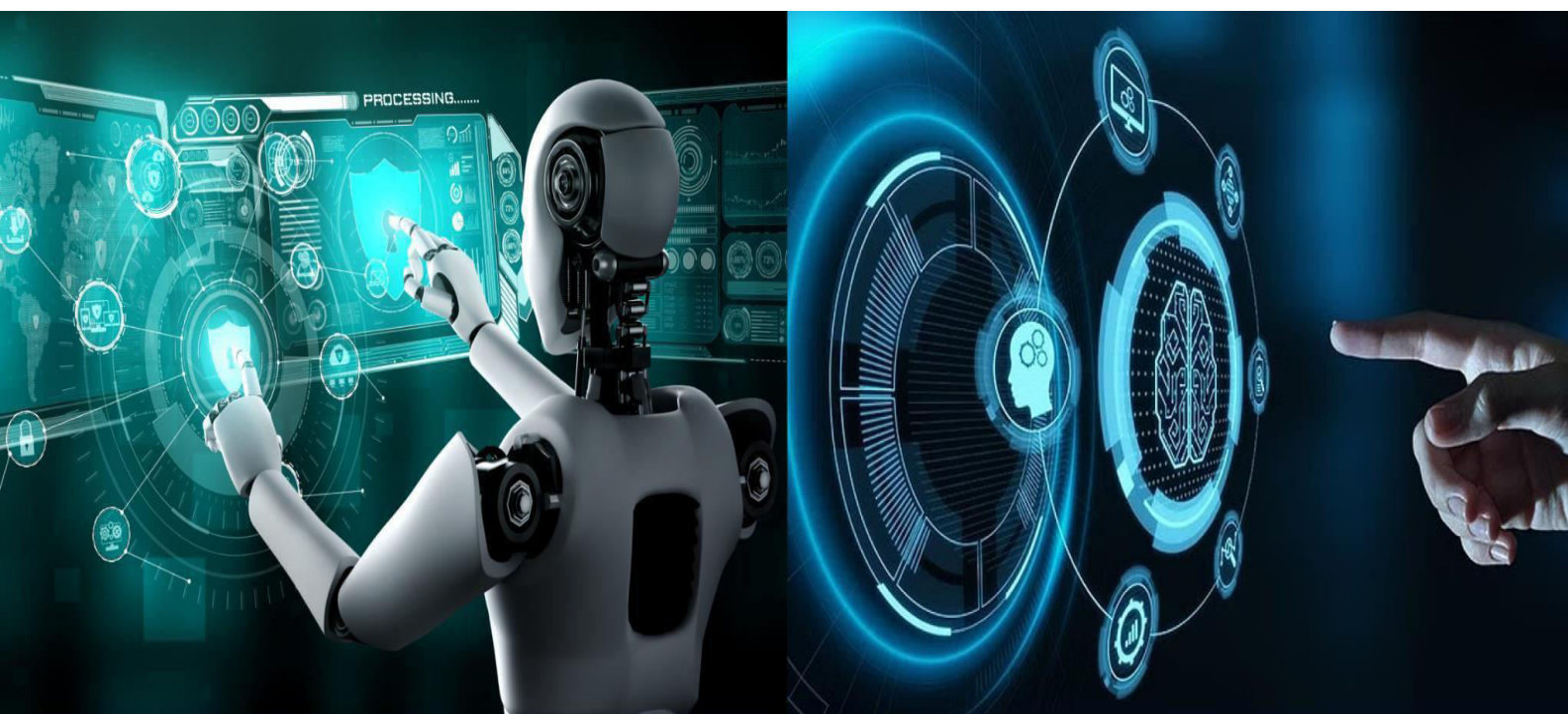


International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Implementation of Apriori Algorithm in Java

Prof. Sonal Rohidas Pawar

Assistant Professor, Department of Masters of Computer Application, CAYMET's Siddhant Institute of Computer Application, Pune, Maharashtra, India

ABSTRACT: Data mining is the process of discovering useful and hidden patterns from large datasets. Association Rule Mining is an important data mining technique, widely used in Market Basket Analysis. The Apriori algorithm is a popular and level-wise algorithm used to find frequent itemsets from a transaction database based on minimum support. It works using the downward closure property, where all subsets of a frequent itemset must also be frequent. The algorithm uses join and prune steps and follows a Breadth First Search approach to generate candidate itemsets of size 1, 2, 3, and so on. In Java, Apriori can be implemented using arrays and collections to store transactions and itemsets. A lattice structure is used to represent itemset levels, making pattern generation easy to understand.

