Itemset Based Pattern Mining Algorithms for Data Streams

A Thesis

submitted to Goa University

for the award of the degree of

Doctor of Philosophy

by

Shankar Bhiwa Naik

under the guidance of

Dr. Jyoti D. Pawar

Department of Computer Science and Technology

Goa University

Taleigao Plateau, Goa

May 2017

Dedicated to

Sahej, Smit, Dipesh & Salman

Statement

As required under the ordinance of Goa University, I, Shankar Bhiwa Naik, state that the present Ph.D. thesis entitled, Itemset Based Frequent Pattern Mining Algorithms for Data Streams' is my original contribution carried out under the supervision of Dr. Jyoti Pawar, Associate Professor, Department of Computer Science and Technology, Goa University and the same has not been submitted on any previous occasion. To the best of my knowledge, the present study is the first comprehensive work of its kind in the area mentioned. The literature related to the problem investigated has been cited. Due acknowledgments have been made whenever facilities and suggestions have been availed of.

Shankar Bhiwa Naik

Certificate

This is to certify that the thesis entitled Itemset Based Frequent Pattern Mining Algorithms for Data Streams, submitted by Shri. Shankar Bhiwa Naik for the award of the degree of Doctor of Philosophy in Computer Science, is based on his original studies carried out under my supervision. The thesis or any part thereof has not been previously submitted for any other degree or diploma in any University or Institute.

Dr. Jyoti D. Pawar
Department of Computer Science and Technology
Goa University, Taleigao Plateau
Goa-403206, India.

Date: 31st May 2017

Place: Department of Computer Science and Technology, Goa University, Goa.

Acknowledgments

I express my sincere gratitude towards my guide **Dr. Jyoti D. Pawar** for her constant help, encouragement and inspiration throughout the research work. Without her invaluable guidance, this work would never have been a successful one. $'DevasiPujitaGurusiAnand, GurusipujitaDevaParmanad'$...It is because of her that I get this opportunity to thank all.

I am grateful to my parents, sister and brother-in-law for all the sacrifices they have made to get this work done.

I thank Dr. Gervasio S.F.L. Mendes, Principal of S.S.A. Govt. College, Pernem for the support and encouragement given to me.

I thank Shri. Bhaskar G. Nayak, Director of Higher Education, Govt. of Goa for having supported me in this journey.

Shankar Bhiwa Naik

Goa University

May 31, 2017

Itemset Based Frequent Pattern Mining Algorithms for Data Streams

by

Shankar B. Naik

Abstract

Itemset mining is defined as the process of discovering interesting and useful patterns in the form of a group of items normally referred to as itemsets which appear together in transaction databases. Although itemset mining was originally designed for market basket analysis, it is viewed more generally as the task of discovering groups of attribute values occurring together in databases. With the technological advancements in both the hardware as well as software the amount of data captured is huge and pose challenges in processing and analyzing it. A data stream belongs to such category of data. The focus of this research work has been to identify patterns from data streams using itemset mining.

A data stream is a continuous sequence or flow of data over time and often at high velocity. Data streams produce a huge amount of data which can come from different sources. Social networks, sensors, financial system and health management systems are good sources of data streams. They generate a huge amount of timely information which may contain packets of hidden knowledge. This knowledge can be extracted in the form of patterns. Detecting patterns from data streams is very challenging due to the inherent characteristics of data streams. The data elements in a data stream arrive in at a high rate and the size of the streams is typically unbounded. It is not possible to store all the elements of the

data stream at once in the system making it impossible to backtrack over data elements or maintain and review the entire history. A straightforward translation of the existing traditional algorithms in data mining is not adequate enough for a data stream mining process. Thus, there is a need to develop frameworks and algorithms to identify patterns from data streams. The elements of the data stream are processed individually or as batches as they arrive. The results of this processing are stored in an intermediate summary data structure which is used to store the summary of the data present in a data stream. This data in the summary data structure is used to generate patterns of interest from a data stream.

The approaches used in the work carried out as part of this research work can be categorized under two broad categories - first category which uses a general approach and the second category which uses an application oriented approach. The work carried out under the first category began with the design of an algorithm which finds frequent itemsets for the entire or a part of an input data stream. Frequent itemsets were generated for segments of the data stream and stored in an intermediate summary data structure. The data in the summary data structure is then used for finding frequent itemsets. During the experimental study, it was observed that the number of frequent itemsets generated increased with increase in the number of segments and became difficult to store them all in the memory. This problem was resolved by designing a single pass incremental algorithm to find closed frequent itemsets from the data stream. A closed itemset has no superset with a similar support. The advantage of finding closed frequent itemsets is that they are less in number and contain complete information about all the subsets of the itemsets that are frequent. These two approaches aimed at generating frequent itemsets from any data stream in general and not from any specific application point of view.

The work carried out under the second category was more from application point of view. Here we proposed a framework to generate itemsets with utilities

and use this utility information to compress the data in the sliding window. The utility of an itemset is a profit value associated with the itemset. In our study, we defined the utility of an itemset in terms of the amount of memory which could be saved by storing the itemset in a compressed form. Itemsets with high utilities were stored in a compressed form, which lead to the reduction in the amount of memory required to store a sliding window thereby allowing more elements in the data stream to be present in the sliding window thereby allowing for more efficient data analysis.

In this research work, we have also presented an approach to dynamically generate the value of the minimum support threshold. Itemset mining on datasets generates a large set of patterns out of which only a few may be of actual interest to the users. A minimum support threshold is used to separate the interesting patterns from the non-interesting ones. The value of minimum support threshold should be carefully chosen. A lower value of it may lead to the generation of many non-interesting patterns as interesting. While a higher value of it may prevent interesting patterns from being shown as interesting. The value of minimum support threshold generated by the approach proposed in this study gives an idea about the data in the sliding window and helps the user to specify his / her minimum support threshold value.

In the concluding part of this research, we have proposed an algorithm to cluster the values of an attribute. This algorithm is applicable to data streams where each element is represented as a pair of values of an attribute. For example, a large number of users create and update their profiles on social websites. These profiles consist of many attributes. User update attribute by replacing the old value with a new one. The pair, of the old and the new values, becomes the element of the data stream considered in this problem. This pair indicates that the two values are related to each other. Analyzing such a pair can produce clusters of attribute values. In case the attribute considered is workplace then each cluster

contains company names related to a similar domain. We extended this algorithm on two attributes and proposed an algorithm to cluster values of an attribute and then generate associations between clusters of two different attributes.

The algorithms were tested using synthetic as well as real datasets. Itemset based pattern mining in data streams has very interesting applications and can be used to mine useful hidden knowledge in transactional data mining, sentiment analysis of messages on social media, web-click pattern mining and sensor-data analysis.

Contents

Statement ii

Certificate iii

Acknowledgments iv

Abstract v

List of Figures xiv

List of Tables xv

1 Introduction 1

1.1 Background 1

1.1.1 Frequent itemset mining 1

1.1.2 Data stream processing 2

1.2 Motivation 3

1.3 Thesis Contribution 4

1.4 Thesis Outline 5

2 Related Work 7

2.1 Frequent Itemset Mining 7

2.2 Closed Frequent Itemset Mining in Data Streams 8

2.3 High utility itemset mining 9

2.4 Attribute Value Clustering 10

3 Frequent Itemset mining using Support Trends 11

3.1 Introduction 11

3.2 Related Work 12

3.3 Problem Definition 13

3.3.1 Preliminaries 13

3.3.2 Problem Statement 13

3.4 Framework for Frequent Itemset Mining Using Support Trends . . . 14

3.4.1 The Intermediate Summary Data Structure 14

3.4.2 Algorithm FIMUST-Frequent Itemset Minimg Using Support Trends 14

3.5 Experimental Study 17

3.5.1 Error calculation 17

3.5.2 Average error versus minimum support 18

3.5.3 Support error versus number of partitions 19

3.6 Conclusion 19

4 Incremental Closed Frequent Itemset Mining using Single Pass Algorithms 21

4.1 Introduction 21

4.2 Related Work and Motivation 22

4.3 Problem Definition 24

4.3.1 Preliminaries 24

4.3.2 Problem Statement 25

4.4 Framework for Incremental Mining of Closed Frequent Itemsets from Data Streams 25

4.4.1 The Intermediate Summary Data Structure 25

4.4.2 Algorithm SPAIM-CFI-Single Pass Algorithm for Incremental Mining of Closed Frequent Itemsets 26
4.5 Experimental Study 36
4.6 Conclusion 38

5 Framework for High Utility Pattern Mining using Closed Frequent Itemsets 39
5.1 Introduction 39
5.2 Related Work 40
5.3 Problem Definition 41
5.3.1 Preliminaries 41
5.3.2 Utility of an itemset 41
5.4 Framework forHigh Utility Itemset Mining using Closed Frequent Itemsets 42
5.4.1 The Intermediate Summary Data Structure 42
5.4.2 The approach 43
5.4.3 Automatic generation of the minimum support threshold s_0 43
5.4.4 Detection of frequent events in the sliding window 44
5.5 Experiments 44
5.5.1 Frequent Itemset Mining using Itemset Utility 44
5.5.2 Frequent pattern generation using dynamically generated s_0 46
5.6 Conclusion 47

6 Clustering Values of Single Attribute in Transitional Data Streams 48
6.1 Introduction 48
6.2 Background and Motivation 51
6.3 Problem Definition 52
6.3.1 Preliminaries 52
6.3.2 Significance of s_0 53

6.3.3 Problem Statement 54
6.4 The Proposed Approach 54
6.4.1 The intermediate summary data structure 54
6.4.2 The Algorithm 55
6.5 Experimental Analysis 59
6.6 Conclusion 60

7 Mining Associations between Clusters of Values of Multiple Attributes in a Data Stream 61
7.1 Introduction 61
7.2 Background and Motivation 63
7.3 Problem Definition 64
7.3.1 Preliminaries 64
7.3.2 Problem Statement 65
7.4 The Approach 65
7.5 The Algorithm 66
7.6 Experimental Analysis 68
7.7 Conclusion 70

8 Conclusion 71

9 Future Work 74

List of Figures

3.1 The framework showing data stream and partition window 13

3.2 Average error in calculated support versus minimum support 18

3.3 Error in calculated support versus number of partitions 19

4.1 Data stream and sliding window 24

4.2 Intermediate summary data structure 25

4.3 Intermediate summary data structure with *Temp* 27

4.4 Algorithm- *Insert* 27

4.5 Intermediate summary data structure 29

4.6 *Index* and *ItemSets* 30

4.7 Intermediate summary data structure 30

4.8 Intermediate summary data structure 31

4.9 Algorithm-*Delete* 32

4.10 Intermediate summary data structure 33

4.11 *ITemp* - Iteration 1 34

4.12 *ITemp* - Iteration 2 34

4.13 Intermediate summary data structure 35

4.14 Intermediate summary data structure 35

4.15 Intermediate summary data structure 35

4.16 Memory required versus minimum support 37

4.17 Execution time required versus minimum support 37

4.18 Execution time required versus minimum support 38

5.1 Data stream and the intermediate summary data structure 42
5.2 Memory saved against sliding window size 45
5.3 Memory saved as window slide over data stream 46
5.4 Dataset- IPL Cricket matches 46
5.5 Snapshot of frequent patterns 47

6.1 The framework 50
6.2 Matrix MAT 56
6.3 Matrix MAT 56
6.4 Algorithm- $Generate$ 57
6.5 The matrix MAT 57
6.6 Iteration 1 of the *Generate* step 58
6.7 Iteration 2 of the *Generate* step 58
6.8 Number of clusters versus minimum support 59
6.9 Execution time versus minimum support 60

7.1 Data stream, clusters and cluster associations 63
7.2 Intermediate summary data structure 65
7.3 $Generate| - Cluster$ step example 67
7.4 $Generate_Association$ step example 68
7.5 Number of clusters versus minimum support 69
7.6 Precision, recall and accuracy of clusters 70

List of Tables

3.1 Parameters of Datasets for FIMUST Algorithm 17
3.2 Experimental results, minsup=0.4 18

4.1 Parameters of Datasets for Closed Frequent Itemset Mining 36

5.1 Dataset Parameters 45

6.1 Parameters of Dataset for Attribute Value Clustering Experiment . 59

7.1 Parameters of Datasets for Cluster Association Generation 68

Chapter 1

Introduction

1.1 Background

In recent years there has been an explosion in the amount of data generated by human activities both on-line and off-line. These data within them contain hidden knowledge that needs to be mined. Data mining aims to design and develop tools and techniques needed to handle such data.

1.1.1 Frequent itemset mining

Data mining is useful in recording the occurrences of certain patterns of knowledge in data. Frequent patterns are sets of data items found together more than a given number of occurrences in data. They are the transactions or itemsets which occur with frequency not less than a predefined threshold called minimum support. The aim of frequent itemset mining is to find out the elements in a dataset which mostly appear together. Frequent itemset mining is one of the beginning steps of other data mining tasks such as frequent sequence mining, association rule mining, amongst others. It was first proposed by Agrawal [3]. There are three methodologies used in generating frequent itemsets from data sets, Apriori[3], FP-Growth[12] and Eclat[11]. The Apriori algorithm generates frequent itemsets

by candidate generation and requires multiple scans of the dataset. FP-Growth generates frequent itemsets without candidate generation and requires less number of dataset scans[12]. Frequent itemset is applied to data stream analysis, sensor network mining, bio-informatics, amongst others [4].

