# CONCEPT ADAPTIVE MAXIMAL FREQUENT ITEMSET ALGORITHM FOR ASSOCIATION RULES

**[1]R.SMEETA MARY, [2]Dr.K.PERUMAL**

[1]Assistant Professor, Department of Computer Applications, Fatima College, Madurai, Tamil Nadu, India

[2]Professor, Department of Computer Applications, Madurai Kamaraj University, Madurai, Tamil Nadu, India

E-mail: [1]smeetamaryr@gmail.com, [2]perumalmala@gmail.com

**ABSTRACT**

There are several mining algorithms of association rules. One of the algorithms is very fast maximal frequent itemset that is used to extract frequent itemsets from large database and getting the association rule for discovering the knowledge. Based on this algorithm, this paper indicates the limitation of the very fast maximal frequent itemset of wasting time by substituting the numbers for the whole database searching on the frequent itemsets, and presents an improvement on concept adaptive maximal frequent itemset by reducing the wasted time by substituting the prime values. The paper shows by experimental results with several groups of transactions, and with several values that applied on the very fast maximal frequent itemset and our implemented concept adaptive maximal frequent itemset that our improved algorithm reduces the time consumed in comparison with the very fast maximal frequent itemset, and makes the algorithm more efficient and less time consuming. For implementation of the work patient database is taken and the patient records are experimented and the final best is identified with quick response time and least error rate. A typical confusion matrix is furthermore displayed for quick check. The concept adaptive maximal frequent itemset algorithmic discussion of the heart disease dataset from kaggle, cardiovascularheart2019dataset, repository of large datasets. The Best results are achieved by using R tool. R is an effective data handling and storage facility especially for data mining. By adding the true positive and true negative values, then dividing the total number of possibilities the accuracy is calculated for concept adaptive maximal frequent itemsets.

**KEYWORDS**

VFMFI, CMFIA, Frequent Itemsets, Time Consuming.

## 1. INTRODUCTION

Data mining is a process that is utilized to extract useful information from an enormous arrangement of raw information. Utilizing one or more software the data patterns can be analyzed from huge clumps of information. Data mining has various applications in different fields like research and science. Data mining can learn more and develop effective strategies related to number of functions and resources in a more optimal and insightful manner. Data mining find out more and

create powerful methodologies identified with number of capacities and warehousing in a more ideal and wise way. It uses sophisticated mathematical algorithms for finding the data evaluating the probability of likelihood of future functions. Data mining is otherwise called Knowledge Discovery in Data (KDD) and it includes an integration of techniques from many disciplines such as machine learning, neural networks, database technology, information retrieval and statistics etc. There are several steps in KDD that are executed to extract patterns to user, such as data selection, data cleaning, pre-processing data transformation, data mining and pattern evaluation [1].

The chief segments of any data mining system are information source, data mining engine, data warehouse server, graphical UI, pattern evaluation module and knowledge base. Data are stored in databases or data warehouses. Data warehouses may contain one or more databases, text records, bookkeeping pages or other kinds of information repositories. Now it is ready to be processed. Here the server is responsible for retrieving the relevant data based on the data mining request of the user. For performing data mining tasks data mining engine has number of modules such as association, characterization, classification, clustering, time-series analysis, prediction etc. and data mining engine focus the search towards interesting patterns. The graphical user interface module imparts between the user and the data mining framework. It helps the user use the system easily and efficiently without knowing the real complexity behind the process. Finally knowledge base might be useful for guiding the search or evaluating the interesting result patterns.

## 2. ASSOCIATION RULE MINING

Association Mining is one of the main data mining's functionalities and it is the most mainstream method has been concentrated by researchers. The researchers use the association rules for mining the database of sales transactions between items. The main advantages of the rules are finding out the unknown relationship, producing results that help to perform decision making and forecasting the result. The association rules are divided into two phases: finding out the frequent item and generating various association rules. In first phase finding the frequent itemset, every set of items that is itemset if they appear greater than the minimum support threshold. In the second phase it can create numerous rules from one itemset. Suppose n is the number of items then the number of the rules may be $n^2$-1. Support and confidence are characterized by the user which represents the requirement of the guidelines. So the support and confidence thresholds must be applied to all the rules to prune the rules where the value must be less than thresholds value. More applications are being motivated by association rules such as health, retail, communication and banking etc.

## 3. RELATED WORK

Mining the frequent itemsets is a significant phase in association mining which finds the frequent itemsets in transactions database. It is the main tasks of data mining that try to find interesting patterns from datasets, such as association, correlation, classifier, clustering etc [2].

Association Mining is one of the most significant functionalities of data mining and it is the most accepted

technique has been studied by researchers. Extracting association rules is that the core of knowledge mining [3].Various algorithms are projected to find frequent itemsets, such as AIS, MFI Miner, Pincer-Search, GenMax, One Pass, MaxMiner, Mafia, FP-growth Apriori.

