Efficient Association Rules Mining from Streaming Data With A Fault Tolerance

Amr Aly Abd Elaty
Information Systems Department
Faculty of Computers and
Information, Menoufia University
Shebin Elkom, Egypt
amraly_267@yahoo.com

Rashed Salem
Information Systems Department
Faculty of Computers and
Information, Menoufia University
Shebin Elkom, Egypt
rsalem@ci.menofia.edu.eg

Hatem Abd Elkader
Information Systems Department
Faculty of Computers and
Information, Menoufia University
Shebin Elkom, Egypt
hatem6803@yahoo.com

Abstract— **Recently, number of applications including social networks, stock market trading and sensor network devices generate a huge amount of data in the form of streams. Streaming data have characteristics different from static data, such as streaming data arrives continuously at high speed with huge amount. Mining and extracting information from these data is a real challenge. Most of traditional algorithms have limitations to deal with streaming data, so there are new issues raised and need to be considered when developing association rule mining techniques for these data. In this paper, a technique to mine an association rules from streaming data efficiently is proposed. The proposed technique develops a tree structure called Fast Update Frequent Pattern Tree (FUFP-Tree) that reduce the number of traversing between tree nodes in both inserting a new transaction and extracting an association rules between items. Also, to avoid congestion during inserting incoming streaming data to FUFP-Tree, a sliding window approach is used to divide incoming data equally to all available windows. The complexity and the efficiency of this technique are discussed, and a dataset of storehouse is used to test the proposed technique and measure its efficiency. The efficiency of the proposed technique is compared with other most related algorithms.**

Keywords— Data Mining, Association Rules, Streaming Data.

I. Introduction

Over the past years, some applications such as social networks, stock market trading and sensor network devices need to process data as they are generated, in other words, as they stream. These types of applications are called streaming data applications. The term of streaming data refers to data that is generated continuously with unbounded size and arrives in high speed, as opposed to static data. Stream data mining is the process of extracting knowledge structures from continuous and rapid data records, it represents the next generation of data mining systems that will support the intelligent and time-critical information needs of mobile users and will facilitate ‘‘anytime, anywhere’’ data mining [1]. Association rule mining is a procedure that is meant to find frequent patterns, correlations or associations from a given dataset. It is termed as “market basket dataset”, where each attribute is termed as an item and the frequencies of different itemsets are transformed in the form of if-then rules based on support-confidence framework and then the relationships between seemingly unrelated data in given dataset can be found out [2]. Streaming data applications require association rule mining to find out the major associations between items. Some of related algorithms are discussed, there are limitations for these algorithms to deal with streaming data and mining an association rules in high performance and in acceptance time. The proposed technique tackles the discussed algorithms limitations and uses a sliding window to build an enhanced FP-Tree called FUFP-Tree. Also, the proposed technique allows fault tolerance layer that aims to save the resulted tree after a period of time to retrieve it in the case of any error or damage to the system rather than rebuild the whole tree again from scratch. Saved tree can be used to apply user query using a minimum support and minimum confidence to generate the association rules. The proposed technique is applied for a dataset and the experimental results proved that it is efficient compared to other similar algorithms. In this paper, the related works are discussed declaring the limitations of each algorithm in section II, then the proposed technique is developed based on the discussed limitations in section III. The complexity of the algorithm is discussed in section IV. Moreover, the experimental results and querying the proposed tree are discussed in section V. Finally, the conclusions and future works are discussed in section VI.

II. Related Works

The main challenge in mining frequent itemsets in streaming data is to enumerate the frequency of them at a rate that is compatible with the speed at which the transactions are presented. This goal requires algorithms with in memory data structures and a minimal dataset scan . According to [2-3], the approaches can be grouped into four main categories: bottom-up, top-down, landmark and sliding-window based mining.

1) Bottom-Up Approach

In this approach, the individual transactions itemsets of the given dataset are specified in detail firstly. Then these itemsets are linked together to form larger sub-transactions in many levels and so on until complete top-level sub-transactions are formed. The popular association rules algorithms in this approach are Apriori Algorithm and Partitioning Approach.