1.1.2 Data stream processing

In this research work we have mainly focused on developing methods for detecting and identifying patterns in data streams. Data Stream is an unbounded, continuous, real-time sequence of data items [5][10]. They produce large amount of data. Their sources are social networks, web-click streams, health management systems, financial systems, radio astronomy and physical science systems, traffic management systems, telecommunication devices, micro-blogging websites, amongst others[4].

For example, the posts tweeted on www.twitter.com is an example of a data stream. A transactional data stream is a sequence of transactions where each transaction is an itemset. A data stream is processed using a sliding window. A sliding window is an excerpt of items pertaining to the stream. Two types of windows are used to process a data stream. They are count-based window and time-based window[9][13]. The count-based window contains a predefined fixed number of elements of a data stream. The time-based window contains elements of the data stream for a fixed period. Our study is based on count-based window. A data stream is processed using three models. They are, the landmark model, the damped window model, and the sliding window model[4]. The landmark window considers elements of the data stream from a timestamp, called as the landmark, through the latest element of the data stream. It treats all elements equally. Usually it starts with the element at the beginning of the stream till the latest element. The damped window model considers the latest elements of the stream more significant than the older ones. It assigns higher weights to new elements

and lower weights to older elements of the data stream. The sliding window model contains the latest n elements of the data stream, where n is the size of the sliding window. A sliding window can be a transactional sliding window or a time-based sliding window. A transactional sliding window contains a fixed number of data stream elements (transactions), whereas a time-based sliding window has a fixed time length.

Approach in data stream processing

It is not possible to store the entire data stream into the memory for processing at once. An element once processed and discarded cannot be revisited again.

The most common approach in processing a data stream is to process the elements of the data stream as they arrive in batches, store them in the memory to generate intermediate results or a summary of the data which are stored in the intermediate summary data structure.

An intermediate summary data structure stores the intermediate results or the summary of the data stream elements which is used to generate final results.

1.2 Motivation

The main objective of this research is to extend frequent itemset mining onto data streams. Processing a data stream is a challenging task due to the following characteristics of a data stream[5][10],(1) the items of the data stream arrive continuously at high rate,(2) the items can be accessed only once, and (3) the data is unbounded and potentially infinite. Itemset based pattern mining in data streams is a particular case of frequent itemset mining on datasets that include extra challenges.

With a huge growth in both the hardware as well as software that produce huge data in real-time, there has been a need for methods which can process such

huge amounts of data. A straightforward translation of the existing traditional algorithms in data mining is not adequate enough for the stream mining process. As it is not feasible to store all the elements of the data stream in the memory, unlike in static datasets, it is difficult to apply analysis to all data of the data stream at once. Unlike in the case of static dataset where the analysis begins when a query is submitted to the system and ends when the results are found, data streams require continuous processing which is never ending. Data streams are processed by using intermediate summary data structures which store summary information about the elements of the data stream.

Our motivation has been to provide a set of frameworks and algorithms to identify patterns containing hidden knowledge in data streams.

Itemset based pattern mining in data streams is applied to online transactional data mining, sentiment analysis of messages on social media, web-click pattern mining, sensor-data analysis, amongst others[9][13].

1.3 Thesis Contribution

The major contribution of the thesis is the design of intermediate summary data structures and algorithms which use the intermediate summary data structures to mine patterns from data stream. The first contribution is the design of an approach which partitions the data stream into segments, generates frequent itemsets for each of the segments and stores them segment-wise in partitions into the intermediate summary data structure. Each partition stores frequent itemsets for a segment of the data stream. An itemset is present in the partitions corresponding to the segments of the data stream in which it is frequent. The approach estimates the support of an itemset in a partition in which it is not frequent, from the supports of the itemset in the partitions in which it is frequent. The estimated support of the itemset is used in calculating the support for a larger part of the

data stream.

The second contribution is the design of incremental single-pass approach to find closed frequent itemsets from a data stream. By single-pass we mean the algorithm does not require multiple scans of the datasets.

The third contribution is the design of a framework which uses high utility itemset mining to store data stream elements in a compressed form and then detect patterns from the sliding window. This approach promises to reduce the memory requirements when applied to frequent pattern mining in data streams.

The fourth contribution is the design of an approach to find clusters of values within and across attributes in data streams. There are two approaches presented. In the first one, the elements of the data stream are pairs of values of a single attribute. It aims at finding clusters of values of a single attribute in the data stream. In the second approach, the elements of the data stream are pairs of values of two different attributes. It aims at first finding attribute-wise clusters of values and then finds associations between clusters of two attributes.

1.4 Thesis Outline

The thesis consists of the following main chapters. Chapter 2 contains the work done related to the research work presented in this thesis. Chapter 3 discusses our work on frequent itemset mining using support trends in data streams. Incremental closed frequent itemset mining in data stream using single-pass algorithm has been discussed in chapter 4. A framework for high utility pattern mining in data streams using frequent itemset mining has been discussed in Chapter 5. Chapter 6 presents the details of work carried out on cluster analysis of single attribute value in transitional data streams whose elements show transition of an attribute from old value to a new one. The cluster analysis of values across two attributes is presented in Chapter 7. Finally, Chapter 8 and Chapter 9 contain the conclusion

and future work, respectively.

Chapter 2

Related Work

2.1 Frequent Itemset Mining

Frequent itemsets mining was introduced by Agrawal [3] back in the early 90s, and it is used for finding common and potentially interesting patterns in databases. The motivation came from the need to analyze 'supermarket transaction' data, i.e., to examine customer behavior in terms of the purchased products. In this scope, data are represented by means of transactions, each of which is a set of items labeled by a unique ID. The purpose of frequent itemsets mining is to find the most frequently-occurring subsets from these transactions. Together with the introduction of the frequent set mining problem, also the first to solve it was proposed. The algorithm was improved by R. Agrawal and R. Srikant and called Apriori. It uses an iterative approach where k-itemsets are used to explore (k+1)-itemsets. It uses the Apriori property that all nonempty subsets of a frequent itemset must also be frequent. The algorithm which at the first instance generates set itemsets of size one and prunes this set to discard the non-frequent itemsets. The algorithm then generates the set of itemsets of size two, from the set of frequent single sized itemsets, which is also pruned to discard the non-frequent itemsets in them. The process continues till no new frequent itemsets are generated. The algorithm wile

pruning needs to scan the entire dataset to identify the frequent itemsets from the candidate itemsets generated. Thus, Apriori algorithm scans the dataset multiple number of times in order to generate the frequent itemsets. It may also generate a huge number of candidate itemsets.

Solution to this problem was provided by the FP-Growth Algorithm proposed by Han.et.al [12]. FP-Growth mines the set of frequent items without generating the candidate itemsets. It generates a frequent pattern tree called as FP-tree. FP-tree contains information about the itemset associations. FP-growth uses the information contained in FP-tree to generate frequent itemsets. Both the algorithms use horizontal data format. Alternatively data can also be presented in vertical data format. Eclat [11] uses a vertical database format. It extends an itemset prefix until it reaches the boundary between frequent and infrequent item sets and then backtracks to work on the next prefix in lexicographic order. Eclat determines the support of an item set by creating the list of transaction ids containing the itemset. An intersection of two lists of transaction ids of two itemsets is done such that both the itemsets differ by one item is performed. The union of the two itemsets generates the new itemset. The support is the number of transactions ids in the present in the new list obtained after intersection.

2.2 Closed Frequent Itemset Mining in Data Streams

Chi.et.al [9] proposed the algorithm Moment, considered as the first, to find closed frequent itemsets from data streams. It uses an in-memory summary data structure called CET (Closed Enumeration Tree) to maintain a set of itemsets in a sliding window. Moment maintains a large number of nodes in its summary data structure. This is because it stores not only the closed frequent itemsets, but also closed infrequent itemsets potentially to become frequent in subsequent times. Li. Et al. [13] proposed the algorithm NewMoment to mine closed frequent item-

sets from a transactional data stream with a transaction-sensitive sliding window. NewMoment uses a bit-sequence representation of items to lower cost in terms of the time and memory needed to slide the window. It uses the in-memory summary data structure NewCET to generate closed frequent itemsets. NewCET consists of three parts: (1) the bit-sequences of all 1-itemsets in the current transaction-sensitive sliding window; (2) a set of closed frequent itemsets in the transaction-sensitive sliding window; and (3) a hash table to store all closed frequent itemsets with their supports as keys. NewMoment works in three steps: (1) Build; (2) Delete; and (3) Append. At the end of the Build step, NewCET contains a set of nodes containing closed frequent itemsets in the first sliding window. The Delete step removes the oldest transaction from the sliding window. It then reconstructs the NewCET to contain the closed frequent itemsets for the current sliding window. The Append step inserts the incoming transaction of the data stream into the sliding window. It then reconstructs the NewCET to generate the new closed frequent itemsets for the current sliding window. Append is almost the same as Build except that it updates the support of the already existing itemsets in the summary data structure.

2.3 High utility itemset mining

High utility itemset mining has been worked upon by Leeuven at.al. [15] and Yang et.al[17]. in both the approaches, the data stream is divided into batches. After the arrival of all the elements of the batch, the algorithms generate codes for every batch which are then used to compress the elements of the data stream. The approach mentioned in [15] compresses the elements using codes generated based on the occurrences of elements in the previous batch. The approach in [17] compresses elements by dividing the data stream into same sized batches. It finds frequent patterns for each batch and uses these patterns to compress the elements

of the subsequent batches. Both the approaches compress the elements of the entire data stream in batches.

2.4 Attribute Value Clustering

Cheng et. al. [8] have worked on clustering attribute values but not from a data stream point of view. In this paper, they have analyzed the job information from the social network point of view. They first collect the job-related information from various social media sources. Thereafter they construct an inter-company job-hopping network. The vertices denoting companies and the edges denoting the movement of people between companies. They use graph mining techniques to mine groups of related companies. Xu et. al.[16] also have presented a similar problem from a social network model point of view. Both these papers have specifically focused on the work company attribute of the user and aim at find relations between companies from employment point of view. The data is analysed offline. In this paper we provide an approach not limited to the attribute work company. It is applicable to attributes beyond just the one mentioned. The approach enables the user to analyse the transition data in online. Our approach allows the users to specify a minimum threshold value to prevent clustering of weakly associated values together.

Chapter 3

Frequent Itemset mining using Support Trends

3.1 Introduction

The main aim of this module is to generate frequent itemsets from a transactional data stream. A transactional data stream is a data stream, where each element of the data stream is an itemset. The stream is divided into segments called sliding window of a particular size. Frequent itemsets are generated for each sliding window. The frequent itemsets for each sliding window are stored in partitions in an intermediate summary data structure for further processing. Each partition corresponds to a sliding window. A partition contains itemsets which are frequent with their supports known. The itemsets which are not frequent are not stored in the partition. Hence, their support information is lost. The approach estimates the support of an itemset in a partition in which it is not frequent from the supports of the same itemset in the other partitions in which it is frequent. The estimated support is used in calculating the support of the itemset for a larger part of the data stream.

This module attempts to find solution to the problem of mining frequent item-

sets over a given data stream given a fixed sized sliding window and a minimum support threshold.

The major contribution in this chapter is the design of an approach to maintain frequent itemsets over data stream from the beginning till the latest element of the data stream using a sliding window. This approach however suffers from the problem of explosion of large number of frequent itemsets for lower values of minimum support threshold[14] which is resolved using closed frequent itemset mining described in the next chapter.

The study carried out in this chapter is our first experience in the area of data stream with limited resources and knowledge about data stream processing. The datasets used were small. Nevertheless, it gave us a better insight about pattern finding in data streams which helped us overcome the limitations to scale up the study described in the subsequent chapters.

3.2 Related Work

Frequent itemsets mining was introduced by Agrawal[3] back in the early 90s, and it is used for finding common and potentially interesting patterns in databases. In this scope, data are represented by means of transactions, each of which is a set of items labeled by a unique ID. The purpose of frequent itemsets mining is to find the most frequently-occurring subsets from these transactions

Frequent itemset mining on data streams has been proposed in [5][10][6][7]. The study has been done by considering the latest elements of the data stream significant. Whereas, the approach presented in this chapter mines frequent pattern for any part or the whole of the data stream i.e till the latest element.

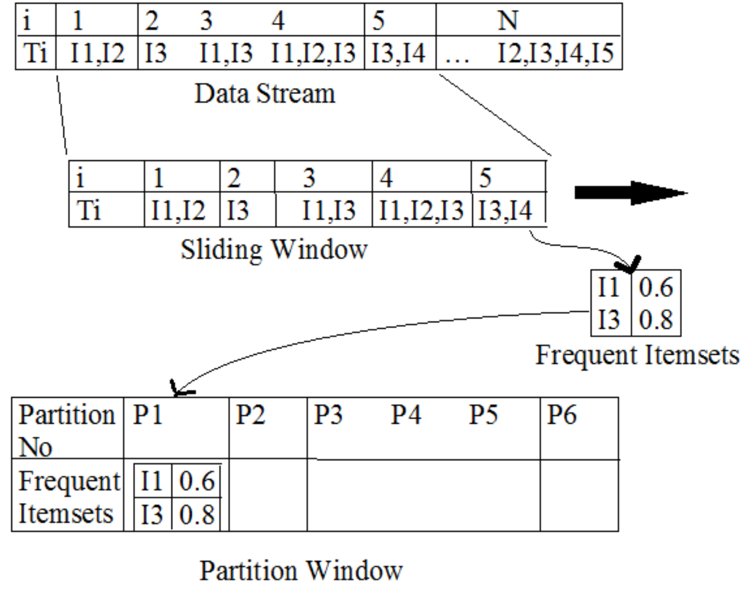


Figure 3.1: The framework showing data stream and partition window

3.3 Problem Definition

3.3.1 Preliminaries

Let D be a data stream of elements where each element is an itemset. Let T_i be the ith element in D. The range $S[a, b]$ denotes a segment of the data stream D having elements belonging to the interval $[a, b]$. Let SW be the sliding window of size w sliding over the data stream D as shown in figure 3.1. Support of an itemset X in this chapter is the ratio of number of elements containing the itemset X upon the total number of elements in the dataset.