The AIS algorithm was the first algorithm projected by Agrawal, Imielinski and Swami for mining association rule [4]. AIS algorithm scans the databases many times to get the frequent itemsets. "The support count of each individual item was gathered during the first pass over the database. Depending on the threshold of support count the items which have the count less than its minimum value are removed from the list of items. Next is the Candidate 2-itemsets which are generated by extending frequent 1-itemsets with others transactions. In the second pass, the support count of those candidate 2-itemsets are obtained and checked against the support threshold. Similarly this process is repeated for the entire item in the same transaction. Until any one of them becomes empty the generation of candidate itemsets and frequent itemsets are generated. Since all the candidate itemsets and frequent itemsets are assumed to be stored in the main memory, memory management is not enough. In [5], An MFIMiner algorithm uses breadth-first search with resourceful pruning method that expertly mines both long and short maximal frequent itemsets. The Pincer-Search [6] algorithm uses horizontal data format. Like Apriori it not only constructs the candidates in a bottom-up manner, but at the same time it starts a top-down search by maintaining a candidate set of maximal patterns. This can help in reducing the amount of database scans, by eliminating non-maximal sets early. The maximal candidate set may be a superset of the maximal patterns, and generally, the overhead of maintaining it are often very high. In contrast GenMax maintains only the present known maximal patterns for pruning. In [7], GenMax algorithm, which uses backtrack search method for mining maximal frequent itemset. It uses various optimizations methods to prune the search space. Progressive focusing technique is used to perform maximality checking, and different set of propagation to perform fast frequency computation. It is found that GenMax to be a highly efficient method to mine the exact set of maximal patterns. In [8] a one-pass algorithm called Data Stream Mining for Maximal Frequent Itemsets which gets the set of all maximal frequent itemsets over data streams. A data structure called summary frequent itemset forest is developed for incrementing and maintaining the information of obtaining maximal frequent itemsets in the stream. MaxMiner is an efficient algorithm for finding out the maximal elements. It narrows the search using some efficient pruning techniques. It mainly focus on breadth-first traversal of the search space and reduces scanning of the database by applying pruning strategy, i.e., if a node is determined to be frequent, then there is no possibility to process that node. To increase the effectiveness of superset-frequency pruning it uses item (re)ordering heuristic. Since MaxMiner utilizes horizontal database format, it can play out similar number of passes over a database as Apriori does [9]. Mafia [10] [11] is one of the recent methods for mining the Maximal Frequent Itemset. Pruning strategies are used to remove non-maximal sets. Like the MaxMiner look-ahead pruning is used in MaxMiner and next is to check whether new set is considered by an existing maximal set. "FP-Growth Algorithm is the next level of frequent itemset. The main of this algorithm was to remove the difficulties of the Apriori algorithm in generating and testing candidate sets and thus it got the name as

frequent pattern tree or FP-tree. FP-Growth uses both the vertical and horizontal database to store the values of the database in main memory. Rather than storing ID for every transaction in the database, it actually stores the database in a tree structure and linked list connects all the items that have its own item [12]. In [13] [14], Apriori algorithm, the name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemsets properties. Apriori uses an iterative approach known as level wise search, where k itemsets are used to explore k+1 itemsets. First the frequent 1 itemsets is found, this is denoted by L1, frequent2 itemset L2 and so on. Two step processes is used to find LK, one is the join step and other is prune step to find the frequent itemset.

## 4. VFMFI ALGORITHM

Very fast maximal frequent itemset is too easy to execute and it's also quite easy to find frequent itemset. The algorithm makes examines the database to find maximal frequent itemset from n-itemsets. The target transaction is scanned horizontally in this case, and the total_count of the column is calculated. The total_count is then sorted in decreasing order, with normal numbers substituted. Now for each data item the regular numbers that is acquired is substituted and finally the row_total of each transaction determined. From the row_total the numbers are deducted therefore on the off chance that the rowtotal isn't zero, at that point the column name will be added as the maximal frequent item set. This procedure continues until one of the row total values reaches zero.

## 5. LIMITATIONS OF VFMFI ALGORITHM

In some cases, the VFMFI method may be poor since it is not straightforward and obvious. The algorithm's primary flaw is that it wastes time when dealing with a large number of transactions. If the database's transactions include 100 column_name, then 100 regular numbers must be substituted each time. For all of them (e.g. age, sex, bp....chol), the regular number must be replaced. As a result, for a large number of transactions scanning takes place many times repeatedly for finding maximal frequent itemset.

As per the total_count only the regular numbers are substituted, if assume the total_count is same for at least two number of transaction then, it causes an issue that the regular numbers are substituted in order. When memory capacity is limited and there are a large number of transactions, the procedure is highly slow and inefficient due to the regular numbers. The recommended technique in this study will minimize the time spent searching for the maximal frequent itemset in the transactions.

## 6. CONCEPT ADAPTIVE MAXIMAL FREQUENT ITEMSET ALGORITHM

This section will address the Concept Adaptive Maximal Frequent Itemset ideas, Concept Adaptive Maximal Frequent Itemset Algorithm, an example of the Concept Adaptive Maximal

Frequent Itemset Algorithm, the analysis and evaluation of the Concept Adaptive Maximal Frequent Itemset Algorithm and the experiments [15].

### 6.1. The Concept Adaptive Maximal Frequent Itemset ideas

In the process of Concept Adaptive Maximal Frequent Itemset Algorithm, the following steps are needed:

Step 1: collect and store the entire preprocessed dataset.

Step 2: Count the individual itemcount in horizontal manner in respect to column name.

Step 3: Sort the Individual items in the colList in descending order.

Step 4: Assign consecutive prime numbers for all the column items.

Step 5: Apply prime numbers to all the individual items and calculate the rowTotalList.

Step 6: From each rowTotalList value subtract the prime number and store it in resultList.

Step 7: If any of the transaction in the resultList =0 then stop the process else repeat step5 and 6.

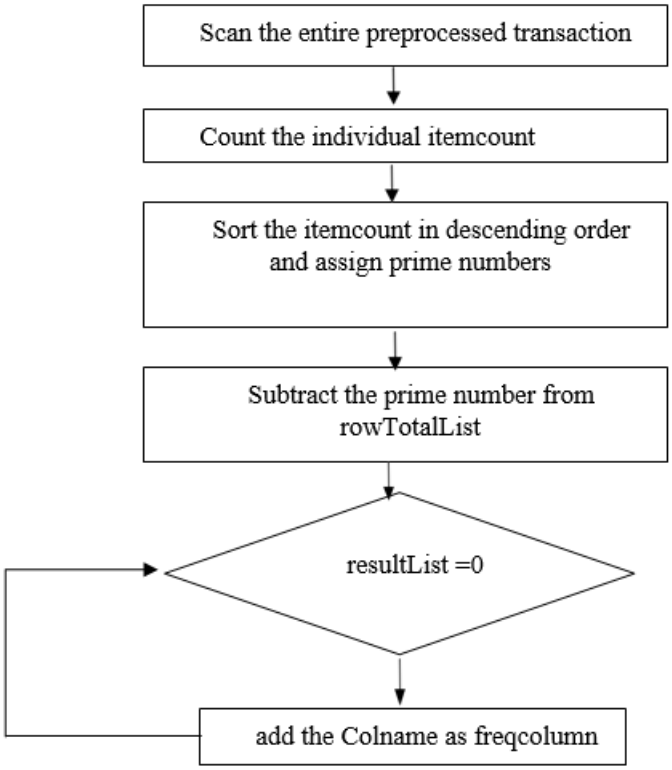


*Figure 1: Steps for generating maximal frequent itemset*

In our proposed approach, we enhance the Very Fast Maximal Frequent Item Set to reduce the time consuming. Because of the assigning of prime number the speed is increased and time consuming is also reduced.