A. Apriori Algorithm

Apriori Algorithm is one of the most popular algorithms in data mining for learning the concept of association rules [4]. It can be used to derive all possible frequent itemsets from a given dataset and generate association rules subject to support and confidence values that are greater than or equal to a user specified minimum support and minimum confidence. Although it is robust as the noisy data can't affect the outcome of the algorithm, but it has limitations when dealing with streaming data. The major limitation of this algorithm is the multiple scanning of the dataset when there is a new record is inserted. Also, it's a costly waste of time to generate a number of candidate sets with much frequent itemsets, so that it will be very slow and inefficient when memory capacity is limited.

B. Partitioning Approach

It can be observed that during the frequent itemset generation, maximum time is consumed while reading the data from the disk. To execute faster, the dataset need to be loaded to the memory. But in most of the cases the dataset is too big to load into the memory. The partitioning approach uses the Apriori algorithm for memory resident data [2]. In this approach, the whole dataset is divided into some smaller partitions, so that each partition is individually loaded in the memory. Then for each partition, a frequent itemsets are generated using the Apriori algorithm. After the generation of frequent itemsets for all the partitions, they are combined together and redundancies are removed. Then for all the remaining itemsets the support is counted by reading the dataset again. This approach practically requires two scans of the whole dataset. This approach still suffers from limitations such as it isn't sensitive to noisy data. This approach also scans the dataset only twice. Moreover, in the final phase, joining of frequent itemsets of individual partitions results in a huge number of itemsets and hence consumes a significant amount of time.

2) Top-Down Approach

In this approach, an overview of the given dataset is formulated, then breaking down to gain insight from given dataset. The most popular association rules algorithm in this approach is FP-tree algorithm.

- *Frequent-Pattern growth (FP-Growth) algorithm*

It is the most efficient tree-based algorithm in finding out the desired association rules. This algorithm firstly scans the dataset to find out frequencies of different items, then it sorts the items based on the frequency of each item in the decreasing order. Using the frequency descending list, the dataset is compressed into a Frequent-Pattern tree, which retains the information about the association of the itemsets. Next, for each item starting with the highest support, a conditional pattern base is constructed and represented as its conditional FP-tree. The growth pattern is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. After the construction of the FP-tree, for every frequent item one conditional FP tree is constructed. However, a major limitation of FP-growth is that, this algorithm needs to scan the given dataset twice [5]: First scan to get frequency of occurrence for each item, second scan to reorder the dataset transaction items according to the frequency of occurrence of each item.

3) Landmark Approach

In landmark algorithms, itemsets are counted on the transactions between a specific timestamp, the landmark, and the present. Thus, in landmark algorithms, records are ongoing in the frame of interest. Landmark algorithms are based on a single pass support count of streaming data and on prefix tree-based pattern representation [3]. DSM-FI algorithm is a popular algorithm that's based on landmark approach.

- *Data Stream Mining for Frequent Itemsets (DSM-FI) Algorithm*

In this algorithm, it constructs and maintains an in-memory prefix-tree based summary data structure, called SFI-forest (summary frequent itemset forest). A DSM-FI algorithm prunes the infrequent information from the current SFI-forest. Finally, the frequent itemsets from the current SFI-forest are generated [6]. The major limitation of this algorithm is that it needs more tree traversals for the frequency count, so that it consumes more time in both inserting and generating an association rules.

4) Sliding-Window Based Mining

The main challenges are avoiding multiple scans because the streaming data come from one source or multiple sources in a high speed. So that this approach is based on the sliding window model, which completely discard stale data and attention is focused on recent data, thus saving memory storage and facilitating the detection of the distribution drift [7]. There are many algorithms that using sliding window based mining such as Weighted Sliding Window (WSW) algorithm.

- *Weighted Sliding Window (WSW) Algorithm*

This algorithm depends on the number of windows for mining, the size of the window and the weight for each window, which are predefined. The incoming transactions are split into equal number of windows, and then calculate the weight of each transaction in each window. Hence, the highest weight has been assigned to the most recent transaction. If the weighted support count of an item is greater than or equal to the minimum weighted value, it is called as frequent itemset. When increasing the window size, the execution time of WSW decreases, this is because when the window size small, the number of transaction containing frequent item sets in each window is small. The main limitation of this algorithm is that weights of each window affected the mining results, so that user should specify the reasonable weight for each window and adjust the weights values for different windows based on the importance of the data. All discussed related works are summarized and compared in Table 1.