3.3.2 Problem Statement

Given data stream D, a sliding window SW of size w and a minimum support threshold s_0, the problem is to mine frequent itemsets over a data stream using the sliding window.

3.4 Framework for Frequent Itemset Mining Using Support Trends

3.4.1 The Intermediate Summary Data Structure

The intermediate summary data structure consist of a partition window PW which is set of partitions $P_1, P_2, ...P_p$, where p is the number of partitions that can be stored in PW. A partition P_i contains frequent itemsets for a segment of the data stream D(Figure 3.1). The partition $P_i[a, b]$ represents the ith partition in PW containing the frequent itemsets of data stream segment $S[a, b]$, where a and b are the ids of the first and the last element of the segment $S[a, b]$ in the data stream. The value of $|P_i|$ denotes the size of segment associated with the partition P_i calculated as

$$|P_i| = b - a + 1 \quad (3.1)$$

3.4.2 Algorithm FIMUST-Frequent Itemset Minimg Using Support Trends

As the elements of the data stream arrive, the first w elements of D are stored in the sliding window SW. At the moment, SW has the elements of the first segment $S[1, w]$ of the data stream. Frequent itemsets are generated by executing the apriori algorithm on SW. The generated frequent itemsets are stored in the partition P_1. In the meanwhile, the next w elements of the data stream are loaded into the sliding window. Frequent itemsets are generated using the apriori algorithm for the second segment, $S[w + 1, 2w]$, which are then stored in the partition P_2. This process is repeated for every sliding window until all p partitions of PW are full. In case all the partitions of PW are full, two oldest partitions representing data stream segments of same size are merged together to

create space for a new partition.

Merging of partitions

Merging of two partitions is done when all the partitions in the partition window are full and there is no partition available to accommodate frequent itemsets of the next sliding window. Let the partitions to be merged be P_i and P_{i+1}. Let P be the partition obtained after merging P_i and P_{i+1}. Partition P will have two kinds of itemsets, the first kind are present in both the partitions P_i and P_{i+1} and the second kind are the ones that are present in only one of the partitions either P_i or P_{i+1}. Both the cases are presented separately in the following sections.

Itemset present in both partitions

Let $X \in P_i, P_{i+1}$ be the itemset present in both partitions. Let $suppP_i(X)$ be the support of X in the partition P_i. Let $|P|$ denote the number of elements in the data stream segment corresponding to the partition P. The support of X in P is calculated as

$$suppP(X) = (|P_i| * suppP_i(X) + |P_{i+1}| * supP_{i+1}(X))/(|P_i| + |P_{i+1}|) \qquad (3.2)$$

Itemset present in one partition

Let the partitions to be merged be P_i and P_j. Let X be the itemset present in one partition P_i. Let $suppP(X)$ be the support of X in the partition P. Let $|P|$ denote the number of elements in the data stream segment corresponding to the partition P. The support of X in P_j is calculated as

$$suppP_j(X) = minsup * suppP_i(X) \qquad (3.3)$$

The term $minsup * suppP_i(X)$ is based on the trend of the itemset X in the partition P_i.

Once value of $suppP_i(X)$ and $suppP_j(X)$ are known, the support of X in P is calculated using equation 3.2.

Selection of partitions for merging

The oldest two adjacent partition representing the smallest sized data stream segments in the partition window are selected for merging. This is because they contain the oldest elements of the data stream which are considered less significant than the latest ones. Both the partitions should correspond to the data stream segments of same size. The time complexity of the 'merging' process is $O(n^2)$.

Generation of frequent itemsets from partitions

The partitions in the partition window have frequent itemsets of their corresponding segments of the data stream. There are two types of itemsets in the partitions. This first type are the ones that are present in all the partition and the second type are the ones that are not be present in some partitions. The supports of the second type of itemsets are not known for the partitions in which they are not present. However, their support in the corresponding segment must be below minsup value. The value of this unknown support for each such itemset is calculate in the following way.

Let X be the itemset which is not present in some partitions. The support of X in the partitions not containing X is calculated as

$$minsup * \sum (suppP_i(X) * |P_i|) / \sum |P_i| \tag{3.4}$$

for all partitions P_i containing X.

The support of X for the part of the data stream represented by the $range[a, b]$

is calculated as

$$\sum(suppP_i(X) * |P_i|)/\sum|P_i|, \forall P_i \tag{3.5}$$

where P_i is the partition in the partition window corresponding to the segments of the data stream in the range $[a, b]$.

3.5 Experimental Study

The above approach was implemented using C++ and executed over synthetic data generated using IBM Synthetic Data Generator [1][3]. The characteristics of these datasets is given in table 3.1.

Dataset	Number of Items	Number of Records
D1	5	1.5K
D2	5	1K

Table 3.1: Parameters of Datasets for FIMUST Algorithm

3.5.1 Error calculation

In this experiment, the size of sliding window was set to 10, number of partitions in PW were 10, and the value of minimum support threshold was set to 0.4. The summary of the results is given in table 3.2. For an itemset, the error in estimated support is calculated by finding the difference between itemset support as calculated by the presented approach and the actual support of the itemsets which is calculated by running the apriori algorithm on the entire data.

Dataset	Number of Frequent Items	Average Error
D1	6	0.05648
D2	5	0.01

Table 3.2: Experimental results, minsup=0.4

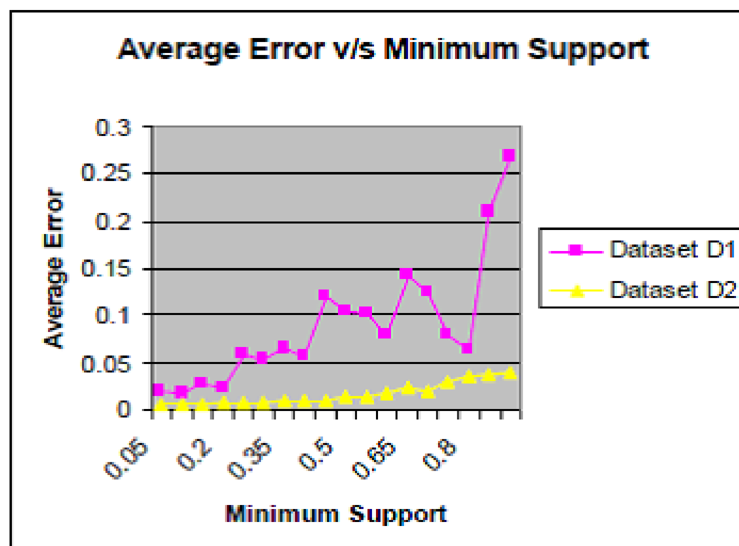


Figure 3.2: Average error in calculated support versus minimum support

3.5.2 Average error versus minimum support

This experiment was performed to observe the change in the average error in the estimated support of the itemsets by varying the value of minimum support. (Figure 3.2).

For lower minimum support values, the average error in the supports is less. The average error in supports increases as the value of minimum support increases. There is an overall increase in the error with increase in minimum support values. However, for small intervals of minimum support the average error is observed to be decreasing. Nevertheless, the average error in support is always less than the minimum support. These trends need not be reflected for all data sets as they depend on the actual data in the data streams.

Two itemsets which were actually frequent were generated as non-frequent by FIMUST algorithm for minimum support value above 0.85 in Dataset D1.

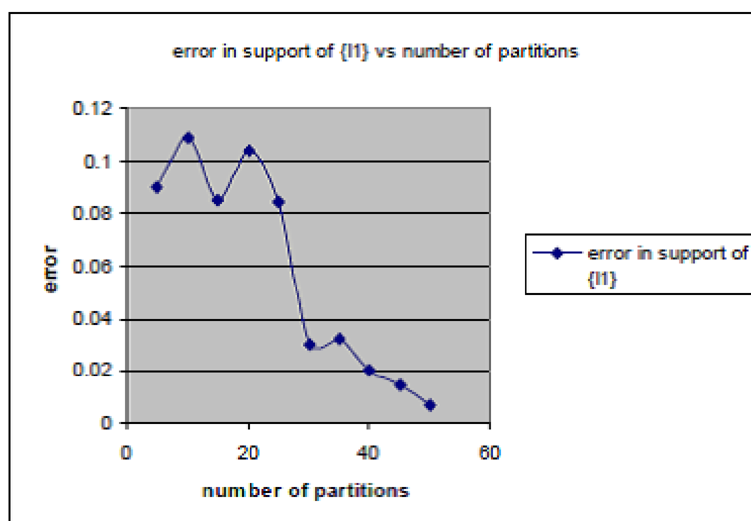


Figure 3.3: Error in calculated support versus number of partitions

3.5.3 Support error versus number of partitions

This experiment was performed to observe the change in error in an itemset support by changing the number of partitions in the partition window (Figure 3.3). The average error in support is very high for small number of partitions. This is because, the frequency of merging of partitions when the number of partitions is small is high. Higher number of merging increases the number of estimates in the support value of an itemset. Whereas, The frequency of merging of partitions reduces as the number of partitions available in the partition window is large. This leads to lesser number of estimates which in turn results in smaller values of average errors.

3.6 Conclusion

Most research efforts in data mining have been focusing upon analysis of static datasets. The approach presented in this chapter is applicable to data streams. We proposed the method to find frequent itemsets in data streams by dividing the data stream into segments of elements that were stored in the memory for

analysis. The frequent itemsets generated for every segment of the stream are stored in the intermediate summary data structure for further analysis.

The intermediate summary data structure is a set of partitions, where each partition contains the frequent itemsets for a segment of the data stream. An itemset frequent in a partition may not be present in another partition as it was not frequent in that segment of the data stream pertaining to the partition in which it is not present. The actual support of the itemset in such a partition is not known and is required while generating the frequent itemsets for the entire stream and while merging two partitions. The support of such an itemset for the segment of the data stream pertaining to the partition in which it is not present is calculated by using a method which uses information about the itemset from those segments (partitions) of data stream that have the itemset as frequent. These estimations introduce some error in the final results.

The major contribution in this module is the new approach to maintain frequent itemsets over data stream evenly throughout the data stream using a sliding window. The approach estimates the support count of an itemset in a partition, in which it is not frequent, from supports of the itemset in the partitions in which it is frequent. The estimated support is used in calculating the support for a larger part of the data stream. As the size of the data stream increases the size of each data stream segment pertaining to the partitions in the partition window increases. It looses the information about the supports of itemsets for smaller parts of the data stream.

This approach however suffers from the problem of explosion of large number of frequent itemsets for lower values of minimum support threshold. This issue is resolved by mining closed frequent itemset as discussed chapter 4.

Chapter 4

Incremental Closed Frequent Itemset Mining using Single Pass Algorithms

4.1 Introduction

In the previous chapter it was observed that the number of frequent itemsets generated for a dataset or in a sliding window was very large, mostly in the case when the value of minimum support threshold was low. A large number of frequent itemset incurs a huge cost in terms memory for storing all the frequent itemsets and requires more time to search for itemsets. In this chapter we have presented an approach to resolve this problem by mining closed frequent itemset.

A closed itemset is an itemset which has no proper superset with similar support [9][13]. A closed itemset is frequent if its support is greater than or equal to the minimum support threshold. The rationale behind generating closed frequent itemsets is that the complete set of frequent itemsets can be generated from the set of closed frequent itemsets and the number of closed frequent itemsets is less than the total number of frequent itemsets.

In this chapter we present an approach to mine closed frequent itemsets from data stream using the sliding window model. The approach is both single pass and incremental. By single pass we mean that it does not require multiple scans of the elements in the sliding window. By incremental we mean that it maintains the itemsets in the intermediate summary data structure without regenerating them for every slide of the sliding window across the data stream.

4.2 Related Work and Motivation

Chi.et.al [9] proposed the algorithm Moment, to find closed frequent itemsets from data streams. It uses CET (Closed Enumeration Tree) as its intermediate summary data structure to generate the closed frequent itemsets. Moment maintains a large number of nodes in its summary data structure. This is because it stores not only the closed frequent itemsets, but also closed infrequent itemsets potentially to become frequent in subsequent times.

Li. Et al. [13] proposed the algorithm NewMoment to mine closed frequent itemsets from a transactional data stream. It uses a transaction-sensitive sliding window to find closed frequent itemsets. Items in the intermediate summary data structure are represented uisng bit-sequences. It uses the in-memory summary data structure called NewCET to generate closed frequent itemsets. NewCET consists of: the bit-sequences of 1-itemsets which are currently present in the transaction-sensitive sliding window; a set of closed frequent itemsets for the transactions currently in the transaction-sensitive sliding window; and a hash table which is used to store the closed frequent itemsets having keys as their supports. NewMoment works in three steps: (1) *Build*; (2) *Delete*; and (3) *Append*. At the end of the *Build* step, NewCET contains a set of nodes containing closed frequent itemsets in the first sliding window. The *Delete* step removes the oldest transaction from the sliding window. It then reconstructs the NewCET to gener-

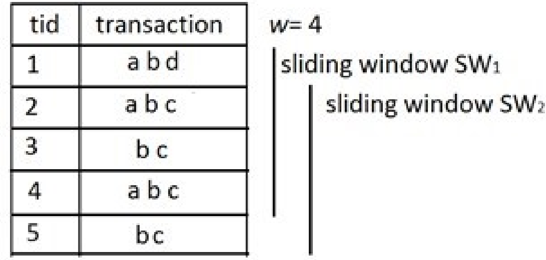
ate the closed frequent itemsets for the current sliding window. The *Append* step inserts the incoming transaction of the data stream into the sliding window. It then reconstructs the NewCET to generate the new closed frequent itemsets for the current sliding window. *Append* is almost the same as *Build* except that it updates the support of the already existing itemsets in the intermediate summary data structure.

