Journal of Statistics & Management Systems
ISSN 0972-0510 (Print), ISSN 2169-0014 (Online)
Vol. 28 (2025), No. 6, pp. 1179–1189
DOI : 10.47974/JSMS-1483

An Apriori algorithm-based framework for measuring and improving transactional dataset efficiency

Sonam *
Jyoti †
Department of Computer Science & Applications
Baba Mastnath University
Rohtak 124021
Haryana
India

Abstract

There are plenty of algorithms designed for mining association rules, but Apriori stands out as one of the most popular. It's especially useful for identifying frequent product sets in large datasets and uncovering association rules that help with knowledge discovery. However, the innovative Apriori algorithm has a significant drawback: it spends too much time scanning the entire database repeatedly to find frequent product sets. This paper tackles that issue by presenting an enhanced Apriori algorithm version. The enhanced approach avoids the need for multiple full-database scans by focusing only on a subset of relevant dealings. Experiments conducted on various transaction groups and with different tiniest support values reveal that the amended algorithm slashes execution time by an impressive 67.38% compared to the innovative. These results clearly show how the proposed enhancement makes Apriori much more efficient and time-saving, offering a practical solution for modern data mining challenges.

Subject Classification: *68N30.*

Keywords: *Enhanced apriori, Frequent product-sets, Support, Candidate product-sets, Computational time.*

1. Introduction

As technology continues to evolve and the need to gain meaningful insights from large datasets grows [1], data mining has become an essential

* *E-mail:* **sonamyadav2706@gmail.com (Corresponding Author)**
† *E-mail:* **pragyavij123@gmail.com**

tool. Data is often stored in systems such as large-scale repositories, analytical processing tools, structured databases, and other data storage platforms. These datasets can be incredibly vast, sometimes spanning terabytes or more. Often referred to as Knowledge Discovery Processes (KDP), data mining brings together methodologies from a wide range of fields, including statistical analysis, artificial intelligence, data systems, machine learning models, and information retrieval techniques. KDP methodologies facilitate the rapid extraction of meaningful patterns within a manageable timeframe [2]. This process involves several stages, including data cleansing, mining, and pattern interpretation.

2. Associated Work

The process of identifying common product sets is a critical step in connotation withdrawal, aimed at uncovering commonly occurring product sets within a transaction database. This step serves as the foundation for numerous data mining applications, including discovering association rules, analyzing episodes, building classifiers, performing clustering, and identifying correlations, among others [2]. Several methods have been developed to extract frequent product sets, which can generally be divided into two main approaches: candidate generation and pattern growth.

The Apriori algorithm [3] exemplifies the candidate generation approach. It works by creating (m+1)-product candidate sets from frequent product sets of length (m). The incidence of these sets is determined by numeration their occurrences across dealings. On the other hand, the FP-growth method, introduced by Han in 2000, embodies the pattern growth strategy. Utilizing a specialized data structure known as the FP-tree, this method identifies frequent product sets without explicitly generating candidate sets [4]. Instead, it constructs a conditional pattern base derived from frequent 1-product sets, which is competently built using the node-link edifice allied with the FP-tree.

3. Apriori Algorithm

It's the most commonly used, easiest, easiest-to-immobilize algorithm that's applied in order to excavate each frequently presented set of objects in the database with the assistance offered by the given constraints [5-7]. The algorithm searches the database for frequently presented sets of products [8-9]. For the result's identical result, two constraints are

implemented: the cap restriction(subliminal) uses an interim probability with the intention that dawn arise the statistics products if it's True, and the likelihood that the products' information are presented in the set's collection, it's represented with the threshold below (Min Supp ()) [6, 10, 11].