### 6.2. The Concept Adaptive Maximal Frequent Itemset Algorithm

The improvement of algorithm can be described as follows:

Algorithm –CMFIA

Input: dataset D, support S

Output: Maximal Frequent Itemsets MFI

CMFIA (Dataset D)

BEGIN

1. Read and store the entire preprocessed dataset.
2. Traverse horizontally through all the elements in the list and generate the sum of each item in respect to their column name, colList
3. Reorder the colList items in descending order.
4. Generate prime numbers for the each column items.

PrimeList = GetPrimeNumbers(colList);

5. If PrimeList is null: continue;
6. Substitute the prime numbers from the PrimeList to the respective colList items.

```
rowTotalList = []
for i = 0,…,len(colList) {
        initialize sum as 0
                for j=0,…,len(colList(i))
                        sum = sum + colList(i)[j];
                rowTotalList.add(sum)
```

7. For primenumber ε PrimeList // PrimeList-> set of prime numbers

```
    get Colname of primenumber
    get the sum of Colname from colList
    if sum is unique
       resultList= from each rowTotalList value subtract the primenumber
       CurColname.add(Colname)
       Adding the count of Colname as freqcolumn//for checking the support
       if resultList.contains(0)
         Add to FI(CurColname)
         break;
    else
       goto step 7
    END
```

**GetPrimeNumbers**

Input: colList

Output: A total number of Prime Numbers according to the count of colList items

GeneratePrimeNumbers(colList)

**BEGIN**

count = len(colList); //count will contain total number of columns

Initialize PrimeList as null

Input N and count

While N is smaller than count

Initialize I to 2

While I is smaller than N

If N is divisible by I

skip loop

Increment the value of I

If N is equal to I

PrimeList.add(N)

Increment the value of N

return PrimeList

**END**

### 6.3. An example of the Concept Adaptive Maximal Frequent Itemset

CMFIA is an advanced approach for determining the most frequent item set. With a single database scan, it may collect significant knowledge from the stream contents. CMFIA's compress feature will indicate that it uses less memory than previous algorithms. The proposed method is based on the numbers being substituted into each subset.

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 18393 | 2 | 168 | 62 | 110 | 80 | 1 | 1 | 0 | 0 |
| 1 | 20228 | 1 | 156 | 85 | 140 | 90 | 3 | 1 | 0 | 0 |
| 2 | 18857 | 1 | 165 | 64 | 130 | 70 | 3 | 1 | 0 | 0 |
| 3 | 17623 | 2 | 169 | 82 | 150 | 100 | 1 | 1 | 0 | 0 |
| 4 | 17474 | 1 | 156 | 56 | 100 | 60 | 1 | 1 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 69997 | 22601 | 1 | 158 | 126 | 140 | 90 | 2 | 2 | 0 | 0 |
| 69998 | 19066 | 2 | 183 | 105 | 180 | 90 | 3 | 1 | 0 | 1 |
| 69999 | 22431 | 1 | 163 | 72 | 135 | 80 | 1 | 2 | 0 | 0 |

*Figure 2: The original dataset of cardiovascular*

At times it might acknowledge and the analysis may succeed, but the result provided may not be correct, or the result generated will be incorrect. As a result, R is used to complete the preprocessing for the

CMFIA. R is used to answer a series of queries, including data cleaning, data transformation, data integration, data normalization, missing data imputation, and noise detection.

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | NaN | NaN | 1.0 | NaN | 1.0 | NaN | 1 | NaN | NaN |
| 1 | 1.0 | 1.0 | NaN | 1.0 | 1.0 | NaN | 1.0 | 1 | NaN | NaN |
| 2 | 1.0 | 1.0 | NaN | 1.0 | 1.0 | 1.0 | 1.0 | 1 | NaN | NaN |
| 3 | 1.0 | NaN | NaN | 1.0 | 1.0 | NaN | NaN | 1 | NaN | NaN |
| 4 | 1.0 | 1.0 | NaN | NaN | NaN | 1.0 | NaN | 1 | NaN | NaN |
| .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |
| 69998 | 1.0 | NAN | NAN | 1.0 | 1.0 | NAN | 1.0 | 1 | NAN | 1 |
| 69999 | 1.0 | 1.0 | NaN | 1.0 | 1.0 | 1.0 | NaN | 1 | NaN | NaN |

*Figure 3: the preprocessed data using R*

### 6.4 Frequent Itemsets Deduction using CMFIA

CMFIA is an advanced approach for determining the most frequent item set. With a single database scan, it may collect significant knowledge from the stream contents. CMFIA's compress feature will indicate that it uses less memory than previous algorithms. The suggested technique is based on the prime numbers being substituted to each subset.