NewMoment generates the NewCET tree every time the *Append* and *Delete* steps are performed. Generation of NewCET requires multiple scans of the elements in the sliding window. Hence, a considerable amount of time is required to update and maintain the data in the intermediate summary data structure.

NewMoment uses a hash table which stores the frequent itemsets based on the order of their supports. The hash table makes it easy to check whether an itemset is frequent or not. However, it does not improve the searching of itemsets in the summary data structure because the entire set of the frequent itemsets has to be scanned to search for an itemset in the summary data structure.

In order to generate and update the set of closed frequent itemsets in the intermediate summary data structure, NewMoment uses an approach that requires multiple scans of the sliding window which is time consuming.NewMoment does not allow the user of the system to specify the value of minimum support threshold online.

The approach presented in this chapter does not need to refer to the elements of the sliding window multiple times. Instead, it reads the element only when it arrives and when it leave the sliding window. Not having to have multiple scans of the sliding window saves a considerable amount of time making the presented approach time efficient. The approach updates the closed itemsets by accessing the data in intermediate summary data structure only. This make it more time efficient. It also allows the user to specify the value of minimum support threshold online.



tid	transaction
1	a b d
2	a b c
3	b c
4	a b c
5	bc

Figure 4.1: Data stream and sliding window

The approach outperforms some of the other algorithms for lower values of minimum support in terms of time efficiency. Its efficiency for higher values can be improved by making it to generate only the frequent closed itemsets.

4.3 Problem Definition

4.3.1 Preliminaries

Let $I = \{i_1, i_2, , i_m\}$ be a set of literals called items.

A transaction $T = (tid, x_1, x_2, ..., x_n)$ is an (n+1)-tuple where $X_i \in I$, for $1 <= i <= n$, n is the size of transaction, and tid is the unique identifier of the transaction.

A transactional data stream $D = T_1, T_2, , T_N$ is a sequence of transactions, where N is the tid of the latest transaction T_N (Figure 4.1). An itemset $X = \{x_1, x_2, ..., x_n\}$ is a collection of items where each item $X_i \in I$.

A sliding window SW of size w contains the latest w transactions of the data stream (Figure 4.1).

The support of an itemset X, $sup(X)$, in a sliding window SW is the number of transactions in SW having X as a subset.

An itemset X is frequent if $sup(X) \geq s.w$, where s is a user specified minimum support threshold ($0 \leq s \leq 1$).

An itemset X is closed if it does not have a proper superset with the same

Index

Item	LocationVector
a	
b	
c	
d	

ItemSets

Id	Itemset	Support

Figure 4.2: Intermediate summary data structure

support.

An itemset X is a closed frequent itemset if it is closed and frequent.

4.3.2 Problem Statement

Given a transactional data stream D, a sliding window SW of size w and a minimum support threshold s, the problem is to mine closed frequent itemsets in SW ,i.e., from the latest w elements of the data stream D.

4.4 Framework for Incremental Mining of Closed Frequent Itemsets from Data Streams

4.4.1 The Intermediate Summary Data Structure

The intermediate summary data structure consists of (1) $ItemSets$; and (2)$Index$ (Figure 4.2).

$ItemSets$

$ItemSets$ contains closed frequent itemsets generated for a sliding window. $ItemSets$ is a table with fields: (1) Id; (2) $Itemset$; and (3) $Support$. Id is the identifier of the $Itemset$ in $ItemSets$ table. The field $Support$ is the support of the $Itemset$ in the current sliding window.

Index

Index is a table which is used to search itemsets in *ItemSets* table. It has two fields: *Item*, and *LocationVector*. Each value of the field *Item* belongs to the set of literals, I. The *LocationVector* is a bit-sequence in which the ith bit is set to 1 if the *Item* belongs to the *Itemset* in *ItemSets* table and i is the *Id* of *Itemset* in *ItemSets*. *Index* is very efficient in searching for itemsets in *ItemSets* table.

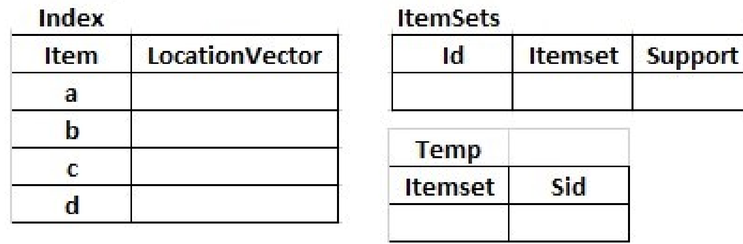
4.4.2 Algorithm SPAIM-CFI-Single Pass Algorithm for Incremental Mining of Closed Frequent Itemsets

The algorithm presented in this chapter works in two steps, The *Insert* step and the *Delete* step. The *Insert* step is executed when a new element of the data stream enters into the sliding window. The *Delete* step is executed when the oldest element leaves the sliding window.

Every element of the data stream is pre-processed with proper word stemming and stop-word removal, if required.

The *Insert* step

This step is performed when a new element enters the sliding window. Let X be the itemset contained in the element that is entering the sliding window. The algorithm searches the *ItemSets* table for the subsets of X and then increases each of their supports each by one. The algorithm uses a temporary list of itemsets called *Temp*. *Temp* is a table having two fields, *ItemSet*, and *SId*. *SId* is the *Id* of the *ItemSet* in the *ItemSets* table which is a superset with highest support as shown in figure 4.3 The algorithm is given in figure 4.4. When a new element containing the itemset X arrives at the sliding window, the algorithm inserts the itemset X into the *Temp* table with *Sid* as 0. Then it searches for the subsets of itemset X in the *ItemSets* table. To do this the algorithm take bit-wise OR



Index

Item	LocationVector
a	
b	
c	
d	

ItemSets

Id	Itemset	Support

Temp

Itemset	Sid

Figure 4.3: Intermediate summary data structure with $Temp$

Input: Datastream D
Output: ItemSets of closed itemsets for SW1
Procedure Insert(Transaction T)

1. *Let X be the itemset in T*
2. *Insert X into Temp with SId 0*
3. *Generate subsets of X from relevant itemsets in ISsets*
4. *For each subset Xi generated itemset with Id I in ItemSets*
   4.1 if Xi not present in Temp
   4.1.1 Add Xi to Temp with SId i
   4.2 else
   4.2.1 replace SId of Xi in Temp with I if Support at Id=I >Support at Id=SId of Xi
5. *For each itemset Xi in Temp with SId i*
   5.1 if Xi=Itemset Xk in ItemSets with Id i
   5.1.2 increase Support of Xk in ItemSets by 1
   5.2 else
   5.2.1 insert Xi into ItemSets
   5.2.2 for each item n belonging to Xi, set the ith bit of Vector of a in Index to 1.

Figure 4.4: Algorithm- $Insert$

of the $LocationVectors$ of each item of X. The positions of the bits having value 1 in the resulting bit-vector will be the locations of the itemsets in the $ItemSets$ table which are subsets of X.

Let $Y \in ItemSets$ with $Id\ i$ be an itemset which is a subset of X. The algorithm finds $X \cap Y$ and inserts it in $Temp$ if it not present in $Temp$. The algorithm inserts $X \cap Y$ into $Temp$ with SId as the Id of Y in the $ItemSets$ table. If $X \cap Y$ is already present in the $Temp$ table with $SId\ k$, then it is assigned the SId as i if support of Y is greater than the support of the itemset with $SId\ k$ in the table $ItemSets$. This step ensures that the itemset $X \cap Y$ in $Temp$ has Sid value which is the Id of the $Itemset$ in the table $ItemSets$ with highest support.

In the next step, the algorithm updates the set of itemsets in the $ItemSets$

table using the data in the *Temp* table. Let X be an itemset with SId i in the *Temp* table. Let Y be the itemset with Id i in the *ItemSets* table. If X and Y are the same sets then the support of Y is increased by one in the *ItemSets* table. Otherwise, X is inserted as a new itemset in the *ItemSets* table. This process is repeated for all the itemsets present in *Temp* table.

Let n be the Id of an itemset X when it is newly inserted into the table *ItemSets*. The nth bits of the $LocationVectors$ of all the itemset belonging to X are set to 1. This helps to quickly search for the itemset X in the *ItemSets* table.

At the end of the *Add* step the contents of the *Temp* table are deleted. In order to prevent from unnecessary expansion of the *ItemSets* table, the algorithm inserts the new itemset into the first available free slot in the *ItemSets* table. The positions of the bits having value 1 in the resulting bit-vector will be the locations of the itemsets in the *ItemSets* table which are subsets of X.

The complexity of the $Insert$ step is $O(n^2)$.

Running example of the *Add* Step

Let the size of sliding window w be 4. Consider the example from Figure 4.1. At the arrival of the first transaction containing the itemset $\{abd\}$, $SPAIM - CFI$ inserts it in *Temp* with SId 0. Since *ItemSets* is empty no intersections are performed. $SPAIM-CFI$ simply inserts itemset $\{abd\}$ into *ItemSets* with Id 1 and *Support*1. The first bits of the $LocationVectors$ of items a,b, and d in $Index$ are set to 1 as the itemset $\{abd\}$ is inserted into *ItemSets* at Id 1 (Figure 4.5). The contents of *Temp* are deleted.

Upon the arrival of the second transaction containing itemset $\{abc\}$,$SPAIM-CFI$ inserts $\{abc\}$ into *Temp* with SId 0. $SPAIM-CFI$ performs bit-wise OR of the $LocationVectors$ of items a,b, and c to find the Ids of itemsets in *ItemSets* containing at least one item from $\{abc\}$.

The only Id is 1 since there is one itemset in *ItemSets*. $SPAIM - CFI$

Index

Item	LocationVector
a	1000
b	1000
c	0000
d	1000

ItemSets

Id	Itemset	Support
0	0	0
1	abd	1

Temp

Itemset	Sid
abd	0

Figure 4.5: Intermediate summary data structure

performs intersection of $\{abc\}$ with $\{abd\}$ to generate $\{ab\}$. This means that itemset $\{ab\}$ which is a subset of $\{abc\}$ may be affected due to the arrival of $\{abc\}$. Since $\{ab\}$ is not present in $Temp$ it is entered into it with SId equal to Id of $\{abd\}$ in $ItemSets$.

$SPAIM-CFI$ inserts $\{abc\}$ from $Temp$ into $ItemSets$ with $Id2$ directly as its SId in $Temp$ is 0. The second bits of the $LocationVectors$ of items a,b, and c in $Index$ are set to 1 as the itemset $\{abc\}$ is inserted into $ItemSets$ at $Id2$. For the second itemset $\{ab\}$ in $Temp$, $SPAIM-CFI$ compares $\{ab\}$ with the $Itemset$ in $ItemSets$ table with Id equal to the SId of $\{ab\}$ in $Temp$ which is 1. The itemset with $Id1$ in $ItemSets$ is $\{abd\}$. Since $\{ab\}$ is not similar to $\{abd\}$, $SPAIM-CFI$ inserts $\{ab\}$ into $ItemSets$ with Id 3. The $Support$ of $\{ab\}$ is calculated by incrementing, the $Support$ of itemset in $ItemSets$ with Id equal to the SId of $\{ab\}$ by one. The SId of $\{ab\}$ is 1. The $Itemset$ in $ItemSets$ at 1 is $\{abd\}$. Hence, $SPAIM-CFI$ calculates the new support of $\{ab\}$ as 2 which is one more than the support of $\{abd\}$. $LocationVectors$ of items in $Index$ are updated accordingly.

Figure 4.6 shows the contents of the summary data structure at the end of sliding window 1.

Upon the arrival of fifth transaction containing $\{bc\}$, $SPAIM-CFI$ inserts $\{bc\}$ into $Temp$ with SId as 0. The Ids generated after performing bit-wise OR of the $LocationVectors$ of items b and c are 1,2,3,4, and 5. $SPAIM-CFI$ performs intersection of $\{bc\}$ with the itemsets in $ItemSets$ table with these Ids generated.