KEYWORDS: *Apriori, Frequent item set, Support, Candidate item set, Time-consuming, Transaction Database, Association Rule Mining, Market Basket Analysis, Data Mining*

I. INTRODUCTION

The Apriori algorithm is array-based. It is also called Level Wise Algorithm. It relies on the Downward Closure Property. As the name suggests, the algorithm is a bottom-up search, moving upward level-wise in a lattice. In Apriori algorithm Join and Prune techniques are used. The BFS (Breadth First Search) mechanism is used in the Apriori algorithm. The Apriori algorithm generates various patterns containing 1 item, 2 items, 3 items, and so on. The Apriori algorithm scans the database multiple times for generating candidate sets. The Apriori algorithm requires large memory space due to a large number of candidate generations. The Apriori algorithm is the mostly well-known association rule algorithm. And it is used in mostly commercial products. In the apriori algorithm the large item set property is used. In the Apriori algorithm, the basic idea is to generate a candidate item set of a particular. It uses prior (a-prior) knowledge on frequent item set properties.

History:

In 1994 Rakesh Agrawal and Srikant introduced Apriori Algorithm.

In 1994, R. Agrawal and R. Srikant developed the Apriori method for identifying the most frequently occurring item sets in a dataset using the Boolean association rule.

Advantages of Apriori Algorithm:

1. Easy to understand:

The Apriori algorithm is simple and easy to learn, so it is widely used for teaching and exams.

2. Simple logic:

It uses a clear rule: *if an itemset is not frequent, its larger combinations are also not frequent.*

3. Well-known and widely used:

Apriori is one of the most popular algorithms for association rule mining.

4. Good for small datasets:

It works well when the database size is small or medium.

5. Clear step-by-step process:

The algorithm follows systematic steps like candidate generation and pruning.

Disadvantages of Apriori Algorithm:

1. Too many database scans:

Apriori scans the database many times, which increases processing time.

2. High time consumption:

It becomes slow when the dataset is very large.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

3. Large number of candidate sets:

It generates many candidate itemsets, which wastes memory.

4. Not suitable for big data:

Performance is poor for large and complex datasets.

5. Uses more memory:

Storing candidate itemsets requires extra memory.

II. LITERATURE REVIEW:

Association Rule Mining is a core area of data mining, and the Apriori algorithm is one of the earliest and most influential techniques in this domain. Agrawal and Srikant (1994) first introduced the Apriori algorithm to identify frequent itemsets and generate association rules from large transactional databases. Their work established the foundation for Market Basket Analysis by using the downward closure property, which significantly reduces the search space by pruning infrequent itemsets early. This approach made Apriori practical for real-world applications despite multiple database scans.

Several researchers have studied and implemented the Apriori algorithm using programming languages such as Java due to its platform independence and strong support for data structures. Java-based implementations commonly use arrays, hash tables, and collections to store transaction data and candidate itemsets efficiently. Studies highlight that although Apriori is easy to understand and implement, it is time-consuming and memory-intensive because of repeated database scans and large candidate generation. To address these limitations, many improvements and optimized versions have been proposed, yet the standard Apriori algorithm remains widely used for educational purposes and as a baseline for comparing advanced association rule mining techniques.

III. METHODOLOGY APPROACH

The methodology adopted for implementing the Apriori algorithm in Java follows a systematic and level-wise approach. Initially, a transaction database is created where each transaction contains a set of items. A minimum support threshold is defined to identify frequent itemsets. In the first step, the algorithm scans the database to generate frequent 1-itemsets by counting the occurrence of each item.

Next, candidate itemsets of higher order (2-itemsets, 3-itemsets, and so on) are generated using the join operation, and infrequent itemsets are eliminated using the prune technique based on the downward closure property. The database is scanned repeatedly to calculate support values for each candidate set. This process continues until no further frequent itemsets can be generated. Finally, association rules are derived from the frequent itemsets. Java data structures such as arrays, lists, and hash maps are used to efficiently store transactions, candidate sets, and support counts.

Assumptions:

All subsets of frequent item sets must be frequent (Apriori Property).