| DB | age | gender | height | weight | ... | cholesterol | gluc | smoke | alco |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | 0.0 | 0.0 | 1.0 | ... | 0.0 | 1 | 0.0 | 0.0 |
| 1 | 1.0 | 1.0 | 0.0 | 1.0 | ... | 1.0 | 1 | 0.0 | 0.0 |
| 2 | 1.0 | 1.0 | 0.0 | 1.0 | ... | 1.0 | 1 | 0.0 | 0.0 |
| 3 | 1.0 | 0.0 | 0.0 | 1.0 | ... | 0.0 | 1 | 0.0 | 0.0 |
| 4 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 1 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 69995 | 1.0 | 0.0 | 0.0 | 1.0 | ... | 0.0 | 1 | 1.0 | 0.0 |
| 69996 | 1.0 | 1.0 | 0.0 | 1.0 | ... | 1.0 | 1 | 0.0 | 0.0 |
| 69997 | 1.0 | 0.0 | 0.0 | 1.0 | ... | 1.0 | 1 | 0.0 | 1.0 |
| 69998 | 1.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 1 | 0.0 | 0.0 |
| 69999 | 1.0 | 1.0 | 0.0 | 1.0 | ... | 1.0 | 1 | 0.0 | 0.0 |

*Figure 4: the original dataset of Concept Adaptive Maximal Frequent Itemset*

Finding the maximum frequent set (or MFS), which is the set of all maximal frequent itemsets, is fundamentally a hypothesis search space search issue (a lattice of subsets). The search for the maximal frequent item set using CMFIA can be done from 1-itemsets to n-itemsets.

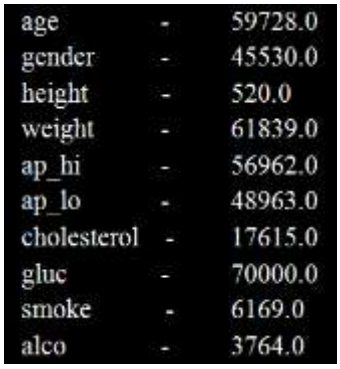


*Figure 5: Frequency of individual items in the dataset*

Finding the sum of all individual column items in the dataset seems to be the most fundamental step in analyzing the preparation phase. The algorithm will begin to work after the total of the column items has been computed.

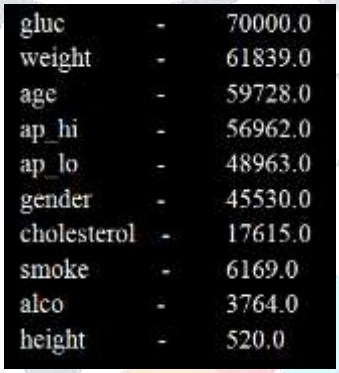


*Figure 6: Frequency of individual items in descending order*

As indicated by the recurrence the items are arranged in descending order. To produce frequent item sets, an iterative process is used to evaluate the specific length of item collection in the provided database.

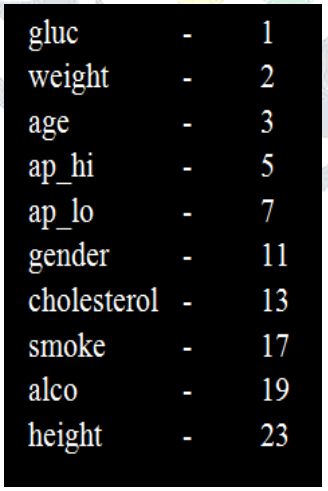


*Figure 7: Substitution of prime values to the itemset*

By utilising prime numbers to represent objects in a transaction where each item is allocated a unique prime number. Each transaction is represented by the matching prime numbers of individual things in the transaction.

| DB | age | gender | height | weight | ... | gluc | smoke | alco | Total |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 3.0 | NaN | NaN | 2.0 | ... | 1 | NaN | NaN | 13.0 |
| 1 | 3.0 | 11.0 | NaN | 2.0 | ... | 1 | NaN | NaN | 35.0 |
| 2 | 3.0 | 11.0 | NaN | 2.0 | ... | 1 | NaN | NaN | 42.0 |
| 3 | 3.0 | NaN | NaN | 2.0 | ... | 1 | NaN | NaN | 11.0 |
| 4 | 3.0 | 11.0 | NaN | NaN | ... | 1 | NaN | NaN | 22.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 69995 | 3.0 | NaN | NaN | 2.0 | ... | 1 | 17.0 | NaN | 35.0 |
| 69996 | 3.0 | 11.0 | NaN | 2.0 | ... | 1 | NaN | NaN | 35.0 |
| 69997 | 3.0 | NaN | NaN | 2.0 | ... | 1 | NaN | 19.0 | 43.0 |
| 69998 | 3.0 | 11.0 | NaN | 2.0 | ... | 1 | NaN | NaN | 29.0 |
| 69999 | 3.0 | 11.0 | NaN | 2.0 | ... | 1 | NaN | NaN | 42.0 |

*Figure 8: Total count of all transaction*

The allocated prime values are substituted in the data of the transaction. The horizontal pass is done to find the total of the each and every transaction that has been completed.

gluc {1: 1}
After sub

| | |
|---|---|
| 0 | 12.0 |
| 1 | 34.0 |
| 2 | 41.0 |
| 3 | 10.0 |
| 4 | 21.0 |
| ... | |
| 69995 | 34.0 |
| 69996 | 34.0 |
| 69997 | 42.0 |
| 69998 | 28.0 |
| 69999 | 41.0 |

*Figure 9: Subtracting the prime vales from the total count*

CMFIA starts working here. This approach starts with the single itemset with the least prime value and decreases the prime value of the transaction by its value in every pass

weight {1: 2}
After sub

| | |
|---|---|
| 0 | 10.0 |
| 1 | 32.0 |
| 2 | 39.0 |
| 3 | 8.0 |
| 4 | 19.0 |
| ... | |
| 69995 | 32.0 |
| 69996 | 32.0 |
| 69997 | 40.0 |
| 69998 | 26.0 |
| 69999 | 39.0 |

*. Figure 10: Item names are added*

On the off chance that guesses any of the items is zero, at that point the activity is halted. When a k-

itemset is determined then, all k items will be examined in the next passes.


*Figure 11: The result of the CMFIA*

Thus the result is obtained. Out of 70000 transactions items are considered as the input for the CMFIA. In this we count the number of scanned transactions to get (1, 2, 3)-itemset whenever the k of k-itemset increase, the gap between our CMFIA and the VFMFIA increase from view of time consumed, and hence this will reduce the time consumed to generate candidate support count.

