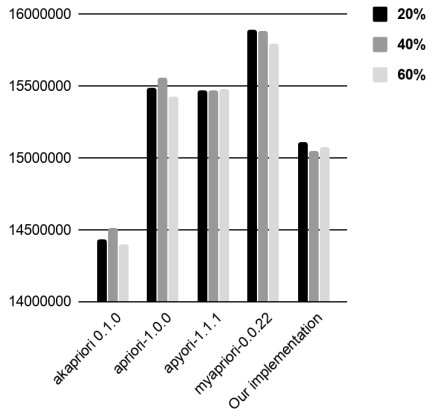

Figure 4 (b): Memory Consumption (in Bytes) for dataset 2

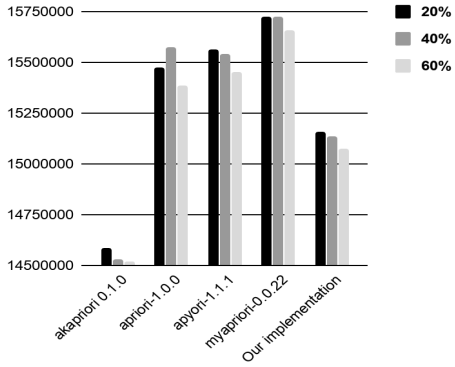

Figure 4 (c): Memory Consumption (in Bytes) for dataset 3

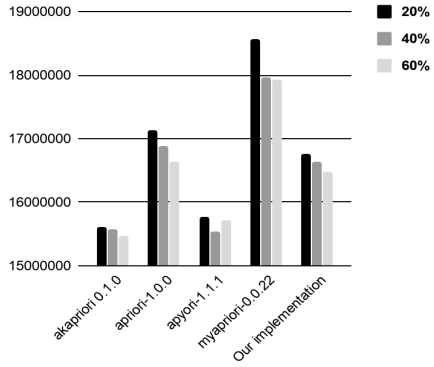

Figure 4 (d): Memory Consumption (in Bytes) for dataset 4

Figure 4: Memory consumption analysis of different implementation of Apriori algorithms

6 Conclusion

Apriori algorithm is the first try to generate association rule mining. Different versions of Apriori algorithm have been developed but no comparison is available with different well known implementation of the same algorithm from performance perspective. We have done an empirical analysis with different algorithmic implementation of the Apriori algorithm done in Python language. This study will help the researchers working in this domain to choose appropriate version of the algorithm for experiment. With respect to our experimental evaluation as stated in this paper, we conclude that the same algorithm might have different performance for different data sets and minimum supports. Lower the minimum support, higher is the time and memory consumption. We see clearly how different implementations perform with respect to time and memory consumption. From our study, akapriori turns out to be the best package in terms of performance, with our implementation coming as a close second in terms of memory and third in terms of time. In addition to this, we have also found that the performance of the Apriori algorithm is mainly dependent on the data structures used. We would like to add that our package has not used any present Python libraries, so there is much scope of improvement in future using techniques like data pruning and dynamic itemset counting.

Reference

1. https://www.philippe-fournier-viger.com/spmf/Apriori.php.
2. https://wiki.csc.calpoly.edu/datasets/wiki/apriori
3. http://fimi.ua.ac.be/data/
4. J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kauffman, San Francisco, 2000
5. M. Mittal, Efficient Ordering Policy for Imperfect Quality Items Using Association Rule Mining., 14, 2014
6. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, In Proceedings of the 20th VLDB Conference, 1994, pp. 487-499
7. F. Ye, J. Wang and B. Shao, "New algorithm for mining frequent itemsets in sparse database," International Conference on Machine Learning and Cybernetics, Vol. 3, Guangzhou, China, 2005, pp. 1554-1558.
8. J. Han, J. Pei, Y. Yin, R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, 2004, pp. 53-87.

9. A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases", VLDB, 1995, pp. 432-443.
10. C. Borgelt, R. Kruse, Induction of Association Rules: Apriori Implementation. In: Härdle W., Rönz B. (eds) Compstat. Physica, Heidelberg, 2002
11. D. N. Goswami et. al., “An Algorithm for Frequent Pattern Mining Based On Apriori”, International Journal on Computer Science and Engineering, Vol. 02, No. 04, 2010, 942-947
12. S. Jain, R. Raghuvanshi, and Md. Ilyas, A Survey Paper on Overview of Basic Data Mining Tasks, “International Journal of Innovations & Advancement in Computer Science”, Vol. 6, Issue 9, 2017
13. A. Silberschatz and A. Tuzhilin, "What makes patterns interesting in knowledge discovery systems," IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 6, 1996, pp. 970-974.
14. M. Hegland, “The Apriori Algorithm – A Tutorial”, Mathematics and Computation in Imaging Science and Information Processing, vol. 11, pp. 209-262, World Scientific Publishing Co., 2007
15. J. Hipp, U. Güntzer, and G. Nakhaeizadeh, Algorithms for association rule mining — a general survey and comparison, SIGKDD Explor. Newsl., Vol. 2, No. 1, 2000, pp: 58 -- 64