Index	
Item	LocationVector
a	1000
b	1000
c	0000
d	1000

ItemSets		
Id	Itemset	Support
0	0	0
1	abd	1
2	abc	2
3	ab	3
4	bc	3
5	b	4

Figure 4.6: *Index* and *ItemSets*

Index	
Item	LocationVector
a	1000
b	1000
c	0000
d	1000

Temp	
Itemset	Sid
bc	4
b	5

ItemSets		
Id	Itemset	Support
0	0	0
1	abd	1
2	abc	2
3	ab	3
4	bc	3
5	b	4

Figure 4.7: Intermediate summary data structure

Intersection of $\{bc\}$ with $\{abd\}$ is $\{b\}$. Since $\{b\}$ does not exist in $Temp$, it is inserted into it with $SId1$ which is the Id of $\{abd\}$ in $ItemSets$. $SPAIM-CFI$ performs intersection of $\{bc\}$ with the next itemset in $ItemSets$, $\{abc\}$ to get $\{bc\}$. Itemset $\{bc\}$ is already present in $Temp$ with $SId0$. Since the $Support$ in $ItemSets$ at $Id0$ is less than the $Support$ of itemset $\{abc\}$, SId of $\{bc\}$ in $Temp$ is replaced by the Id of $\{abc\}$ in $ItemSets$. $SPAIM-CFI$ performs intersection of $\{bc\}$ with the third itemset $\{ab\}$ in $ItemSets$ to get $\{b\}$. Itemset $\{b\}$ is already present in $Temp$ with $SId1$. Since the $Support$ in $ItemSets$ at $Id1$ is less than the $Support$ of itemset $\{ab\}$, SId of $\{b\}$ in $Temp$ is replaced by the Id of $\{ab\}$ in $ItemSets$. Figure 4.7 shows the contents of the summary data structure after performing intersection with all relevant itemsets of $ItemSets$.

For the first itemset $\{bc\}$ in $Temp$, $SPAIM-CFI$ compares it with the itemset in $ItemSets$ having Id equal to SId of $\{bc\}$ which is 4. Since they both

Index

Item	LocationVector
a	11100
b	11111
c	00010
d	10000

Temp

Itemset	Sid
bc	4
b	5

ItemSets

Id	Itemset	Support
0	0	0
1	abd	1
2	abc	2
3	ab	3
4	bc	4
5	b	5

Figure 4.8: Intermediate summary data structure

are the same *Support* of $\{bc\}$ in *ItemSets* is increased by 1 which is 4. Similarly, for the itemset $\{b\}$, *Support* of $\{b\}$ in *ItemSets* is increased by 1 to be 5 (Figure 4.8).

The *Delete* Step

Let the itemset in the element leaving the sliding window be X. When the element leave the sliding window the supports of only the subsets of X must be decreased by one. In this case some subsets of X in the *ItemSets* table may cease to be as closed itemset. Such itemsets should be deleted from the *ItemSets* table. The algorithm is given in figure 4.9.

The *Delete* step maintains a temporary list of itemsets called $ITemp$. $ITemp$ is a table having three fields, $ItemSet$, SId and $HSSId$. SId is the Id of the itemsets in the *ItemSets* table with support equal to the support of X. $HSSId$ is the Id of the closed superset of X in the *ItemSets* table with the largest support and is not a proper subset of X.

The *Delete* step finds the subsets of X from the *ItemSets* table by using the $LocationVectors$ in the *Index* table. It performs a bit-wise OR operation on the $LocationVectors$ corresponding to the items in the itemset X. It then performs intersection with each of these subsets.

Procedure Delete(Transaction T)

1. Let X be the itemset represented by T
2. Perform bitwise OR of Vectors of items belonging to X to generate PoritionVector(X)
3. For i such that the ith bit in PositionVector(X) is 1
   - 3.1. Generate TempX=X∩Itemset(i)
   - 3.2. If TempX is not present in ITemp
     - 3.2.1. Insert TempX in ITemp
     - 3.2.2. Set SId to i
     - 3.2.3. Set HSSId to j, where j is the Id of the proper superset of TempX which has the highest support
   - 3.3. Else if TempX is present in ITemp with SId s and HSSId h
     - 3.3.1. If sup(Id=s)<sup(Id=i)
       - 3.3.1.1. Set SId of TempX to i
     - 3.3.2. If sup(Id=h)<sup(Id=i) and TempX⊏Itemset(i) then
       - 3.3.2.1. Set HSSId of TempX as i
4. Decrease the supports of itemsets in ItemSets present in ITemp by one
5. For each itemset I in ITemp
   - 5.1. If I is not frequent
     - 5.1.1. Remove I from ItemSets
     - 5.1.2. Update Index accordingly
     - 5.1.3. Is sup(HSSId)=sup(SId) then
       - 5.1.3.1. Remove I from ItemSets
       - 5.1.3.2. Update Index accordingly
     - 5.1.4. Insert Id of I into ISVList

Figure 4.9: Algorithm-*Delete*

Let $Y \in ItemSets$ with Id i be an itemset which is a subset of X. The algorithm finds $X \cap Y$ and inserts it in $Temp$ if it not present in $Temp$. The algorithm inserts $X \cap Y$ into $Temp$ with SId as the Id of Y in the $ItemSets$ table. It assigns the $HSSId$ of $X \cap Y$ as the Id of the itemset in $ItemSets$ table which is a proper superset of $X \cap Y$ with highest support. If $X \cap Y$ is already present in the $ITemp$ table with SId $s$and $HSSId$ h, then the $Delete$ step sets the SId of $X \cap Y$ to i if support of the itemset in $ItemSets$ table with Id s is less than support of Y in $ItemSets$ table. It sets the $HSSId$ of $X \cap Y$ to i if the support of the itemset in $ItemSets$ table with Id h is less than the support of Y in $ItemSets$ table.

The $Delete$ step decreases the supports of all the itemsets in $ItemSets$ table which are present in $ITemp$ table. The $Delete$ step then updates the $ItemSets$ table from the information in $ITemp$ table. Let Y be an itemset in $ITemp$ table with SId as s and $HSSId$ as h. The $Delete$ step removes the itemset with Id value s from the $ItemSets$ table if its support is same as the support of itemset with Id h in the $ItemSets$ table.

The complexity of the $Delete$ step is $O(n^2)$.

Index	
Item	LocationVector
a	1100
b	1111
c	1010
d	0000

ItemSets		
Id	Itemset	Support
1	abc	2
2	ab	3
3	bc	3
4	b	4

ITemp		
Itemset	Sid	HSSid

Figure 4.10: Intermediate summary data structure

Running example of the *Delete* step

Consider the example from Figure 4.1. The deleted transaction contains the itemset is $\{abd\}$.

The contents of $ItemSets$, $Index$, and $ITemp$ are shown in Figure 4.10. The algorithm first finds the bitwise OR of the $LocationVectors$ of items a,b, and d and gets the bit-sequence 1111. This means that an intersection is requires with all the four itemsets in $ItemSets$ table. The intersection of $\{abd\}$ with the first itemset $\{abc\}$ is $\{ab\}$. Since $ITemp$ does not contain $\{ab\}$, it is entered into $ITemp$ table with SId and $HSSId$ values set to 1 and 0, respectively. The intersection of $\{abd\}$ with the second itemset in $ItemSets$ Table $\{ab\}$ is $\{ab\}$. Itemset $\{ab\}$ already exists in $ITemp$ table with SId and $HSSId$ values 1 and 0, respectively. Since support of the itemset with Id 2 is greater than the support of the itemset with Id 1 in the $ItemSets$ table, the SId of $\{ab\}$ in $ITemp$ is set to 2. The algorithm sets the $HSSId$ of $\{ab\}$ to 1 since $\{abc\}$ contains $\{ab\}$ and the support of $\{abc\}$ is the highest among the itemSets containing $\{ab\}$ in $ItemSets$ so far (Figure 4.11).

The algorithm performs intersection of $\{abd\}$ with the third itemset $\{bc\}$. The intersection generates the set $\{b\}$ which is not present in $ITemp$ table. It stores $\{b\}$ into $ITemp$ table with $SId3$ and $HSSId$ as 2 since $\{ab\}$ contains b and has the highest support among the Itemsets containing $\{b\}$ in $ItemSets$ so far. The

ITemp		
Itemset	Sid	HSSid
ab	1	0

after intersection of {ab} with {abc}

ITemp		
Itemset	Sid	HSSid
ab	2	1

after intersection of {ab} with {ab}

Figure 4.11: $ITemp$ - Iteration 1

ITemp		
Itemset	Sid	HSSid
ab	2	1
b	3	2

after intersecton of {ab} with {bc}

ITemp		
Itemset	Sid	HSSid
ab	2	1
b	4	2

after intersection of {ab} with {b}

Figure 4.12: $ITemp$ - Iteration 2

algorithm then performs intersection of $\{abd\}$ with $\{b\}$ to generate $\{b\}$ which is already existing in $ITemp$ table (Figure 4.12). The contents of $ItemSets$, $Index$ and $ITemp$ are shown in Figure 4.13.

The supports of $\{ab\}$ and $\{b\}$ are decreased by 1 (Figure 4.14). Itemset $\{ab\}$ in $ITemp$ table has SId and $HSSId$ values as 2 and 1 respectively. From the $ItemSets$, support at Id 2 is same to support at $Id1$. Hence, itemset $\{ab\}$ is not closed itemset. Itemset $\{ab\}$ is removed from $ItemSets$. The first bits in Vectors of a and b are set to 0.

For the second itemset $\{b\}$ in $ITemp$ table, the support values for $Id4$ and 2are different. Therefore, itemset $\{b\}$ is closed frequent, so it is retained in $ItemSets$ as shown in Figure 4.15.

Index	
Item	LocationVector
a	1100
b	1111
c	1010
d	0000

ItemSets		
Id	Itemset	Support
1	abc	2
2	ab	3
3	bc	3
4	b	4

ITemp		
Itemset	Sid	HSSid
ab	2	1
b	4	2

after intersection of {ab} with {b}

Figure 4.13: Intermediate summary data structure

Index	
Item	LocationVector
a	1100
b	1111
c	1010
d	0000

ItemSets		
Id	Itemset	Support
1	abc	2
2	ab	2
3	bc	3
4	b	3

ITemp		
Itemset	Sid	HSSid
ab	2	1
b	4	2

Figure 4.14: Intermediate summary data structure

Index	
Item	LocationVector
a	1100
b	1111
c	1010
d	0000

ItemSets		
Id	Itemset	Support
1	abc	2
2		
3	bc	3
4	b	3

ITemp		
Itemset	Sid	HSSid
ab	2	1
b	4	2

Figure 4.15: Intermediate summary data structure

4.5 Experimental Study

Experiments were performed to compare the performance of SPAIM-CFI approach with NewMoment algorithm. The specifications of the system on which the experiments were done are, 2.26GHz Intel Core i3 Processor, 3 GB RAM, and Windows 7 OS. The proposed algorithm is implemented using C++. The compiler used was GNU GCC compiler. The synthetic dataset was generated using IBM Synthetic Data Generator[1][3]. The parameters of the dataset generated are given in table 4.1.

Parameter	Value
Number of transactions	200K
Average items per transaction	10
Number of items	200

Table 4.1: Parameters of Datasets for Closed Frequent Itemset Mining

Mining for different size of sliding window

This experiment is performed by changing the sliding window size w from 10K to 100K. The value of minimum support threshold is set to 0.2.For smaller sizes of sliding windows, the time required was less.There has been an increase in the execution time per sliding window in both the algorithms. This is because of the increase in the number of data stream elements processed in a sliding window.

Mining for different values of minimum support

This experiment is performed by varying the minimum support threshold s from 0 to 1. In case of NewMoment the execution time is more for lower values of s and decreases for higher values of s. This happens because of the decrease in the number of closed frequent itemsets generated by NewMoment. Figure 4.16 and figure 4.17 show that the proposed approach requires less memory and time for a

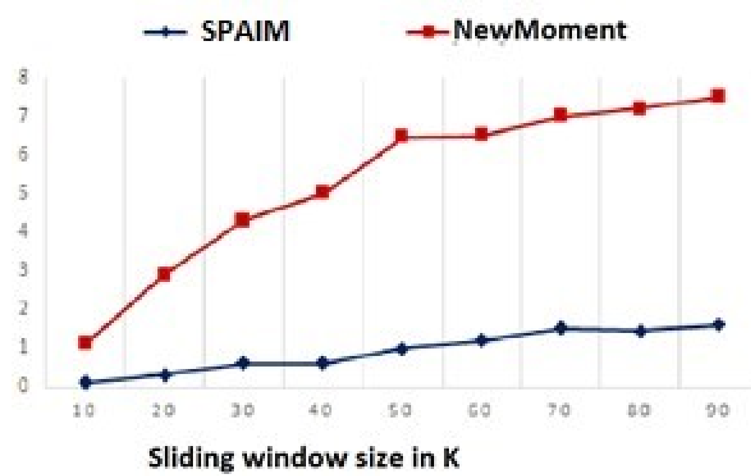


Figure 4.16: Memory required versus minimum support

Figure 4.17: Execution time required versus minimum support

transition of a sliding window as compared to NewMoment. These observations are done by taking average of 50 transitions.

However, the results shown in figure 4.18 have also shown that the NewMoment outperforms the proposed approach for higher values of minimum support threshold as NewMoment maintains only the frequent closed itemsets, whereas, the proposed approach maintains all the closed itemsets.

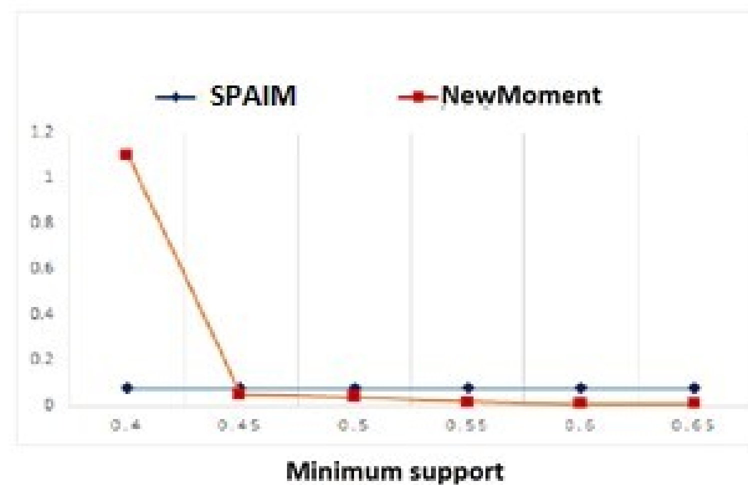


Figure 4.18: Execution time required versus minimum support

4.6 Conclusion

In this section we have proposed an incremental approach to mine closed frequent itemsets over transactional data streams. It uses an intermediate summary data structure which is efficient in terms of searching of itemsets stored in it. The proposed approach outperforms the NewMoment algorithm in mining closed frequent itemsets over transactional data streams.