4. Apriori Algorithm Drawbacks

Despite its simplicity and clarity, the Apriori algorithm has some notable limitations. One major drawback is the significant time expenditure caused by managing a large the total count of candidate sets., particularly when dealing with numerous Frequent product sets identified under low tiniest support thresholds., or huge product sets. For occurrence, If the number of frequent 1-productsets is 10^4, the algorithm must produce over 10^7 candidates for 2-product sets, which must then be evaluated and accumulated [2]. Additionally, identifying frequent patterns of size 100 (e.g., m1, m2, …, m100) requires generating 2^{177} candidate product sets [1], which is computationally expensive and time-consuming. This process involves evaluating numerous candidate product sets and repeatedly scanning the database to locate these candidates. Consequently, the Apriori algorithm becomes highly inefficient and performs poorly, particularly when memory capacity is limited, and the database contains a proliferation of dealings [5].

In this tabloid, we present an alternative method aimed at reducing the time required to search catalogue dealings for frequent product sets.

5. The amended algorithm of Apriori

This segment focuses on the enhancements made to the Apriori algorithm. It includes an overview of the amended Apriori concepts, an example illustrating the enhanced algorithm, and a detailed analysis and evaluation. Additionally, experimental results are presented to demonstrate the effectiveness of the proposed improvements.

5.1 *The Amended Apriori Ideas*

To understand the enhanced Apriori algorithm, the following definitions are introduced:

1. **Description 1:** Let Z = {Z1, Z2, ..., Zm}, where m ≥ 1, represent a set of dealings. Each transaction Zj = {J1, J2, ..., Jn}, where n ≥ 1, is a set

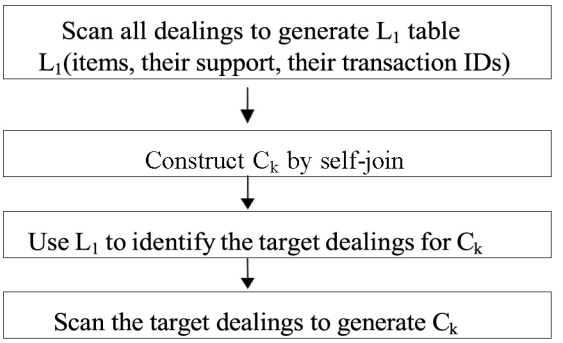

Figure 1

Steps for C_k group

of products. An m-product set is defined as {j1, j2, ..., jm}, where m ≥ 1. Here, the m-product set is a subset of J.

2. **Description 2:** The provision count, denoted as σ (product set), refers to the regularity of amount of an artefact set in dealings.
3. **Definition 3:** Cm denotes the intrant product set of size m, and L_m represents the regular product set of extent.

The proposed technique enhances Apriori to efficiently produce candidate sets in less time. The process begins by scanning all transactions to create the initial set, which includes the items, their respective frequencies, and the transaction identifiers where these items occur. This initial set is then used as a reference to generate subsequent sets, such as L2, L3, and so on, up to L_k. To generate the second set, we apply a self-merge on L1 to form two-item sets represented as pairs (m,n), where m and n are items from the second set.

Rather than examining all transaction records to compute the frequency of each candidate, we utilize L1 to pinpoint the transaction IDs containing the item with the smallest frequency for each pair. This approach narrows down the scanning process to only those transactions that are relevant for generating the second set. The same strategy is applied to construct three-item sets, represented as set of three (m, n, o), where m, n, and o are items in the third set. Using L1, we identify the transaction IDs for the least frequent item among x, y, and z, ensuring that only pertinent transactions are scanned for this step. This iterative process continues until no additional frequent product sets are discovered.

The entire process is depicted in Figure 1.