## 7. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed CMFIA, it is implemented in python on a 2.10 GHZ Intel Pentium Dual Core with 4.00 GB of main memory. The performance of the proposed CMFIA is compared with R language. Timing in the figure is depends on the total time comprise of all preprocessing costs.

We experimented with datasets from different applications. These preprocessed datasets were obtained from the kaggle [15]. Our algorithm has the capability to work with datasets with a huge number of items (more than 70000 items per transaction). We used cardiovascular dataset to show this capability for the proposed algorithm CMFIA.

| Transactions | CMFIA (time in seconds) | VFMFIA (time in seconds) | MFIF (time in seconds) | APRIORI (time in seconds) |
|---|---|---|---|---|
| 100 | 0:00:00.28 | 0.006 | 0.016 | 0.187 |
| 500 | 0:00:00.38 | 0.050 | 0.062 | 0.422 |
| 5000 | 0:00:00.33 | 0.199 | 0.266 | 1.047 |

*Table 1: performance comparison with different transactions*

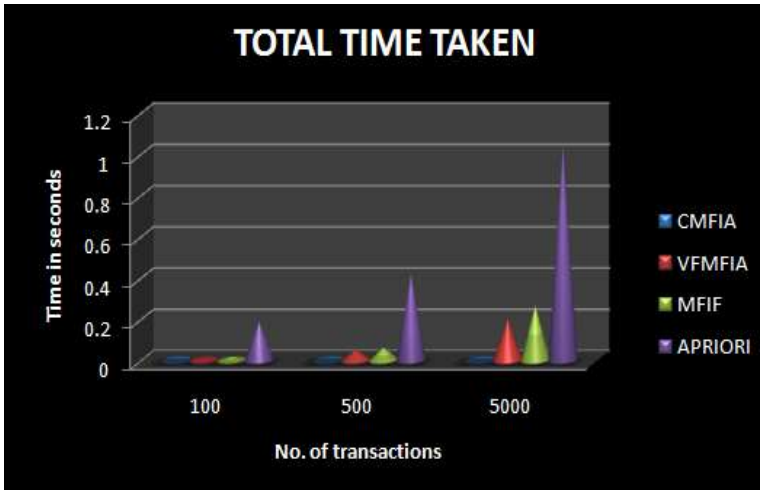


*Figure 12: Time taken by various algorithms*

The results are shown in Table 1 and graphical comparison in fig.12. Time taken by VFMFIA, MFIF and Apriori for 100, 500, 5000 transactions is shown in the Table 1.

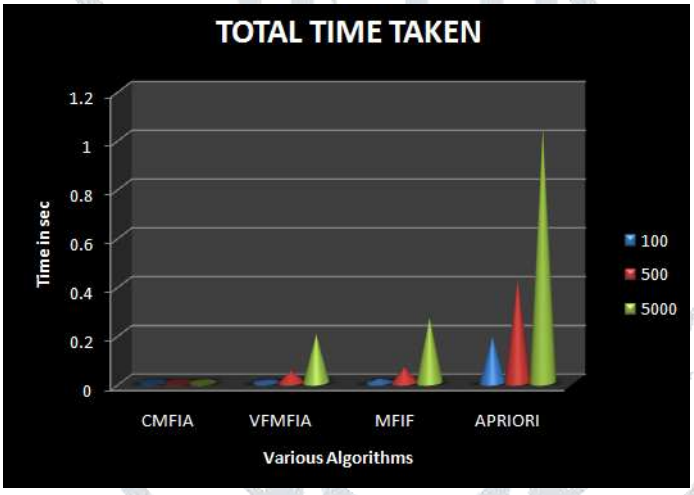


*Figure 13: Time taken by various algorithms*

Figures 13 and Table 1 are comparison of CMFIA, VFMFIA, MFIF and APRIORI were made with 100 transactions based on risk of heart diseases. This figures show MFIA the time it needs to process is 0.0.0.28, 0.006, MFIF needs 0.016 and APRIORI needs 0.187. Overall, these results of area reveal better performance in CMFIA algorithm

Here confusion matrix is used as a tabular way of visualizing the performance of the CMFIA. Each entry in a confusion matrix denotes the number of predictions made by the model where it classified the classes correctly or incorrectly.

True Positive (TP) refers to the number of predictions where the classifier correctly predicts the positive class as positive. True Negative (TN) refers to the number of predictions where the classifier correctly predicts the negative class as negative. False Positive (FP) refers to the number of predictions where the classifier incorrectly predicts the negative class as positive. False Negative (FN) refers to the number of predictions where the classifier incorrectly predicts the positive class as negative.

Using Google Colaboratory, CMFIA is being executed to find the performance measures for your model. Some of the most common performance measures of the confusion matrix were explained.

**Accuracy:** The overall accuracy of the CMFIA, the fraction of the total samples that were correctly classified by the classifier. *(TP+TN)/ (TP+TN+FP+FN)* is used to calculate the accuracy.

| ID | No. of transactions | Accuracy |
|---|---|---|
| 1. | 30000 | 0.80 |
| 2. | 50000 | 0.80 |
| 3. | 70000 | 0.80 |

*Table 2: Comparison of accuracy with different transactions*

CMFIA uses 30000, 50000 and 70000 transactions to find the accuracy. The accuracy of CMFIA with 10000 transactions is 0.80, 50000 transactions is 0.80 and 70000 transactions is 0.80.