If an item set is infrequent, all its supersets will be infrequent. And thus can be ignored (antimonotone property).

Note: Antimonotone: Given $X \subseteq Y$, if $c(X)$ is not true then $c(Y)$ is not true, i.e., $\neg c(X) \Rightarrow \neg c(Y)$

The frequency counts in apriori algorithm –A minimum threshold is set on the expert's advice or user understanding.

Two Principles of Apriori Algorithm:

1. Join Step:

The Join step generates (k+1) item sets, from K-Item sets by joining each item for itself.

2. Prune Step:

The Prune step scans the count of each item in the database. If the candidate item does not meet minimum support, then it is considered infrequent and thus it is removed.

In the Prune stage, the size of the candidate item sets is reduced.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Components of the Apriori algorithm:

There are three major components of the Apriori algorithm which are as follows.

1. Support:
2. Confidence:
3. Lift:

Apriori accuracy:

How to balance support, confidence, and lift a rule?

Apriori delicacy how to balance support, confidence, and lift a rule?

This substantially gives us three criteria to illustrate

- i) **Support:** The number of times, or the chance, that the products attend.
- ii) **Confidence:** Confidence is the conditional probability.
- iii) **Lift:** lift measured by the strength of association.

Apriori algorithm Steps:

Apriori is a pretty straightforward algorithm that performs the following sequence of calculations:

1. Calculate support for item sets of size 1.
2. Apply the minimum support threshold and prune item sets that do not meet the threshold.
3. Move on to item sets of size 2 and repeat steps one and two.
4. Continue the same process until no additional item sets satisfying the minimum threshold can be found.

Example:

Consider A threshold support of 50%.

Transaction Id	Item Sets
T1	Milk,Bread,Butter,Sugar
T2	Milk,Bread,Butter
T3	Milk,Bread,Sugar
T4	Bread, Sugar
T5	Bread,Butter,Sugar
T6	Milk,Bread,Butter

Solution:

Support threshold=50% => $0.5 * 6 = 3$ => min_sup=3

Step 1: Create the frequency table of all items that occur in all the transactions, Prune the frequency table to include only those items having a threshold Support Count greater than or equal to 50%.

Item	Frequency
Milk	4
Bread	6
Butter	4
Sugar	4

Step 2: Make 2 items in 1 Pair, and calculate the frequency count from the transaction table.

Item set	Frequency
Milk, Bread	4
Milk, Butter	3
Milk, Sugar	2
Bread, Butter	4
Bread, Sugar	4
Butter, Sugar	2



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Apply the same threshold, and we finally get Milk-Bread, Bread-Butter, and Bread-Sugar

Step 3: Now again take 3 items from the above item sets, we have Milk-Butter-Sugar, Bread-Butter-Sugar, and Milk-Bread-Sugar.

Repeat the previous 2 Steps to calculate the frequency and apply the threshold to eliminate the non-frequent item sets.