TABLE 1. Comparative analysis of frequent pattern algorithms

Algorithm Name	*Category*	*Advantages*	*Limitations*
Apriori Algorithm	Bottom-Up	Can derive all possible frequent itemsets from a given dataset and generate association rules subject to minimum support and minimum confidence values	Multiple scanning of the dataset when there is a new record is inserted
Partitioning Approach	Bottom-Up	Generate frequent itemsets faster without burden on the memory	Not sensitive to noisy data
Frequent-Pattern growth (FP-Growth) algorithm	Top-Down	Find out frequencies of different itemsets, then order the itemsets descending into a compressed frequent pattern tree	Needs to scan the given dataset twice
Data Stream Mining For Frequent Itemsets (DSM-FI) Algorithm	Landmark	Compact tree structure has been designed to store the frequent patterns	It needs more tree traversals for the frequency count
Weighted Sliding Window (WSW) Algorithm	sliding-window based mining	A single pass algorithm was developed to discover the frequent itemsets	Weights of each window affected the mining results. So, user should specify the reasonable weight for each window

III. Efficient Association Rules Mining With A Fault Tolerance

In this section, a proposed technique is discussed. The main purpose of the proposed technique is improving a tree structure which can accommodate streaming data and change continuously, so that, a Fast Updated Frequent Pattern Tree (FUFP-Tree) is proposed. Furthermore, a Sliding-Window technique is used to speed up preprocessing of incoming streaming data before sending it to FUFP-Tree in a parallelism form with a fault tolerance level. Finally the association rules between items can be extracted easily from the built FUFP-Tree according to given parameters such as minimum support value, maximum support value and confidence value. All these issues are detailed in the following sections.

A. Streaming Data Preprocessing Using Sliding-Window

With the exponential growth of streaming, an unprecedented amount of structured, semi-structured, and unstructured data is available. So that, data preprocessing is a major phase to solve incoming data problems before insertion in a tree such as inconsistencies, missing values and noise data to provide a high quality data to improve the performance of used algorithms to extract the association rules. To handle continuous data streams, a sliding window model is used for parallel preprocessing entry data. Typically, the incoming data will be split into equal chunks according to the window slide size. Given a degree of parallelism in the system, each window slide will act as a separate part and apply a set of data preprocessing operations such as data cleaning, data integration, data transformation,…,etc. A common sliding window technique called "Pane-based Partitioning" is used which based on the panes [9]. The main idea of this technique is to divide overlapping windows internally into individual panes, over which sub-aggregates can be computed whose results can be combined into the final aggregate. The panes technique has been originally proposed to reduce the space and computation cost of sliding window by sub-aggregating computation. The number of sliding windows, sliding window size and sliding window pane size must be identified firstly before receiving data from sources. In "Pane-based Partitioning" technique each window acts as a shared-nothing cluster of commodity hardware in which each window is independent, self-sufficient according to the window size. There is no shared memory or disk storage, so that if there is any failure in any sliding window, other sliding windows will continue their work and not affected by the existing failure. A sliding window can become out of service due to a common failure called “Fail-Stop Failure”.

- *Fail-Stop Failure*

This failure means that if the sliding window gets out of services during a computation, it requires an external impact to bring the sliding window back to working state again. For instance, a system administrator checks for the status of sliding window components and solve the problem or retires the broken sliding window and reconfigure the system such as reinitializing the number of sliding windows.