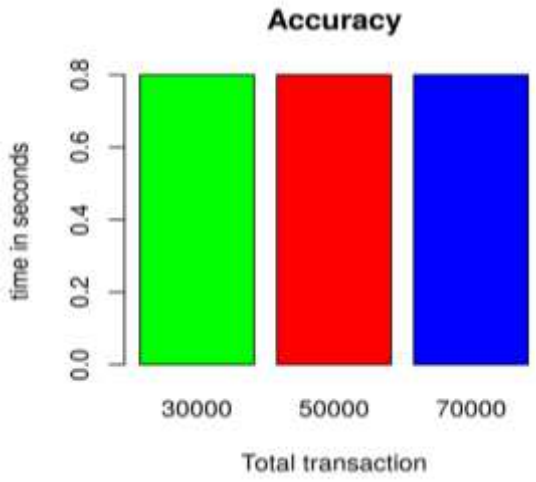


*Figure 14: Accuracy of various transactions*

The results are shown in Table 2 and graphical comparison in fig.14. Accuracy 30000, 50000, 70000 transactions is shown.

**Precision:** It gives what fractions of predictions in CMFIA were as positive classes were actually positive. *TP/ (TP+FP)* is used to calculate precision.

| ID | No. of transactions | Precision Yes | Precision No |
|---|---|---|---|
| 1. | 30000 | 0.00 | 0.80 |
| 2. | 50000 | 0.00 | 0.80 |
| 3. | 70000 | 0.00 | 0.80 |

*Table 3: Comparison of precision with different transactions*

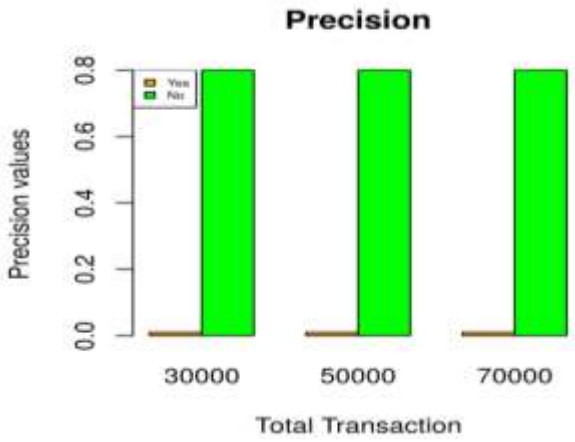


*Figure 15: Precision of various transactions*

The precision results are shown in Table 3 and graphical comparison in fig.15 is shown. The precision of CMFIA with precision yes for 30000 transactions is 0.00, 50000 transactions is 0.00 and 70000 transactions is 0.00, precision no for 30000 transactions is 0.80, 50000 transactions is 0.80 and 70000 transactions is 0.80.

**Recall:** It gives what fractions of all positive samples in CMFIA were correctly predicted as positive by the classifier. It is also known as True Positive Rate (TPR), Sensitivity, and Probability of Detection. *TP/ (TP+FN)* is used to calculate Recall.

| ID | No. of transactions | Recall Yes | Recall No |
|---|---|---|---|
| 1. | 30000 | 0.00 | 1.00 |
| 2. | 50000 | 0.00 | 1.00 |
| 3. | 70000 | 0.00 | 1.00 |

*Table 4: Comparison of recall with different transactions*

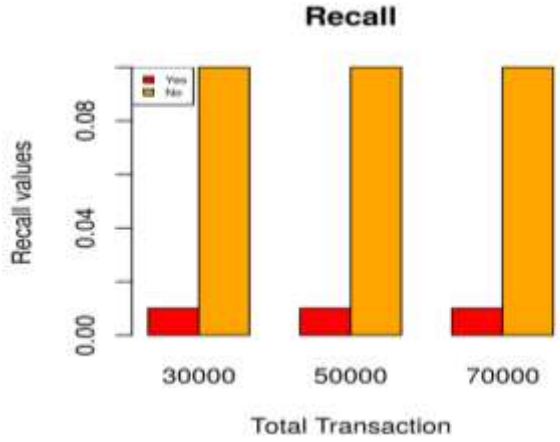


*Figure 16: Recall of various transactions*

The recall results are shown in Table 4 and graphical comparison in fig.16 is shown. The recall of CMFIA with recall yes for 30000 transactions is 0.00, 50000 transactions is 0.00 and 70000 transactions is 0.00, recall no for 30000 transactions is 1.00, 50000 transactions is 1.00 and 70000 transactions is 1.00.

**F1-score:** It combines precision and recall into a single measure in CMFIA. Mathematically it's the harmonic mean of precision and recall. *2TP/ (2TP+FP+FN)* is used to calculate F1-score.

| ID | No. of transactions | F1-score Yes | F1-score No |
|---|---|---|---|
| 1. | 30000 | 0.00 | 0.89 |
| 2. | 50000 | 0.00 | 0.89 |
| 3. | 70000 | 0.00 | 0.89 |

*Table 5: Comparison of F1-Score with different transactions*

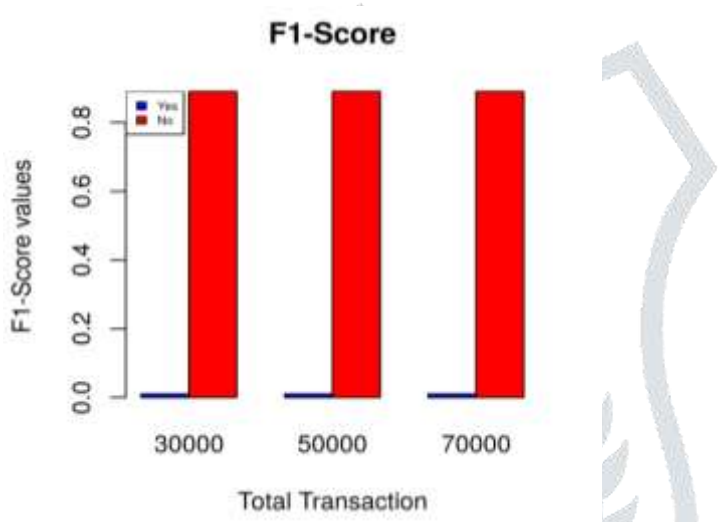


*Figure 17: F1-Score of various transactions*

The F1-Score results are shown in Table 5 and graphical comparison in fig.17 is shown. The precision of CMFIA with F1-Score yes for 30000 transactions is 0.00, 50000 transactions is 0.00 and 70000 transactions is 0.00, F1-Score no for 30000 transactions is 0.89, 50000 transactions is 0.89 and 70000 transactions is 0.89.