The NewMoment algorithm outperforms the proposed approach for higher values of minimum support threshold as NewMoment maintains only the frequent closed itemsets, whereas, the proposed approach maintains all the closed itemsets. However, the difference in time requirements is negligible.

The proposed algorithm has the following limitation. The size of $LocationVector$ depends upon the number of itemsets stored in the intermediate summary data structure. it is difficult to predict this vale a priori.

Chapter 5

Framework for High Utility Pattern Mining using Closed Frequent Itemsets

5.1 Introduction

The number of active users on microblogging sites is large and so are the messages posted by them. This data can be used to mine knowledge such as the current topic of discussion between the users. Finding answers to such queries is challenging due to the large amount of data generated.

In this chapter we consider such data streams where each of its element is compressed using high utility itemset mining method. High utility itemset mining associates a profit value, called as utility, to an itemset. The utility of an itemset could be profit on retail price in case of superstore or retail market datasets, frequency of its occurrence, etc.

In this chapter we define utility in terms of the amount of memory saved by storing the itemset in a compressed form in the sliding window. This approach reduces the amount of memory required to store the itemsets in a sliding window

which in turn reduces the size of the sliding window.

5.2 Related Work

High utility itemset mining has been worked upon in [15] and [17]. According to the approach presented in [15], the data stream is divided into batches of elements. For each itemset in a batch, codes are generated based on the occurrences of itemsets in that batch. The elements of every batch are then compressed using the codes generated. For a current batch the compression is based on the codes of itemsets generated in the previous batch. Using codes of a previous batch to compress the elements of current batch is not appropriate as itemsets need not show similar trends in all batches. The approach in [17] compresses elements by dividing the data stream into same sized batches. It finds frequent patterns for each batch and uses these patterns to compress the elements of the subsequent batches. Both approaches compress the elements of the entire data stream in batches.

In this chapter we propose a framework which incrementally compresses the elements in a sliding window. By incremental we mean that compression is done while an element enters or leaves the sliding window unlike in [15] and [17] where itemsets are compressed at the end of a batch. The compression of elements is based on the utility of the elements in the current sliding window. The approach presented in this chapter is suitable for processing data stream using a sliding window model as it allows more elements to be stored in a sliding window for a fixed memory budget.

5.3 Problem Definition

5.3.1 Preliminaries

Let $D = (T_1, T_2, ...)$ be a data stream where T_i is an itemset. Let SW be a sliding window, of size w, which slide over the data stream D. The support of an itemset X, denoted as $supp(X)$, is the number of element containing X in the sliding window SW.

The idea is to replace the itemsets in elements of the sliding window by links to the itemsets themselves stored as closed itemsets in the intermediate summary data structure. Not all the itemsets in the sliding window are replaced but the ones which are chosen based their utility in the sliding window. The method to find the utility of an itemset is described in the subsequent subsections.

5.3.2 Utility of an itemset

Utility of an itemset is the profit value associated with the itemset. In this chapter we define utility of an itemset based on two factors. The first factor is the amount of memory saved by replacing the itemset in the sliding window by a link. This is calculated as the difference between the amount of memory required to store the itemset and the size of the link to the itemset. The second factor is the frequency of occurrence of the itemset in the sliding window, which is the support of the itemset.

The utility of an itemset X, denoted as $utility(X)$, is calculated in as follows. If v_i is the size of an item in X, then size of X is calculated as

$$v(X) = \sum v_i, \forall v_i \epsilon X \quad (5.1)$$

If γ is the memory required by X in summary data structure, then $utility(X)$ is

Data Stream	
tid	Itemset
1	{a,b}
2	{a,d,e,f}
3	{a}
4	{b,c}
.	.
.	.

ItemList			
Id	ItemSet	Support	Utility

Figure 5.1: Data stream and the intermediate summary data structure

given by

$$utility(X) = support(X) * (v(X) - \gamma) - v(X) \tag{5.2}$$

where $v(X)$ is the size of X and is memory size of X. The term $u(X) - \gamma$ is the memory saved by compressing X once. The term $support(X) * (v(X) - \gamma)$ is the memory saved in compressing $X support(X)$ number of times in the sliding window. $utility(X)$ is memory saving offered by X if compressed. If $utility(X) > 0$ then X should be compressed and not otherwise.

5.4 Framework forHigh Utility Itemset Mining using Closed Frequent Itemsets

5.4.1 The Intermediate Summary Data Structure

Intermediate summary data structure is used to store the temporary results generated while processing the data stream. The intermediate summary data structure presented in this chapter is a table *ItemList*. *ItemList* has three fields- *Id*, *Support*, *ItemSet* and *Utility* as shown in the 5.1.

5.4.2 The approach

The approach uses the algorithm described in chapter 4 to generated closed itemsets. When an itemset X is inserted into $ItemList$, the approach calculates the value of $u(X) - \gamma$ and uses it to calculate $utility(X)$ which is then stored in $ItemList$ table.

The approach then selects the itemsets to be replaced by the link pointing to them in the $ItemList$ table. This selection is based on the utility values of the itemsets. Itemsets with utilities above zero can be replaced.

5.4.3 Automatic generation of the minimum support threshold s_0

Most of the approaches, including the ones presented in this thesis, use minimum threshold values that are specified by the users. Users have no idea about the distribution of data in sliding window and in datasets, and are not in a position to specify a proper value of minimum support threshold. This is mostly true in the case of data streams produced by social websites and micro-blogging websites. A lower value of minimum support threshold may lead to generation of large number of frequent patterns which are trivial. Similarly, a high value of minimum support threshold may prevent frequent patterns from being shown as frequent. It is a good idea to provide the user with a suitable value of minimum support threshold based on the data present in the sliding window or in the dataset. However, the user can use his/her discretion to specify the value of minimum support threshold based on the one specified by the system. In this section we present a method to dynamically generate the value of minimum support threshold based on the data in the sliding window or the dataset.

This approach finds the mean and the standard deviation of supports of all the itemsets in the summary data structure. The sum of mean and the standard

deviation of supports of all itemsets is assigned to s_0.

The value of s_0 is defined as

$$s_0 = \frac{\sum supp(X)}{N} + \delta, \forall X \epsilon Isets \tag{5.3}$$

where N is the number of itemsets in $ItemList$ and δ is the standard deviation of the supports of all itemsets in $ItemList$ at that moment.

Deciding which pattern is interesting is a matter of discretion of the user. However, the user can specify his/her own minimum support threshold based on dynamically generated value of s_0.

5.4.4 Detection of frequent events in the sliding window

The approach generates closed frequent itemsets from the set of closed itemsets in the intermediate summary data structure by comparing their supports with the the minimum support threshold s_0.

5.5 Experiments

The experimental study was performed on system with 2.26GHz Intel Core i3 processor, 3 GB memory and Windows 7 OS and implemented in C++. The compiler used is GNU GCC compiler.

5.5.1 Frequent Itemset Mining using Itemset Utility

The data set used for these experiments is generated using IBM Synthetic Data Generator [1][3] and is a synthetic dataset. The parameters of the dataset are mentioned in table 5.1.

Table 5.1: Dataset Parameters

Parameter	Value
Number of transactions	210K
Average items per transaction	10
Number of items	250

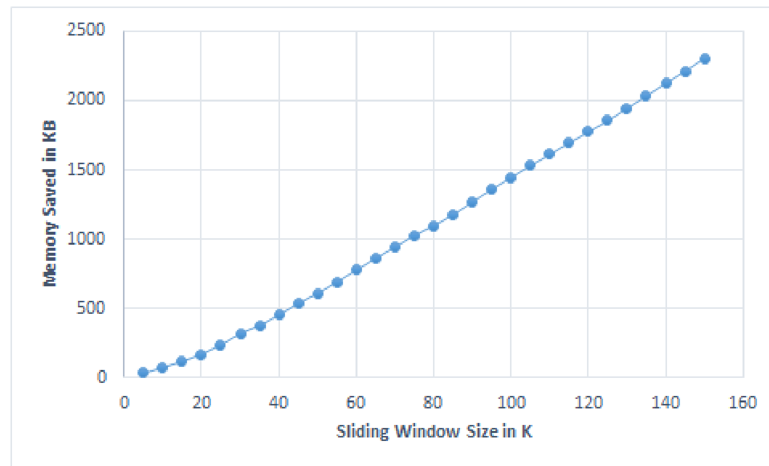


Figure 5.2: Memory saved against sliding window size

Varying sliding window size

The sliding window size was changed from $5K$ to $15K$ with intervals of $5K$. Figure 5.2 depicts that the amount of memory saved in storing the elements of sliding window is increases with the sliding window size increases.

Fixed sliding window size

The sliding window was slided over the data stream by one element. The amount of memory saved was observed for each sliding window for 100 transitions. Figure 5.3 depicts that the amount of memory saved for sliding windows in the beginning is less and increases to become almost stable later. This can be explained as the algorithm learns about the itemset utility at the beginning. Thus, the number of itemsets replaced(compressed) is less.

The amount of memory saved by our approach depends upon the kind of data in the sliding window. The results of the proposed algorithm need not be promising

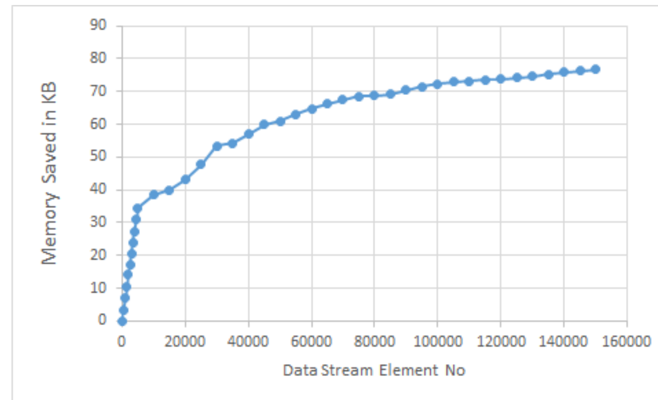


Figure 5.3: Memory saved as window slide over data stream

1	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over
2	1	1	Kolkata Knight Ric	Royal Challenge	1	1	SC Ganguly	BB McCullum	P Kumar	0
3	1	1	Kolkata Knight Ric	Royal Challenge	1	2	BB McCullum	SC Ganguly	P Kumar	0
4	1	1	Kolkata Knight Ric	Royal Challenge	1	3	BB McCullum	SC Ganguly	P Kumar	0
5	1	1	Kolkata Knight Ric	Royal Challenge	1	4	BB McCullum	SC Ganguly	P Kumar	0
6	1	1	Kolkata Knight Ric	Royal Challenge	1	5	BB McCullum	SC Ganguly	P Kumar	0
7	1	1	Kolkata Knight Ric	Royal Challenge	1	6	BB McCullum	SC Ganguly	P Kumar	0
8	1	1	Kolkata Knight Ric	Royal Challenge	1	7	BB McCullum	SC Ganguly	P Kumar	0
9	1	1	Kolkata Knight Ric	Royal Challenge	2	1	BB McCullum	SC Ganguly	Z Khan	0
10	1	1	Kolkata Knight Ric	Royal Challenge	2	2	BB McCullum	SC Ganguly	Z Khan	0
11	1	1	Kolkata Knight Ric	Royal Challenge	2	3	BB McCullum	SC Ganguly	Z Khan	0
12	1	1	Kolkata Knight Ric	Royal Challenge	2	4	BB McCullum	SC Ganguly	Z Khan	0
13	1	1	Kolkata Knight Ric	Royal Challenge	2	5	BB McCullum	SC Ganguly	Z Khan	0
14	1	1	Kolkata Knight Ric	Royal Challenge	2	6	BB McCullum	SC Ganguly	Z Khan	0
15	1	1	Kolkata Knight Ric	Royal Challenge	3	1	SC Ganguly	BB McCullum	P Kumar	0
16	1	1	Kolkata Knight Ric	Royal Challenge	3	2	SC Ganguly	BB McCullum	P Kumar	0
17	1	1	Kolkata Knight Ric	Royal Challenge	3	3	SC Ganguly	BB McCullum	P Kumar	0

Figure 5.4: Dataset- IPL Cricket matches

always. It shall work best if the number and size of frequent itemsets are large.

5.5.2 Frequent pattern generation using dynamically generated s_0

The data set used in this experiment is a real dataset available on the website [2] The dataset is a set elements where each element is a set words and is ball-by-ball description(commentary) of 636 matches in IPL. These elements were processed to have only significant information alike matchid, team details, bowler, batsman, runs scored, etc. There were 150460 elements in the data stream. Figure 5.4 depicts a glimpse of the dataset.

It was observed that 48798 itemsets were generated. The value assigned by the approach to s_0 was 0.596009. Out of 48798 patterns 7268 patterns were frequent for $s_0 = 0.596009$.