B. Sliding-Window Fault Tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components due to one or more faults. As mentioned previously, the streaming data comes in continously and with high speed rate, so that there is no ability to hold longer until the faults are processed from system

`

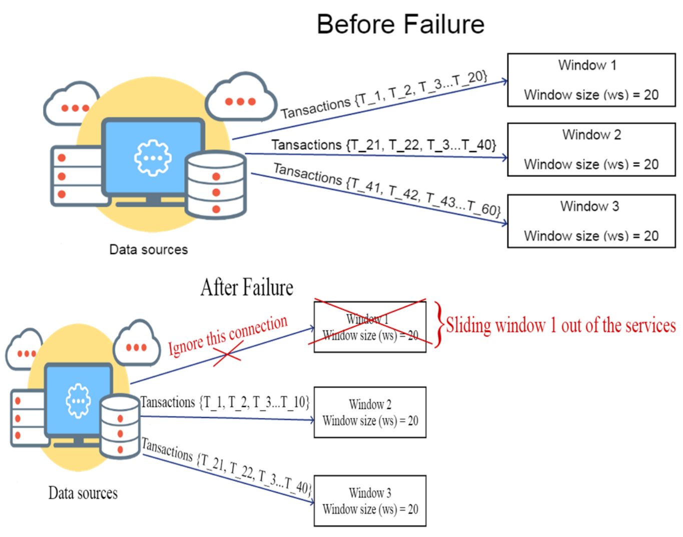

Fig 1. Sliding window fault tolerance

adminstrator. In the statue of Fail-Stop Failure, the proposed solution is that the system will automatically disable and ignore the connection to the failure sliding window, so that all incoming transactions will be split directly to other working sliding windows without change the initial configuration of the sliding windows until the system admin interferes to resolve the issue with a appropriate solution. For instance, assume there is a system consist of *n* sliding window and each window can handle *m* transaction (*i.e.*, window size = *m*), if one sliding window became out of service, that means there are *n*-1 working sliding window rather than *n* working sliding window and the transactions of the crashed sliding window will be redirect automatically to the other working sliding windows, see Fig 1.

C. Fast Updated Frequent Pattern Tree (FUFP-Tree)

The FUFP-tree construction algorithm is based on the FP-tree algorithm. The links between parent nodes and their child nodes are bi-directional linking that help fasten the process of item insertion in the maintenance process. Besides, the frequent items are sorted in descending order and kept in the top nodes. Also, FUFP-Tree saves the last path that eases moving from and to nodes. For example, if last inserted transaction is {B, D, A, E}, then the saved path is {B:5, D:3, A:2, E:1} as shown in Fig 2. If there is a new transaction {B, D, A, E, C}, there is no need to return to Null node and start from scratch to search for the header node, i.e., {B}, the saved path can be used as its items are found in the new incoming transaction, so the support of each item in the current saved path will be incremented as {B:6, D:4, A:3, E:2}. Then after adding a new item node {C} with its frequency, now the current saved path is updated to {B:6, D:4, A:3, E:2, C:1} as illustrated in Fig 3. If the new transaction starts with an item that doesn't match the first saved path item, the return to Null node and search for header node that match the first transaction item is inevitable. When new transactions are added, the FUFP-tree maintenance algorithm will process them to maintain the FUFP-tree. The top nodes of FUFP-tree are updated whenever necessary. When an originally large item becomes small, it is directly transfer from top node to lower node according to its frequency value and its