**Support:** In the confusion matrix of CMFIA with and without normalization by class support size (number of elements in each class).

| ID | No. of transactions | Support Yes | Support No |
|---|---|---|---|
| 1. | 30000 | 1475 | 6025 |
| 2. | 50000 | 2458 | 10042 |
| 3. | 70000 | 3461 | 14039 |

*Table 6: Comparison of Support with different transactions*

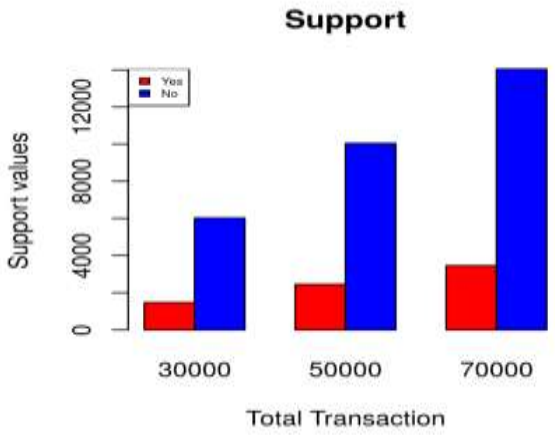


*Figure 18: Support of various transactions*

The Support results are shown in Table 5 and graphical comparison in fig.17 is shown. The Support of CMFIA with precision yes for 30000 transactions is 1475, 50000 transactions is 2458 and 70000 transactions is 3461, Support no for 30000 transactions is 6025, 50000 transactions is 10042 and 70000 transactions is 14039.

**Micro:** It is calculated by considering the total TP, total FP and total FN of the CMFIA. It does not consider each class individually and it calculates the metrics globally.

| ID | No. of transactions | Precision | | |
|---|---|---|---|---|
| | | Micro | Macro | Weighted |
| 1. | 30000 | 0.80 | 0.40 | 0.65 |
| 2. | 50000 | 0.80 | 0.40 | 0.65 |
| 3. | 70000 | 0.80 | 0.40 | 0.64 |

*Table 7: Comparison of Precision Micro, Macro and Weighted with different transactions*

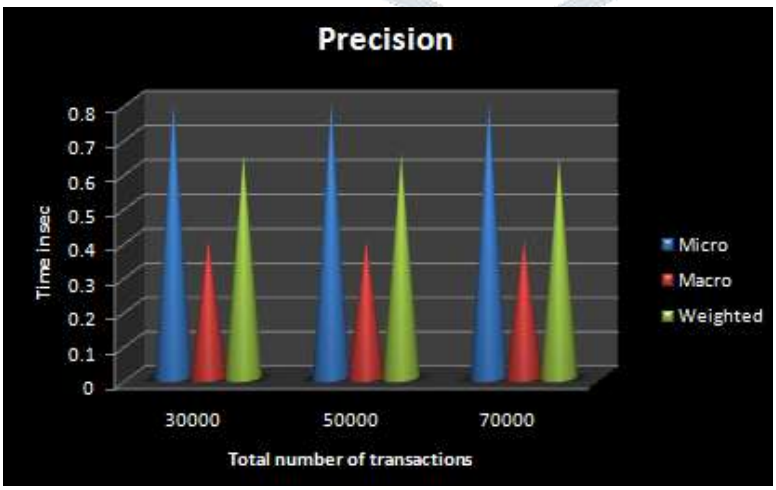


*Figure 19: Comparison of Precision Micro, Macro and Weighted with different transactions*

**Macro**: It calculates metrics for each class individually and then takes unweighted mean of the measures in CMFIA.

| ID | No. of transactions | Recall | | |
|---|---|---|---|---|
| | | Micro | Macro | Weighted |
| 1. | 30000 | 0.80 | 0.50 | 0.80 |
| 2. | 50000 | 0.80 | 0.50 | 0.80 |
| 3. | 70000 | 0.80 | 0.80 | 0.80 |

*Table 8: Comparison of Recall Micro, Macro and Weighted with different transactions*

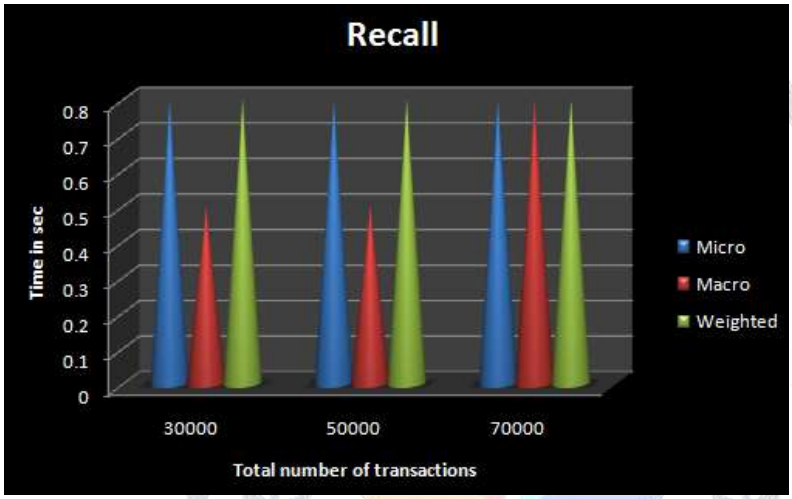


*Figure 20: Comparison of Recall Micro, Macro and Weighted with different transactions*

**Weighted:** It takes a weighted mean of the measures of CMFIA. The weights for each class are the total number of samples of that class.

| ID | No. of transactions | F1-Score | | |
|---|---|---|---|---|
| | | Micro | Macro | Weighted |
| 1. | 30000 | 0.72 | 0.45 | 0.72 |
| 2. | 50000 | 0.80 | 0.45 | 0.72 |
| 3. | 70000 | 0.80 | 0.45 | 0.71 |