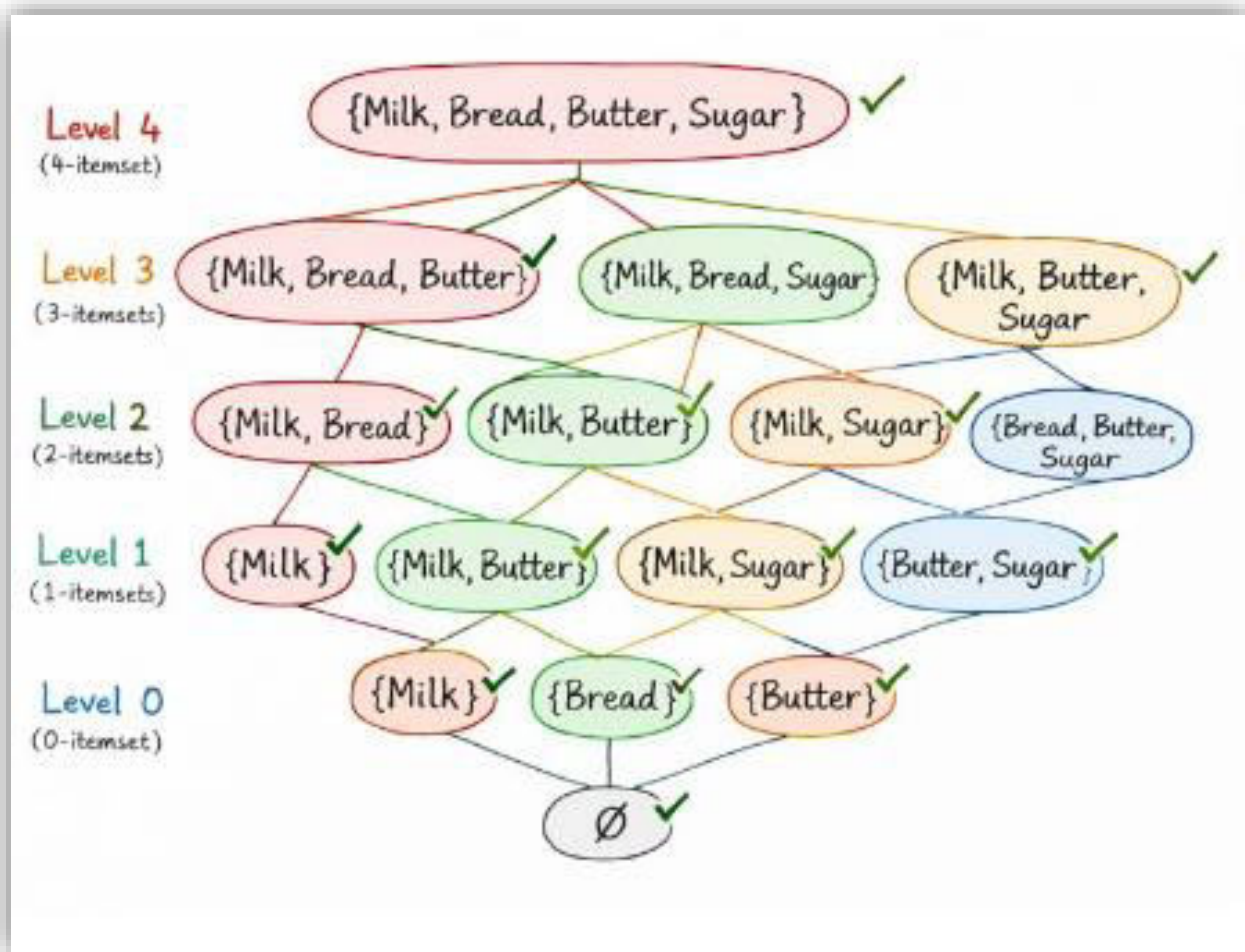
Item set	Frequency
Milk-Butter-Sugar	3
Bread-Butter-Sugar	2
Milk-Bread-Sugar	2

Step 4:

After pruning items from transactions we only get only one item set left, Milk-Bread-Butter, which support count is: 3

In real-time we will have several transactions to go through to get these results. There will be multiple combinations that go on to arrive at the best results or association o items.

Lattice of Subset:

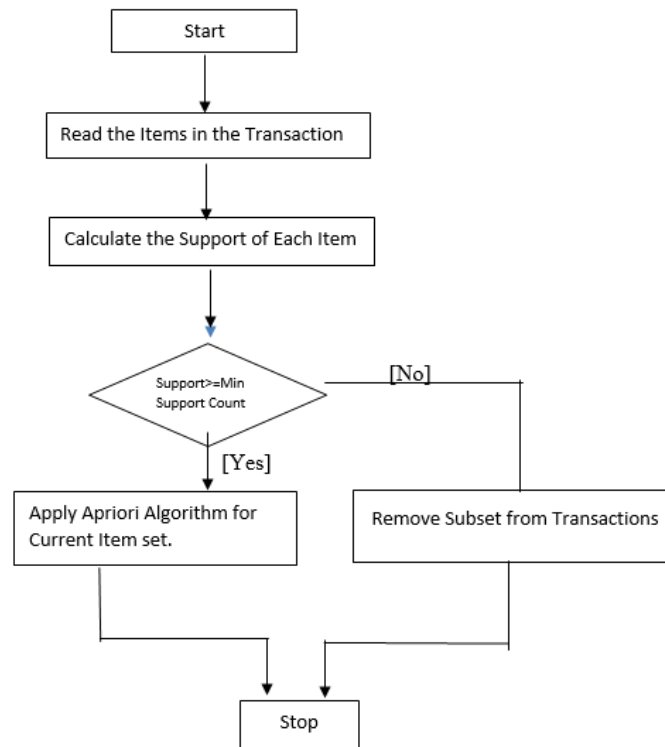




International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Flowchart:



Applications of Apriori Algorithm:

Apriori Algorithm has picked up a pace in recent years and is used in different industries for data mining and handling. Some fields where Apriori is used:

1. Medical

Hospitals and medicals every day need to retrieve more data which is stored in the past for existing patients. The Apriori algorithm helps to manage the database of patients without mixing it with other patients.