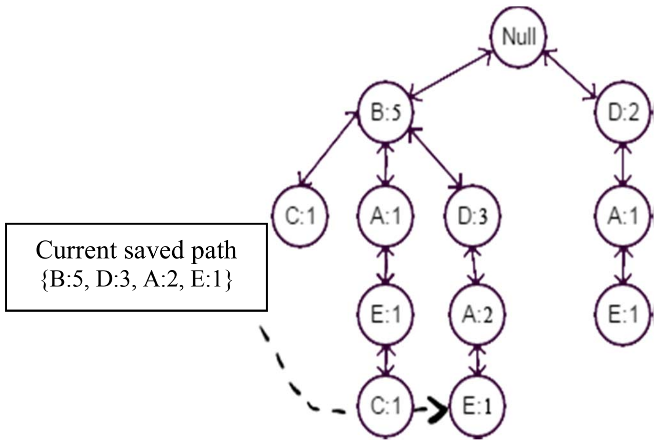

Fig 2. FUFP-Tree before inserting {B, D, A, E, C} transaction

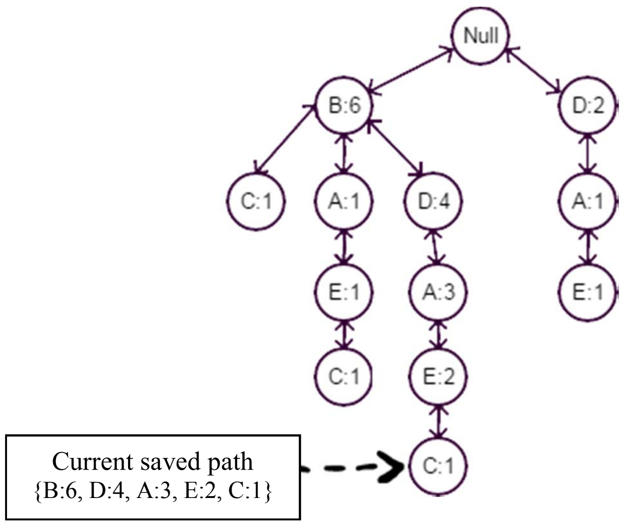

Fig 3. FUFP-Tree after inserting {B, D, A, E, C} transaction

parent and child nodes are then linked together. The FUFP-tree can thus be least updated in this way, and the performance of the FUFP-tree maintenance algorithm can be greatly improved. The entire FUFP tree can then be re-constructed in a batch way when a sufficiently large number of transactions have been inserted.

D. Extracting Association Rules

Association rule is a data mining method for discovering interesting relations between different items. After constructing the FUFP-Tree, Algorithm A.1 is used to generate the complete set of exact frequent patterns from the current window [1]. Firstly, a minimum support threshold value must be identified to extract a frequent pattern list. In FUFP-Tree, the nodes are ordered in descending order, so that if the top node is not greater than or equal to the minimum support threshold value, this node and its children will be ignored from frequent pattern list. For example, from the same FUFP-Tree which shown in Fig 3, if the minimum support threshold value is 4,

```
Input: FUFP-Tree, Predefined minimum support threshold min_sup
Output: The frequent patterns list $L_{fp}$
Begin
1-  For each node N in first level of FUFP-Tree
2-    If frequency(N) >= min_sup   Then
3-        Extend the Node N
4-            For each item I in Node N
5-                If frequency(I) >= min_sup   Then
6-                    Add item I to $L_{fp}$ , Then goto next level
7-                Else
8-                    Break
9-                End If
10-           End For
11-       End If
12- End For
```

A.1. Extract frequent patterns

```
Input : $L_{fp}$ list, Predefined confidence (Conf)
Output: The association rule list $L_{ar}$
Begin
1-  For each item $L_i$ in $L_{fp}$
2-    For each item $L_j$ in $L_{fp}$
3-      Rule_val = Sup($L_i$ U $L_j$)/Sup($L_i$)
4-        If (Rule_val  >=  Conf)   Then
5-          Add $L_i$ and $L_j$ to $L_{ar}$ list
6-        End If
7-    End For
8-  End For
```

A.2. Deduce association rules

the resulted frequent pattern list will be {B},{B,D}. After generating all possible frequent patterns, association rules are deduced according to a certain confidence using Algorithm A.2.

IV. Complexity of Algorithm

In this section, the algorithm complexity is analyzed. Runtime complexity is divided into two steps.

- Data preprocessing step which is depending on the number of sliding windows and the number of transactions per a sliding window (n),so that the complexity of sliding windows is $O(n)$.
- Generating frequent patterns list from FUFP-Tree step which its complexity is $O(\log n)$.

V. Experimental Results

To assess the validity of the proposed technique, an experiment is conducted. A sample of a retail market basket dataset of a retail supermarket store is used. The number of transactions is 88163 and the number of unique stock keeping unit products is 16470 [10].