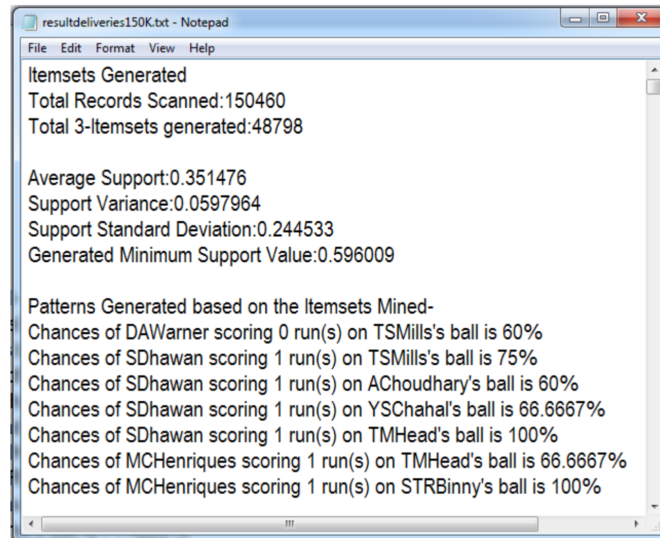


Figure 5.5: Snapshot of frequent patterns

A snapshot of the patters obtained in experimental study is shoqn in Figure 5.5.

5.6 Conclusion

The major contribution in this section is the design of an approach that stores element of a sliding window in a compressed form and then detects events for the sliding window using the data from summary data structure.

The replacement of the itemsets in the data stream by the link to the itemset stored in the intermediate summary data structure may generate false positive results. This could be avoided by generating closed frequent itemsets.

This approach also detects itemsets in the sliding window by dynamically generating a suitable value for the minimum support threshold. This estimated value of minimum support threshold gives an idea about the data distribution in the sliding window and helps the user to specify his/her own minimum support threshold value. This approach is useful in detecting events from posts on micro-blogging websites.

Chapter 6

Clustering Values of Single Attribute in Transitional Data Streams

6.1 Introduction

Social media websites have gained immense popularity among people since the advancement of internet technology. The influx of people into social media websites is further greater among the younger generation who do not want to be left out from the emerging technology. To be part of the social media website, there is a need to create a user profile to uniquely identify the individual belonging to the website. User profiles often contain a set of attributes pertinent to their personal, educational, and/or other relevant data. It is often seen that users update their profiles by changing the values of these attributes. The change is mostly in terms of replacing an old value by a new one. This change to an attribute in the form of a transition from an old value to a new one can be mined to extract valuable pieces of information which further can aid in better decision making.

For example, when a user changes his/her work company attribute value to a

new one, it can be learnt that both the new value and the replaced old value represent names of the companies belonging to the same sector or domain. Processing this information from different user profiles can help find knowledge patterns in the form of clusters of companies offering similar kinds of jobs. This information can be used by employer's recruitment departments to locate companies which may have the human resource required by them. Also, this information can be used by job aspirants and employees to locate companies offering recruitment opportunities in their domain.

Similarly, studying the change in values of address attribute can help identify migratory patterns of humans between places. Also, the analysis of contact number attribute can help to identify the switching patterns of people across telecom service providers.

Extending this study from a single attribute to two or more attributes can help identify relations between values across different attributes. For example, analysis across two attributes, Educational Institute and Work Company, can find patterns containing information about the educational institutes and the companies in the same domain. This information can be used by students to locate companies offering job opportunities in the area related to their studies. This information can also be used by companies to locate educational institutes from which students passing out can be absorbed by them.

Due to the large number of users on social media, the number of updates done to user profiles is also large. Let an update done to a single attribute by a user to his/her profile be considered as an event. The frequency of occurrence of such events is high. These events can be modelled as elements of a data stream of which every element is a pair of values of the attribute. All elements of the data stream consist of values which belong to a single attribute. The data stream in this case will have its elements as a pairs of values of an attribute in which the first value is the old one and the second value is the new one replacing the old. We call such

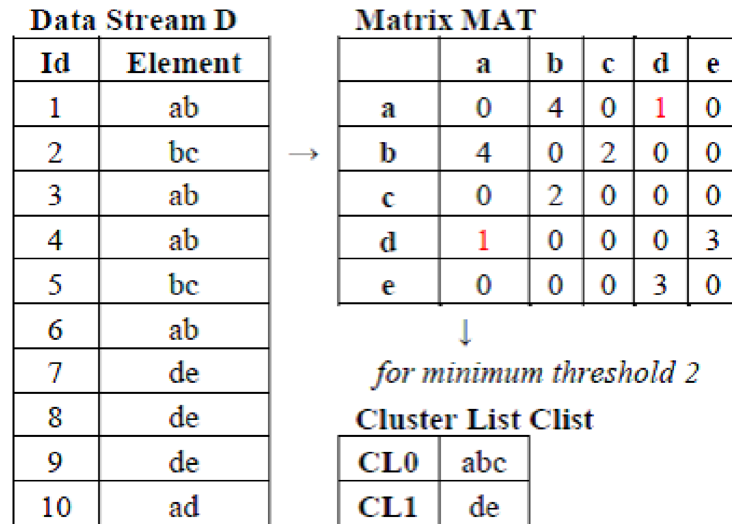


Figure 6.1: The framework

a data stream as a transitional data stream. Every element of a transitional data stream denotes a transition in an attribute from an old value to a new one.

In this chapter we present an approach to cluster values of attributes similar to each other from a transitional data stream and the results of the experimental study done.

The approach is based on two frameworks. The first framework is the market basket analysis approach that is used in finding frequent itemsets of values from the transitional data stream. Each itemset has two items. The second framework is the clustering framework that is used to group the values of the attribute. The approach can be made more intelligent by incorporating text mining and semantic analysis methods. In this chapter we have limited our study to attribute values only. The problem and the approach presented in this chapter is demonstrated in figure 6.1.

Let the attribute under consideration be A. Let $\{a, b, c, d, e\}$ be the set of values that can be assigned to the attribute A. As shown in the figure 6.1, the data stream D has ten elements. Matrix MAT is the intermediate summary data structure used to maintain summary of results as the elements are processed. $CList$ stores

the clusters as they are generated. As the elements are processed, the intermediate summary data structure, matrix MAT, is updated. The algorithms described in the subsequent sections are executed on matrix MAT to generate clusters which are stored in $CList$. The approach generates two clusters, $CL_0 = \{a, b, c\}$ and $CL_1 = \{d, e\}$ for a minimum threshold of 2. A minimum threshold value is specified to enforce the condition of generating quality clusters.

This is a recent area of research and to the best of our knowledge no such approach has been proposed. However, a study on network transition analysis has been performed in an offline manner [8][16].

6.2 Background and Motivation

A similar problem has been worked upon but not from a data stream point of view in [8]. The study is based on social network point of view. Job related information about the users was collected from various social media websites and this information was used to construct an inter-company job hopping network graph. In the graph, the vertices denote companies and the edges denote the movement of the people between these companies. Further, graph mining techniques were used to mine cluster of companies related to each other. A similar work has also been presented in [16].

The study carried out in both the papers, [8] and [16] has been specifically on the work place attribute of people with the aim to find out clusters called as talent circles of companies offering job opportunities pertaining to a particular domain. The analysis is done on a dataset in static environment which cannot be directly translated to be applied onto a data stream environment.

In this chapter, we present an approach that is not limited to a specific attribute. It enables the user to analyze data in online and incremental manner. It allows the user to prevent grouping of weakly associated values together by

specifying a minimum threshold value s_0.

6.3 Problem Definition

6.3.1 Preliminaries

Let the attribute of study be A. For example, A could be the *workcompany* of a person. Let $V = \{v_1, v_2, ..., v_n\}$ be the set of values that could be assigned to the attribute A. For example, $V = \{Technostar, Digiworld, Protech, ...\}$ is the set of company name. An element T is a pair of values, (v_o, v_n), where value v_o is replaced by v_n for the attribute A and $v_o, v_n \epsilon V$. For example, if a person on her/his profile replaces the value of his/her work place from $Technostar$ to $Digiworld$, then this event generates the pair $(Technostar, Digiworld)$ as an element of the data stream with v_o as $Technostar$ and v_n as $Digiworld$. When a person updates an attribute on her/his user profile on social media an element of this form gets created. The study presented in this chapter is limited to un-directional associations between the values, i.e., the elements $(Technostar, Digiworld)$ and $(Digiworld, Technostar)$ are considered as the same. Incorporation of information about the directions of transitions of attributes across values will yield more knowledgeable patterns.

We define similarity between two values of an attribute, denoted as $sim(v_1, v_2)$, based number of elements containing both the values in it. Two values v_1 and v_2 are similar, denoted as $v_1 \sim v_2$ if $sim(v_1, v_2) >= s_0$, where s_0 is a user specified minimum support threshold. This means that both v_1 and v_2 should belong to the same cluster.

If $v_1 \sim v_2$, and $v_2 \sim v_3$, then $v_1 \sim v_3$, i.e. all three are similar. As v_1 and v_2 are similar, they both belong to same cluster. Let the cluster to which they both belong be C. Similarly, as v_2 and v_3 are similar, they v_3 has to belong to the same cluster C which also contains v_1. Now since both v_1 and v_3 belong to the same cluster C, they are similar.

A cluster C is a set of similar values, i.e. $C = \{v_i/all\ v_is\ are\ similar\}$. For example, if a large number of persons update their user profiles by changing work place from *Technostar* to *Digiworld* and a large number of persons update their user profile by changing work company from *Digiworld* to *Protech*, then the values *Technostar*, *Digiworld* and *Protech* are similar to each other and belong to the same cluster. In this example, the cluster has names of companies offering similar job. A data stream D is a stream of such elements, i.e. $D = \{T_1, T_2, , T_n\}$. Two elements T_1 and T_2 of the same data stream are same if they both have same sets of values.

The approach presented here is based on sliding window model.

6.3.2 Significance of s_0

It is quite possible that few elements in a data stream have values which may be completely from a different domain and should not be grouped together. These elements represent those values that hardly have any transitions happening between them. The value of s_0 prevents the grouping of values which are not related to each other. For example, a very few persons changing their work company by hopping between companies of completely different domains. These companies if grouped together could be misleading and should not be grouped together.

Nevertheless, this information can be mined to find out hidden patterns that would reflect exceptions in job transitions and would help user discover knowledge useful to people who would switch over to new fields in their careers. However, such a pattern mining process is out of the scope of the study presented in this chapter.

6.3.3 Problem Statement

For an attribute A with set of possible values $V = \{v_1, v_2, ...\}$, a data stream D consisting of elements $T = (v_o, v_n)$, $v_o, v_n \in V$, the problem is to generate clusters containing values similar to each other.

6.4 The Proposed Approach

6.4.1 The intermediate summary data structure

The intermediate summary data structure has two parts, a matrix MAT and a list of clusters $CList$ as described in the following subsections.

MAT

MAT is the square matrix of size $n * n$, where n is the number of attribute values in V. The rows and columns of MAT represent the values in V. The value of $MAT[v_i][v_j]$ denotes the number of data stream elements containing both the values v_i and v_j. Matrix MAT represents an un-directional weighted graph in which the vertices represent the values of attribute, the edges represent the transition between values, and the weights on the edges represent the similarity between the values joined by the corresponding edge.

MAT can be implemented in two forms, as an array and as a list. If implemented as an array it is required that the list of all the values possible for the attribute be known earlier. Whereas, when implemented as a list it allows a new value to be added dynamically to the list of values V. No sooner does a new value is found the first time in the data stream, then it is added to V and is listed in MAT. We have implemented the algorithm using an array.

CList

CList is a set of attribute value clusters which are generated as the data stream elements are processed.

The intermediate summary data structure also maintains s bit-vector $B = (b_1, b_2, ..., b_n)$, where n is the number of values in V. The value of b_i is set to 1 if v_i belongs to some cluster in *CList*, otherwise it is set to 0.

6.4.2 The Algorithm

The algorithm works in two steps, *Update* and *Generate*. The *Update* step updates the matrix *MAT* as the new element enters and the old element leaves the sliding window, i.e. when the sliding window slides across the data stream by one element. The *Generate* steps uses the data in matrix *MAT* to generate and maintain the clusters *CList*. The *Generate* step can be executed at any time by the user.

The *Update* step

The *Update* step is performed when the sliding window slides across the data stream by one element. As the oldest element leaves the sliding window and a new one enters it, the *Update* step updates the matrix *MAT* in the way described in the next two subsections. The *Update* step is made up of two steps, the *Add* step which is performed when a new element enters the sliding window, and the *Remove* step which is performed when the oldest element leaves the sliding window.

The *Add* step

This step is performed when a new element arrives at the sliding window. When a new element $T = (v_i, v_j)$ enters in the sliding window the *Add* step increases the values of both $MAT[v_i][v_j]$ and $MAT[v_j][v_i]$ by one each as shown in 6.2.

Data Stream D

Id	Element
1	ab
2	bc
3	ab
4	ab
5	bc
6	ab
7	de
8	de
9	de
10	ad

$\rightarrow$

Matrix MAT

	a	b	c	d	e
a	0	3	0	0	0
b	3	0	2	0	0
c	0	2	0	0	0
d	0	0	0	0	0
e	0	0	0	0	0

Matrix MAT for the sliding window having first 5 elements

Figure 6.2: Matrix MAT

Data Stream D

Id	Element
1	ab←
2	bc
3	ab
4	ab
5	bc
6	ab
7	de
8	de
9	de
10	ad

$\rightarrow$

Matrix MAT

	a	b	c	d	e
a	0	**2**	0	0	0
b	**2**	0	2	0	0
c	0	2	0	0	0
d	0	0	0	0	0
e	0	0	0	0	0

Matrix MAT for the sliding window having second 4 elements i.e after removing {ab}.