5.2 *The amended Apriori*

The development of the procedure can be designated as follows:

1. **Generate Initial Product sets:** Create L1 by finding the frequent 1-product sets from the transaction data T:

   L_1 = find_frequent_1_product sets(T);

2. **Iterative Candidate Generation:** Repeat the following process for k = 2 until L(k-1) is empty (L(k-1) ≠ Φ):

   For (k = 2; $L_{(k-1)}$ ≠ Φ; k++) {

3. **Generate Candidate Product sets (Ck):** Construct Ck, the set of candidate product sets, from L_{k-1}:

   C_k = candidates generated from L_{k-1};

4. **Identify Tiniest Support Product:** Find the product x in Ck with the tiniest support count using L1:

   x = Get_product_min_sup(C_k, L_1);

5. **Retrieve Relevant Dealings:** Identify the transaction IDs (Tgt) containing product x:

   Tgt = get_Transaction_ID(x);

6. **Count Support in Relevant Dealings:** For each transaction t in Tgt, update the support count for all products in C_k that appear in t:

   For each transaction t in Tgt Do

   Increment the count of all products in C_k that are found in Tgt;

7. **Generate Frequent Product sets:** Update L_k by including only those products in C_k that meet the tiniest support threshold:

   L_k = products in C_k ≥ min_support;

8. **End of Iteration:** Close the loop for k:

   End;

9. **Repeat Process:** Continue this process until no new frequent product sets can be generated:}

5.3 *A sample of the amended Apriori*

Suppose we have a transaction set D consisting of 9 dealings, with a tiniest support threshold of 3. The transaction set is presented in Table 1.

Table 1

The dealings

X_ID_s	$Products_p$
Xs1	Pp1, Pp2, Pp5
Xs2	Pp2, Pp4
Xs3	Pp2, Pp4
Xs4	Pp1, Pp2, Pp4
Xs5	Pp1, Pp3
Xs6	Pp2, Pp4
Xs7	Pp1, Pp4
Xs8	Pp1, Pp2, Pp3, Pp5
Xs9	Pp1, Pp4, Pp3

Table 2

The candidate 1-product set [10]

Products	Support	
Pp1	6.0	
Pp2	7.0	
Pp3	5.0	
Pp4	3.0	
Pp5	2.0	Deleted

First, scan all dealings to identify the normal 1-itemset (Table 2 [10]), which includes the products, their sustenance counts, and the operation IDs where these products appear. Next, remove contenders that are infrequent or have support counts below the tiniest support threshold (minm_sup). The resulting frequent 1-itemset is displayed in Table 3 [8].

Table 3

Frequent 1_product set

Products	Support	X_IDs	
Pp1	6	Xs1, Xs4, Xs5, Xs7, Xs8, Xs9	
Pp2	7	Xs1, Xs2, Xs3, Xs4, Xs6, Xs8, Xs9	
Pp3	5	Xs5, Xs6, Xs7, Xs8, Xs9	
Pp4	3	Xs2, Xs3, Xs4	
Pp5	2	Xs1, Xs8	Deleted

The next step involves generating candidate two-itemsets from the initial set. To determine the frequency of each itemset, we break each two-itemset into its individual components and then use the initial table to locate the transactions containing these components, instead of scanning the entire transaction dataset [4].

For instance, consider the first two-itemset in Table 4, represented as (P1, P2). In the traditional Apriori algorithm, all nine transactions would need to be scanned to find this itemset. However, in our enhanced approach, (P1, P2) is split into P1 and P2, and the initial set is used to identify the component with the lowest frequency. In this case, P1 has the smallest frequency value. Based on this, we limit our search for (P1, P2) to only the relevant transactions: Xs1, Xs4, Xs5, Xs7, Xs8, and Xs9 [6].

Table 4

Frequent 2_product set

Products	Support	Min	Found in	
Pp1, Pp2	4	Pp1	Xs1, Xs4, Xs5, Xs7, Xs8, Xs9	
Pp1, Pp3	4	Pp3	Xs5, Xs6, Xs7, Xs8, Xs9	
Pp1, Pp4	1	Pp4	Xs2, Xs3, Xs4	Deleted
Pp2, Pp3	3	Pp3	Xs5, Xs6, Xs7, Xs8, Xs9	
Pp2, Pp4	3	Pp4	Xs2, Xs3, Xs4	
Pp3, Pp4	0	Pp4	Xs2, Xs3, Xs4	Deleted

The similar thing to generate 3-product set conditional on L_1 table, as it is shown in table 5.