TABLE 2. Total runtime with different sliding window configuration

Number of windows (nw)	*Window size (ws)*	*Number of panes per window (wp)*	*Size of pane per each pane in window (ps)*	*Total Runtime (sec)*
10	8800	80	110	20.5
20	4400	50	88	12.9
30	2930	10	293	8.7
40	2200	20	110	7.4
50	1760	10	176	4.2

A. *Sliding Windows Configuration*

The main parameters for sliding windows are: number of windows (*nw*), window size (*ws*), number of panes per window (*wp*), size of pane per each pane in window (*ps*). All of these parameters are determined based on machine capabilities such as the machine operating system, hard drive type, hard drive capacity, processor and memory capabilities. In this experiment, the used machine specifications are as follow; Ubuntu 18.10 OS, solid state drive (SSD) with 100GB available capacity, processor is Intel core i5, 2.40 GHz and 4 GB RAM. Sliding windows configuration must be done quite ideally, because these sliding windows have a major role as they receive a raw data from sources and then apply a preprocessing function to be ready to be inserted into the FUFP-Tree. So that one of the best tools that is used during this experimental is called "Apache Kafka". Kafka is an open source tool that is used for building streaming applications. This tool is scalable, fault-tolerant and allows executing operations fast. In this experiment, the parameters are configured with different values in proportion to the capability of the machine and deduce the execution time as shown in Table 2.

B. *Building FUFP-Tree Using Pane-Based Partitioning VS. Building FUFP-Tree Using Weighted Sliding Window*

In this experiment the FUFP-Tree building time is measured by applying the proposed technqiue and applying weighted sliding window (WSW) algorithm. As mentioned previously, WSW algorithm needs to identify number of windows for mining, the size of the window and the weight for each window firstly. The major limitation is that the weights of each window affect the mining results. It requires very reasonable weight for each window and adjusts the weights values each time for each window, so that it takes more time to construct FUFP-Tree than using pane based partitioning approach. In pane-based partitioning approach, there is no need to adjust the configurations that are identified before starting, so that the FUFP-Tree is constructed faster as shown in Fig 4.

`

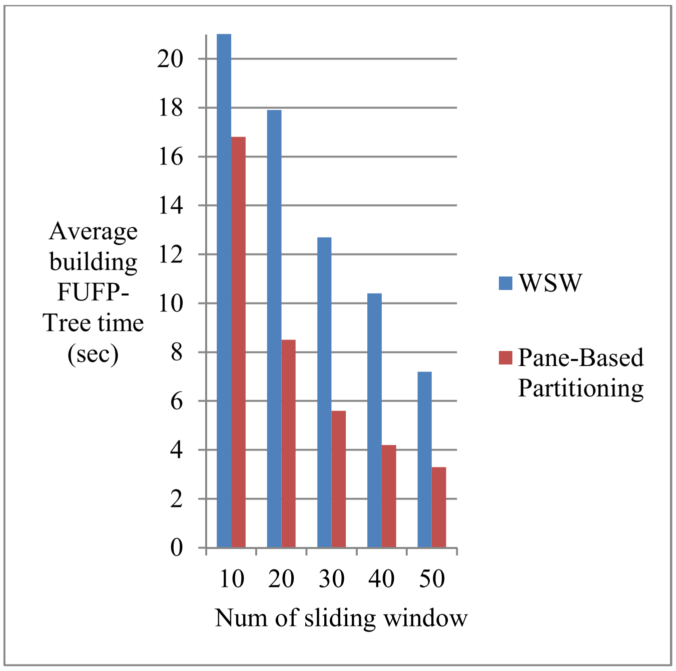

Fig 4. Average time for building FUFP-Tree

TABLE 3. Top 5 frequent patterns list

Product Barcode	***Frequency***
39	4321
48	3436
41	2024
32	1397
38	1348

C. *Extracting Frequent Patterns List*

After the construction of FUFP-Tree with pane-based partitioning sliding window approach, the association rules algorithm has been applied to extract the frequent patterns list. There are more than 8000 frequent patters are extracted. The top 5 frequent itemsets are listed in Table 3.

D. *Results Analysis*

The results of the proposed algorithm to construct the FUFP-Tree are presented. The results show that FUFP-Tree is highly efficient in terms of memory storage when finding exact frequent patterns from a streaming data. The runtime changes based on sliding windows configurations. As shown in Table 2, if there are more sliding windows, high number of incoming streaming data are processed in the same time, the runtime is reduced and the FUFP-Tree is built in short time.

VI. Conclusion And Future Work

In this paper, the characteristics of streaming data have been discussed and presented some related algorithms for generating association rules from a dataset. The proposed technique works on speeding up streaming data mining using a sliding window technique to build FUFP-Tree with fault tolerance for any failure in the system. In the future, the proposed technique will be improved to adjust the number of needed sliding window and the pane size for each window dynamically according to the rate of streaming data. Such improvement results in memory usage reduction and speed up building FUFP-Tree.