*Table 9: Comparison of F1-Score Micro, Macro and Weighted with different transactions*

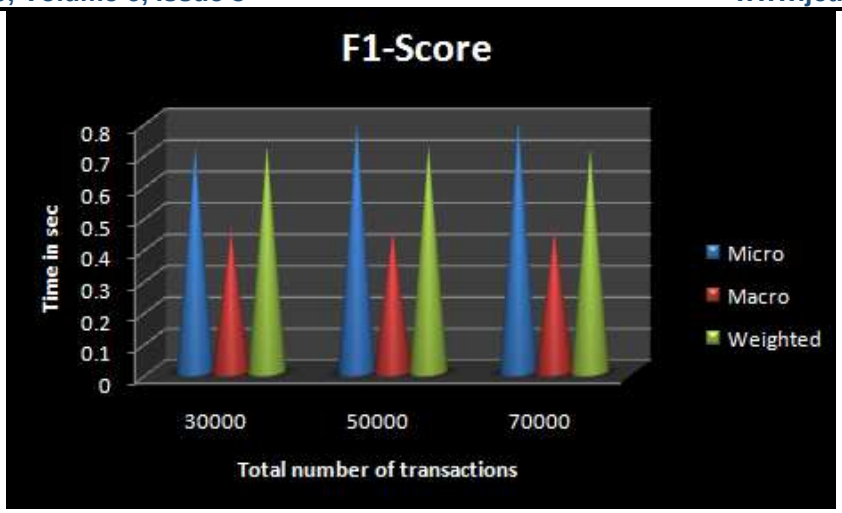


*Figure 21: Comparison of F1-Score Micro, Macro and Weighted with different transactions*

As we observe that the time consuming in CMFIA in each value of minimum support is less than it in the VFMFI, and the difference increases more and more as the value of minimum support decreases.

## 8. CONCLUSION

In this paper, a CMFIA is proposed through reducing the time consumed in transactions scanning for candidate itemsets by reducing the number of transactions to be scanned. Whenever the k of k-itemset increases, the gap between our CMFIA and the VFMFI increases from view of time consumed, and whenever the value of minimum support increases, the gap between our CMFIA and the VFMFI decreases from view of time consumed. The time consumed to generate candidate support count in our CMFIA is less than the time consumed in the VFMFI. In this research paper, we have presented Heart disease prediction system using data mining is used. The method is developed by arranging the total in descending order and substituting a prime integer. Because the CMFIA model produces superior results and assists domain experts and others in the field in planning for a better diagnosis and providing early diagnosis results to patients, it functions reasonably well even when not retrained. The results of the trial reveal that CMFIA is approximately 100 percent accurate in predicting heart disease.

## 9. REFERENCES

1. H. H. O. Nasereddin, “Stream data mining,” International Journal of Web Applications, vol. 1, no. 4, pp. 183–190, 2009.

2. F. H. AL-Zawaidah, Y. H. Jbara, and A. L. Marwan, “An Improved Algorithm for Mining Association Rules in Large Databases,” Vol. 1, No. 7, 311-316, 2011

3. S. Rao, R. Gupta, “Implementing Improved Algorithm Over APRIORI Data Mining Association Rule Algorithm”, International Journal of Computer Science And Technology, pp. 489-493, Mar. 2012

4. Agrawal, R, Imielinksi, T and Swami, A (1993). “Mining association rules between sets of items in large database”. The ACM SIGMOD Conference.

5. Karam Gouda, Mohammed J. Zaki GENMAX: An Efficient Algorithm For Mining Maximal Frequent Itemsets, Data Mining and Knowledge Discovery, 11, 223–242, 2005_c 2005 Springer Science

6. Lin, D.-L., and Kedem, Z.M. 1998. Pincer-search: A new algorithm for discovering the maximum frequent set. In 6th Intl. Conf. on Extending Database Technology, pp. 105–119.

7. Qinghua Zou, Wesley W. Chu and Baojing Lu, "Smartminer: A Depth First Algorithm Guided By Tail Information For Mining Maximal Frequent Itemsets" Proceedings 2002 IEEE International Conference on Data Mining. ICDM 2002 , vol., no., pp.570 – 577, 2002

8. Hua-Fu Li, Suh-Yin Lee and Man-Kwan Shan, "Online Mining (Recently) Maximal Frequent Itemsets Over Data Streams", Research Issues in Data Engineering: Stream Data Mining and Applications,2005. RIDE-SDMA 2005. 15th International Workshop on , vol., no., pp. 11- 18, 3-4 April 2005 International journal of Data Mining & Knowledge Management Process (IJDKP) Vol.3, No.1, January 2013 38

9. Bayardo, R.J. 1998. Efficiently mining long patterns from databases. In ACM SIGMOD Conf. on Management of Data, pp. 85–93.

10. Burdick, D., M. Calimlim and J. Gehrke, "MAFIA: A maximal frequent itemset algorithm for transactional databases", In International Conference on Data Engineering, pp: 443 – 452, April 2001

11. Lin, D.-L., and Kedem, Z.M. 1998. Pincer-search: A new algorithm for discovering the maximum frequent set. In 6th Intl. Conf. on Extending Database Technology, pp. 105–119.

12. Han, J, Pei, H and Yin, Y (2000). "Mining Frequent Patterns without Candidate Generation". Conf. on the Management of Data (SIGMOD'00, Dallas, TX). New York, NY, USA.

13. Lin, D., Kedem, Z.M.: Pincer-Search: A New Algorithm For Discovering The Maximum Frequent Set. In: Proceedings of VI International Conference on Extending Database Technology (1998)

14. Roberto Bayardo, "Efficiently mining long patterns from databases", in ACM SIGMOD Conference 1998

15. https://www.kaggle.com/smeetamary/cardiovascularheart2019dataset/heart_jesus.csv