Table 5

Frequent 3-product set

Products	Sustenance	Min	Found in	
Pp1, Pp2 , Pp3	2	Pp3	Xs5, Xs6, Xs7, Xs8, Xs9	Deleted
Pp1, Pp2 , Pp4	1	Pp4	Xs2, Xs3, Xs4	Deleted
Pp1, Pp3 Pp4	0	Pp4	Xs2, Xs3, Xs4	Deleted
Pp2, Pp3 , Pp4	0	Pp4	Xs2, Xs3, Xs4	Deleted

For a given common product set, L_k, all non-empty subsets that meet the tiniest confidence ar e identified, and candidate association rules are then generated.

In the preceding example, if we analyze the number of scanned dealings required to obtain the (1, 2, 3)-product sets using the innovative Apriori algorithm and our amended Apriori algorithm, we can observe a clear difference. Table 6 highlights that while the number of dealings scanned for the 1-product set is the same for both algorithms, as the value of $_k$ in the $_k$-product set increases, the gap widens significantly [3,8]. This means our amended Apriori algorithm scans far fewer dealings compared to the innovative algorithm, resulting in reduced time consumption for generating candidate support counts.

Table 6

Number of Dealings Scanned During Experiments

	Innovative Apriori	Amended Apriori
1-product set	45	45
2-product set	53	24
3-product set	35	13
summation	133	82

We implemented both the innovative Apriori and our amended Apriori algorithms and tested them using five different groups of dealings, which are as follows:

- **X1:** 554 dealings
- **X2:** 901 dealings
- **X3:** 1231 dealings
- **X4:** 2361 dealings
- **X5:** 3001 dealings

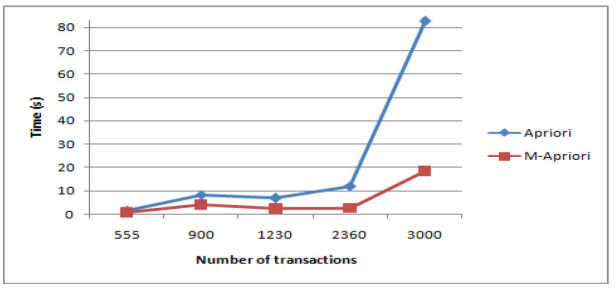

Figure 2

Time consuming contrast for dissimilar groups of communications

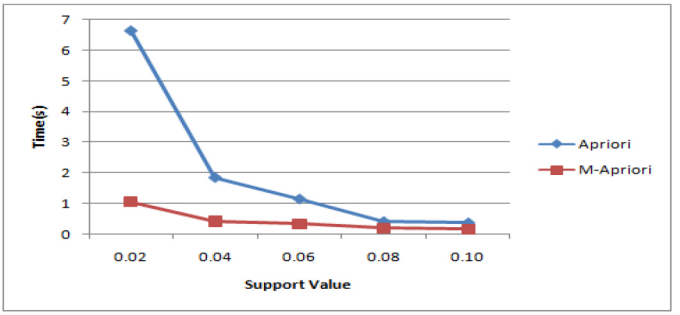

Figure 3

Time overwhelming judgement for different values of tiniest support

The initial experiment evaluates the time spent by both the innovative Apriori and our amended algorithm when applied to these five transaction groups. The results of this comparison are shown in Figure 2.

The next experiment equivalences the execution period of the innovative Apriori algorithm and our planned algorithm by applying a single group of dealings with varying tiniest support values. The results are presented in Figure 3.

1.4 *The investigation and assessment of the amended Apriori*

According to figure 2, the amended Apriori consistently consumes less time than the innovative Apriori across all transaction groups, with the difference becoming larger as the number of dealings increases.