2. Education

The apriori algorithm is used in educational organizations to keep and monitor students' data about age, gender, traits, parent details, etc.

3. Forestry

The apriori algorithm is used in forests, for storing the information of every earth's unique flora and fauna of the field to analyse and store.

4. New Tech Firms

Tech firms use the Apriori algorithm to maintain the record of various items of products that are purchased by various customers for suggested structures.

5. Mobile Commerce

Big data can help mobile e-commerce companies to deliver an easy, convenient, and personalized shopping experience. With the Apriori algorithm, the real-time product recommendation accuracy increases, this creates an excellent customer experience and increases sales for the company.

How can we improve the Apriori Algorithm's efficiency?

Many methods are available for improving the efficiency of the algorithm.

- Hash-Based Technique:** This method uses a hash-based structure called a hash table for generating the k-item sets and their corresponding count. It uses a hash function for generating the table.
- Transaction Reduction:** This method reduces the number of transactions scanned in iterations. The transactions which do not contain frequent items are marked or removed.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

3. **Partitioning:** This method requires only two database scans to mine the frequent item sets. It says that for any item set to be potentially frequent in the database, it should be frequent in at least one of the partitions of the database.
4. **Sampling:** This method picks a random sample S from Database D and then searches for a frequent item set in S. It may be possible to lose a global frequent item set. This can be reduced by lowering the min_sup.
5. **Dynamic Item Set Counting:** This technique can add new candidate item sets at any marked start point of the database during the scanning of the database.

Implementation of Apriori Algorithm in Java:

Program Code:

```
import java.io.*;
import java.util.*;

public class AprioriExample {

    static int MIN_SUPPORT = 3; // 50% of 6 transactions

    public static void main(String[] args) {

        List<Set<String>> transactions = new ArrayList<>();

        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread", "Butter", "Sugar"))); // T1
        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread", "Butter"))); // T2
        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread", "Sugar"))); // T3
        transactions.add(new HashSet<>(Arrays.asList("Bread", "Sugar"))); // T4
        transactions.add(new HashSet<>(Arrays.asList("Bread", "Butter", "Sugar"))); // T5
        transactions.add(new HashSet<>(Arrays.asList("Milk", "Bread", "Butter"))); // T6

        // Step 1: Generate L1 (Frequent 1-itemsets)
        Map<Set<String>, Integer> frequentItemsets = getFrequentItemsets(transactions);

        int k = 1;
        while (!frequentItemsets.isEmpty()) {
            System.out.println("\nFrequent " + k + "-itemsets:");
            for (Map.Entry<Set<String>, Integer> entry : frequentItemsets.entrySet()) {
                System.out.println(entry.getKey() + " -> Support: " + entry.getValue());
            }

            // Generate candidate itemsets
            Set<Set<String>> candidates = generateCandidates(frequentItemsets.keySet(), k + 1);

            frequentItemsets = getSupportCount(transactions, candidates);
            k++;
        }

        // Count support and return frequent itemsets
        private static Map<Set<String>, Integer> getFrequentItemsets(List<Set<String>> transactions) {
            Map<Set<String>, Integer> countMap = new HashMap<>();

            for (Set<String> transaction : transactions) {
                for (String item : transaction) {
                    Set<String> itemset = new HashSet<>();
                    itemset.add(item);
                    countMap.put(itemset, countMap.getOrDefault(itemset, 0) + 1);
                }
            }
        }
    }
}
```