References

[1] A. Moustafa, Badr Abuelnasr, and Mohamed Said Abougabal, “Efficient mining fuzzy association rules from ubiquitous data streams”, published in Alexandria Engineering Journal, vol. 54, pp. 163-174, June. 2015.

[2] B. Nath, D K Bhattacharyya, and A Ghosh, “Incremental Association Rule Mining: A Survey”, published online in Wiley InterScience, vol. 3,pp. 157-169, February 2013.

[3] LuigiTroiano and GiacomoScibelli, “Mining frequent itemsets in data streams within a time horizon”, published online in ELSEVIER, vol. 89, pp. 21-37, January 2014.

[4] Akshita Bhandari, AshutoshGupta, and DebasisDas, “Improvised apriori algorithm using frequent pattern tree for real time applications in data mining”, published online in ELSEVIER, vol. 46, pp. 644-651, 2015.

[5] Jagrati Malviya, Anju Singh, and Divakar Singh, “An FP Tree based Approach for Extracting Frequent Pattern from Large Database by Applying Parallel and Partition Projection”, published in International Journal of Computer Applications (0975 – 8887), vol. 114, pp. 1-5, March 2015.

[6] Hua-Fu Li, Man-Kwan Shan and Suh-Yin Lee, “DSM-FI: an efficient algorithm for mining frequent itemsets in data streams”, published in Spriner Knowledge and Information Systems, vol. 17, pp. 79-97, October 2008.

[7] Fabio Fumarola, Anna Ciampi, Annalisa Appice, and Donato Malerba “A Sliding Window Algorithm for Relational Frequent Patterns Mining from Data Streams”, published in Springer International Conference on Discovery Science, vol. 5808, pp. 385-392, 2009.

[8] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer and A. Swami, “An Interval Classifier for Database Mining Applications,”Proceedings of the VLDB Conference, pp.560-573, 1992.

[9] J. Li et al. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In ACM SIGMOD Conference, Baltimore, MD, USA, June 2005.

[10] T. Brijs, Retail market basket data set, Workshop on Frequent Itemset Mining Implementations (FIMI’03), 2003

[11] Philippe Fournier-Viger, Espérance Mwamikazi, Ted Gueniche1 and Usef Faghihi, “MEIT: Memory Efficient Itemset Tree for Targeted Association Rule Mining”, published in Spriner

[12] International Conference on Advanced Data Mining and Applications, vol. 8347, pp. 95-106, 2013.

[13] Chun-Wei LIN, Tzung-Pei HONG and Wen-Hsiang LU, “Using the Structure of Prelarge Trees to Incrementally Mine Frequent Itemsets”, published in Springer New Generation Computing, vol. 28, pp 5–20, January 2010.

[14] Nataliya Shakhovska, Roman Kaminskyy, Eugen Zasoba and Mykola Tsiutsiura, “Association rules mining in big data”, published in International Journal of Computing, vol. 17, pp. 25-32, 2018.

[15] David del Rio Astorga, Manuel F. Dolz, Javier Fernández and J. Daniel García, ”A generic parallel pattern interface for stream and data processing”, published online in Wiley InterScience, vol. 29, pp. 1-12, May 2017.

[16] Rashed Salem, Jérôme Darmont and Omar Boussaïd, “Efficient incremental breadth-depth XML event mining”, published in 15th International Database Engineering & Applications Symposium, pp. 197-203, September 2011.

`

[17] Xiuli Yuan, “An improved Apriori algorithm for mining association rules”, published in AIP Conference Proceedings, vol. 1820, pp. 080005-1–080005-6,March 2017.

[18] S.Alagukumar, R.Lawrance, “A selective analysis of microarray data using association rule mining”, published online in ELSEVIER, vol. 47, pp. 3-12, 2015.

[19] Min Chen, Shiwen Mao and Yunhao Liu, “Big Data: A Survey”, published in Springer Mobile Networks and Applications, vol. 19, pp. 171-209, April 2014.

`