Table 7 reveals that the amended Apriori reduces time consumption by 61.87% in the first transaction group (X1) and by 77.80% in the fifth

Table 7

Illustrates the rate at which the amended Apriori algorithm reduces processing time compared to the innovative Apriori algorithm, based on the sum of dealings.

T_c	Innovative Apriori (S1)	Amended Apriori (S#)	Stretch dipping amount(%)
X1	1.775	0.677	61.87%
X2	8.222	4.001	51.31%
X3	6.872	2.303	66.46%
X4	11.941	2.457	79.40%
X5	82.557	18.332	77.81%

transaction group (X5). The time reduction rate grows as the number of dealings surges. On average, the amended Apriori achieves a time reduction rate of 67.38%

As observed in figure 3, the stretch consumed by the amended Apriori for each value of minutest sustenance is consistently lower than that of the innovative Apriori. Moreover, the alteration becomes increasingly significant as the value of tiniest sustenance reductions.

Table 8 demonstrates that the amended Apriori reduces the time consumption by 83.09% compared to the innovative Apriori when the tiniest support is 0.021, and by 56.010% when the tiniest support is 0.10. As the tiniest support value increases, the reduction rate decreases accordingly. On average, the amended Apriori achieves a time reduction rate of 68.39%.

Table 8

The rate at which the amended Apriori reduces time compared to the innovative Apriori depends on the rate of the tiniest support.

Minm_ Sups	Innovative Apriori (S1)	Amended Apriori (S#)	Time dipping amount (%)
0.021	6.637	1.053	83.09%
0.041	1.854	0.423	77.24%
0.062	1.156	0.331	71.51%
0.085	0.425	0.200	53.08%
0.100	0.383	0.167	56.01%

6. Conclusion

This tabloid introduces an amended version of the Apriori algorithm aimed at dipping the while spent on scanning communications for contender product sets by minimizing the quantity of communications that need to be skimmed. As the value of *k* in the k-product conventional rises, the time efficiency breach among the amended Apriori and the innovative Apriori becomes more pronounced. Conversely, when the tiniest support value increases, the time efficiency gap decreases. The amended Apriori algorithm generates candidate support counts faster than the innovative Apriori, achieving a 67.38% reduction in processing time. These findings are supported by experimental results, As shown in Fig. 2, Fig. 3, Table 7, and Table 8.

References

[1] 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).

[2] S. Rao, R. Gupta, "Implementing Amended Algorithm Over APRIORI Data Mining Association Rule Algorithm", *International Journal of Computer Science And Technology*, pp. 489-493 (Mar. 2012).

[3] H. H. O. Nasereddin, "Stream data mining," *International Journal of Web Applications*, vol. 1, no. 4, pp. 183–190 (2009).

[4] F. Crespo and R. Weber, "A methodology for dynamic data mining based on fuzzy clustering," *Fuzzy Sets and Systems*, vol. 150, no. 2, pp. 267–284 (Mar. 2005).

[5] R. Srikant, "Fast algorithms for mining association rules and sequential patterns," UNIVERSITY OF WISCONSIN, 1996.

[6] J. Han, M. Kamber,"Data Mining: Concepts and Techniques*", Morgan Kaufmann Publishers*, Book (2000).

[7] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, p. 37 (1996).

[8] F. H. AL-Zawaidah, Y. H. Jbara, and A. L. Marwan, "An Amended Algorithm for Mining Association Rules in Large Databases," Vol. 1, No. 7, 311-316 (2011).

[9] T. C. Corporation, "Introduction to Data Miningand Knowledge Discovery*", Two Crows Corporation*, Book (1999).

[10] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of products in large databases," in *ACM SIGMOD Record*, vol. 22, pp. 207–216 (1993).

[11] M. Halkidi, "Quality assessment and uncertainty handling in data mining process," in Proc, EDBT Conference, Konstanz, Germany (2000).

Received November, 2024