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

```

}

return filterBySupport(countMap);
}

// Generate candidates
private static Set<Set<String>> generateCandidates(Set<Set<String>> prevItemsets, int k) {
    Set<Set<String>> candidates = new HashSet<>();

    for (Set<String> set1 : prevItemsets) {
        for (Set<String> set2 : prevItemsets) {
            Set<String> union = new HashSet<>(set1);
            union.addAll(set2);

            if (union.size() == k) {
                candidates.add(union);
            }
        }
    }
    return candidates;
}

// Count support of candidates
private static Map<Set<String>, Integer> getSupportCount(
    List<Set<String>> transactions,
    Set<Set<String>> candidates) {

    Map<Set<String>, Integer> countMap = new HashMap<>();

    for (Set<String> candidate : candidates) {
        for (Set<String> transaction : transactions) {
            if (transaction.containsAll(candidate)) {
                countMap.put(candidate, countMap.getOrDefault(candidate, 0) + 1);
            }
        }
    }
    return filterBySupport(countMap);
}

// Filter itemsets by minimum support
private static Map<Set<String>, Integer> filterBySupport(Map<Set<String>, Integer> countMap) {
    Map<Set<String>, Integer> frequentItemsets = new HashMap<>();

    for (Map.Entry<Set<String>, Integer> entry : countMap.entrySet()) {
        if (entry.getValue() >= MIN_SUPPORT) {
            frequentItemsets.put(entry.getKey(), entry.getValue());
        }
    }
    return frequentItemsets;
}
}

```

Output:

Frequent 1-itemsets:
 [Butter] -> Support: 4
 [Sugar] -> Support: 4



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

[Milk] -> Support: 4
[Bread] -> Support: 6

Frequent 2-itemsets:

[Butter, Milk] -> Support: 3
[Sugar, Bread] -> Support: 4
[Butter, Bread] -> Support: 4
[Milk, Bread] -> Support: 4

Frequent 3-itemsets:

[Butter, Milk, Bread] -> Support: 3

==== Code Execution Successful ====

IV. RESULT & DISCUSSION

The implementation of the Apriori algorithm in Java successfully generated frequent itemsets from the given transaction database based on the defined minimum support threshold. The algorithm produced frequent 1-itemsets, followed by 2-itemsets and higher-level itemsets, demonstrating its level-wise and bottom-up nature. The lattice representation clearly showed the relationship between itemsets at different levels, making the frequent pattern generation process easy to understand and analyze. The results indicate that Apriori is effective in identifying meaningful associations among items, which is useful in applications such as Market Basket Analysis. However, during execution, it was observed that the algorithm required multiple database scans and generated a large number of candidate itemsets, leading to increased execution time and memory usage. Despite these limitations, the Apriori algorithm remains a reliable and widely used technique for association rule mining, especially for small to medium-sized datasets and educational implementations in Java.

V. CONCLUSION

It reduces the size of the item sets in the database significantly providing a good performance. So, data mining helps consumers and industries better in the decision-making process. The Apriori algorithm is a fundamental and widely used technique in data mining for discovering frequent itemsets and association rules from transactional databases. The Java-based implementation of the Apriori algorithm demonstrates its simple, level-wise working mechanism using join and prune operations based on the downward closure property. The results show that Apriori effectively identifies meaningful patterns and relationships among items, making it useful for applications such as Market Basket Analysis. Although the algorithm is time-consuming and requires multiple database scans with high memory usage due to large candidate generation, it remains valuable for learning and baseline analysis. Overall, the implementation highlights both the effectiveness and limitations of the Apriori algorithm in practical data mining applications. The Apriori algorithm is a systematic algorithm that scans the database only once.

REFERENCES

- [1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Int. Conf. Very Large Data Bases (VLDB), Santiago, Chile, 1994, pp. 487–499.
- [2] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proc. ACM SIGMOD, Washington, DC, USA, 1993.
- [3] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed. Morgan Kaufmann, 2012.
- [4] K. Rajalakshmi and S. S. Dhenakaran, *Apriori Algorithm – A Review*, International Journal of Computer Science and Engineering, Vol. 1, No. 4, pp. 161–164, 2009.
- [5] M. H. Dunham, *Data Mining: Introductory and Advanced Topics*, Pearson Education, 2006.
- [6] S. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, Pearson Education, 1st Edition, 2006.
- [7] A. K. Jain and B. Satish, *Implementation of Apriori Algorithm on Retail Data*, International Journal of Computer Applications, Vol. 67, No. 25, 2013.



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING



9940 572 462



6381 907 438



ijircce@gmail.com



www.ijircce.com

Scan to save the contact details