Figure 6.3: Matrix MAT

The *Remove* step

The *Remove* step is performed when the oldest element leaves the sliding window. When the element $T = (v_i, v_j)$ leaves the sliding window the *Remove* step decreases the values of both $MAT[v_i][v_j]$ and $MAT[v_j][v_i]$ by one each as shown in 6.3.

Input: *MAT,so*
Output: *CList={CL1,CL2,...}* set of clusters

1. For every row value *vi* of matrix *MAT*, *vi*ϵ*V*, and *vi* not already assigned to any cluster
   a. *Co={vi};n=0*
   b. *S(Cn)={vs/vs* is similar to some value in *Cn*,i.e *MAT(vs,v)>=so* for all *v*ϵ*Co*.}
   c. *Cn+1=CnUS(Cn);n++;*
   d. Goto step b if *Cn-1*≠*Cn*
   e. Add *Cn* to *CList*

Figure 6.4: Algorithm- *Generate*

Matrix MAT

	a	b	c	d	e
a	0	3	0	0	0
b	3	0	2	0	0
c	0	2	0	0	0
d	0	0	0	0	3
e	0	0	0	3	0

Figure 6.5: The matrix MAT

The *Generate* Step

The *Generate* step generates attribute value clusters based on the data in matrix MAT and stores them in $CList$. The time complexity of the algorithm is of $O(n^2)$. The algorithm is given in figure 6.4.

The *Generate* step can be called at any time by the user by specifying the value of minimum support threshold s_0.

Running example for the *Generate* step

The execution of the above algorithm is demonstrated by generating clusters based on the matrix MAT as shown in figure 6.5. Let the value of s_0 be 2. The first row attribute value of MAT is a. Hence, $C_0 = \{a\}$ and makes $B[a] = 1$. Then it finds $S(C_0) = S(a) = \{b\}$, since b is the only element in the row a with $MAT(a, b) = 3 >= s_0$. The next step finds $C_1 = C_0US(C_0)$, i.e., $C_1 = \{a\}U\{b\} = \{a, b\}$ and

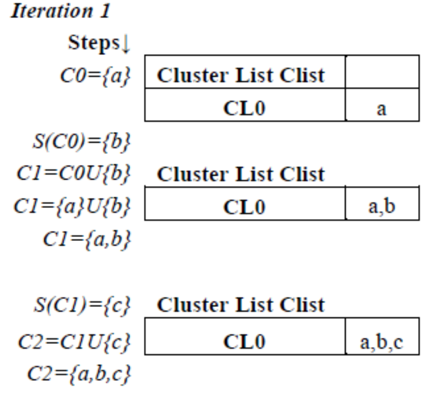


Figure 6.6: Iteration 1 of the *Generate* step

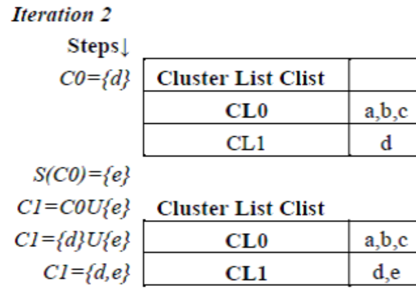


Figure 6.7: Iteration 2 of the *Generate* step

makes $B[b] = 1$. Since $C_0 C_1$, the algorithm finds $S(C_1) = S(\{a, b\}) = \{c\}$, since c is the only new element related to either a or b, i.e., $MAT(b, c) = 2 >= s_0$. Thus $C_2 = C_1 US(C_1)$, i.e., $C_2 = \{a, b\}U\{c\} = \{a, b, c\}$ and makes $B[c] = 1$. Similarly $C_3 = \{a, b, c\}$. Since $C_3 = C_2$, it adds C_3 as a cluster in $CList$ as CL_0 (Figure 6.6) and proceeds to the next row b.

Since b and c are already included in the cluster the algorithm moves to the row d. Since d is not included in any cluster, $C_0 = \{d\}$. Repeating the steps that were applied to the previous row a, the algorithm find $C_2 = \{d, e\}$ and adds it to $CList$ as CL_1. Meanwhile, $B[d]$ and $B[e]$ are set to 1 (Figure 6.7).

Parameters	Values
Number of elements in data stream	2000K
Number of values in the attribute	10

Table 6.1: Parameters of Dataset for Attribute Value Clustering Experiment

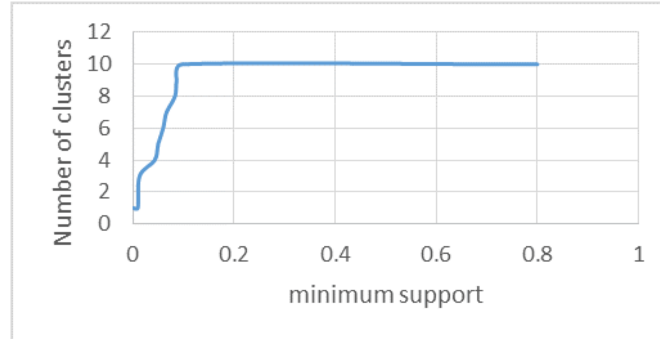


Figure 6.8: Number of clusters versus minimum support

6.5 Experimental Analysis

The experiments were carried on 2.26GHz Intel Core i3 PC with 3 GB memory on Windows 7 system. The proposed approach was implemented in C++ and compiled using GNU GCC compiler.

The experiments were performed on synthetic data generated using IBM Synthetic Data Generator [1][3]. The generated dataset had itemsets of different sizes. The dataset was pruned to discard itemsets of size other than two. The parameters of the dataset are given in the table 6.1. The experiment was performed using the sliding window model approach to determine the number of clusters by varying the value of minimum support threshold from 0 to 1. This range is obtained by dividing the value of the minimum support threshold s_0 by the size of the sliding window. The size of the sliding window was kept to 10K (Figure 6.8).

The number of clusters increases as the value of minimum support threshold increases. For lower values of minimum support threshold, the number of clusters is less. This is because the attribute values with low with low similarity between them, but higher than the minimum support threshold value are grouped together. As the value of minimum support threshold increases, more attribute

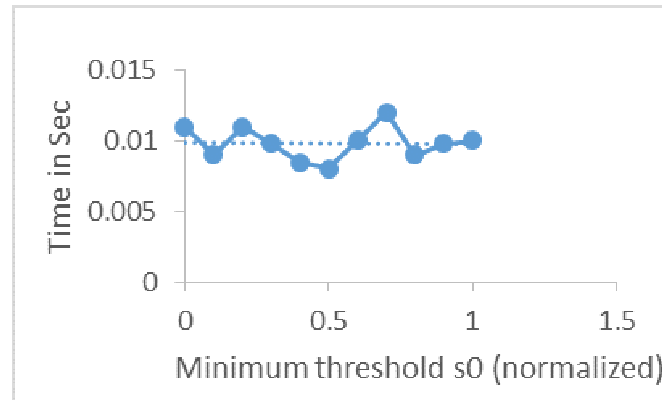


Figure 6.9: Execution time versus minimum support

values become dissimilar from each other as the similarity between them begins to fall below the minimum support threshold. Figure 6.9 shows the time required to generate the clusters by varying the value of s_0. The number of data stream elements analyzed at this time is 2000K.

6.6 Conclusion

In this chapter we proposed framework and algorithms to analyze data streams having elements which are pair of values of a single attribute. These attribute values are clustered together. These values can be objects like company names, educational institute name, names of places, etc. In this approach we have considered values of attributes only. Intelligent mining can be done by incorporating more data about the values themselves.

The study is limited to a data stream with one attribute. In the next chapter we have extended the study to data streams with two attributes

Chapter 7

Mining Associations between Clusters of Values of Multiple Attributes in a Data Stream

7.1 Introduction

In chapter 6 we presented an approach to cluster values of a single attribute in a data stream. There the criteria for two values to belong to a cluster was based the number of occurrences of both the values together in the sliding window. In this chapter, we present an approach extended to two attributes. A data stream element is a pair of values where each value belongs to a different attribute. The approach generates clusters of values for each attribute and then finds association rules between two clusters of different attributes.

For example, let us consider two attributes educational-institute and work-place of people. This approach when applied to both the attributes will first generate clusters for the attribute educational-institute having names of institutes that offer education in similar domain. Similarly, the clusters generated for the attribute work-place will have a name of companies offering jobs in domains related

to each other. The approach then generates association rules between clusters from the educational institute attribute and clusters from workplace attribute. Each association rule has two clusters in it, one containing names of educational institutes and the other containing name of companies. Such an association is useful to find out the institutes and the companies in similar domains.

As already discussed in the previous chapter, many persons have their user profiles created on social media website. A user profile often consists of a set of attributes. As the number of users on social media is huge, the number of updates done to their user profiles is large. It may be recalled that the number of attributes considered in this study is two. Let the update done by a user by assigning a new value to any one or both the attributes of study be called as an event. This event creates a pair of two values each belonging to the attributes of study. The frequency of such events is high. The sequence of such events can be modeled as a data stream in which every element of the data stream is a pair of updated values each belonging to different attributes.

As mentioned earlier in this section, we present an approach to cluster values for each attribute and then find associations between two clusters of different attributes. The approach is based on two frameworks. The first framework is clustering analysis that is used to group values of attributes. The second framework is the market basket analysis used to find associations between clusters. The approach can be made more intelligent by incorporating text mining and semantic analysis methods. In this chapter we have limited our study to attribute values only.

The problem and the approach presented in this chapter is briefly explained in figure 7.1. Let the attributes of study be A_1 and A_2. Let $V_1 = \{a, b, c\}$ and $V_2 = \{u, v, x, y, z\}$ be the sets of values that can be assigned to the attributes A_1 and A_2, respectively. As shown in the figure the data stream D has ten elements. The approach uses an intermediate summary data structure which stores

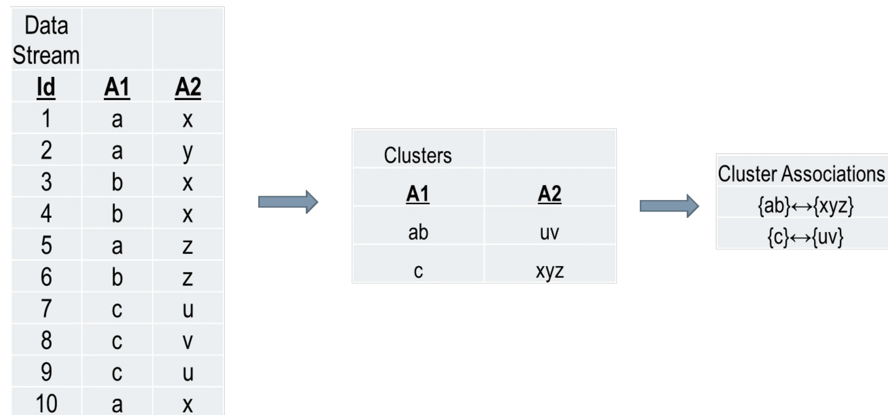


Figure 7.1: Data stream, clusters and cluster associations

the clusters as they are generated for each attribute. As the elements are processed, the intermediate summary data structure is updated. The algorithms described in the subsequent sections are executed on the data in the intermediate summary data structure to generate clusters which are stored as a list of clusters. The approach generates two clusters $\{a, b\}$ and $\{c\}$ for the attribute A_1 and two clusters $\{x, y, z\}$ and $\{u, v\}$ for the attribute A_2, for a minimum threshold of 2. A minimum threshold value is specified to enforce the condition of generating quality patterns. The approach then executes algorithms described in the following sections which generate associations $\{a, b\} \leftrightarrow \{x, y, z\}$ and $\{c\} \leftrightarrow \{u, v\}$. This is a recent area of research and to the best of our knowledge no such approach has been proposed. However, a study on network transition analysis has been performed in an off-line manner.

7.2 Background and Motivation

A similar study has been presented by us in the chapter 6. The study was limited to clustering of values of a single attribute in a data stream. Whereas, the work carried out in this chapter clusters values of two attributes and find relations between two clusters of different attributes.

7.3 Problem Definition

7.3.1 Preliminaries

Let A_1 and A_2 be the attributes of study. Let $V_1 = \{v_{11}, v_{12}, , v_{1n}\}$ and $V_2 = \{v_{21}, v_{22}, ..., v_{2m}\}$ be the sets of values for an attribute A_1 and A_2, respectively. For example $V_1 = \{GMC, KLE, ...\}$ is a set of educational institutes and $V_2 = \{Govt.Hospital, Vision, ...\}$ is a set of organizations employing people.

An element T is a pair of values (v_{1i}, v_{2j}) where $v_{i1} \in V_1$ and $v_{2j} \in V_2$. For example the element $T = (GMC, Vision)$ means the person has studied in GMC school and works for $Vision$ organization.

Two values $v_{11} \in T_1$ and $v_{12} \in T_2$ are similar if there exists $T_i = (v_{11}, v_{2i})$ and $Tj = (v_{12}, v_{2j})$ where $v_{2i} = v_{2j}$ and $i \neq j$. If v_{11} is similar to v_{12} and v_{12} is similar to v_{13} then v_{11}, v_{12} and v_{13} are similar. For example, if $(GMC, Vision)$ and $(KLE, Vision)$ are elements of the data stream, then GMC and KLE are similar. The similarity between two values v_{11} and v_{12}, denoted as $sim(v_{11}, v_{12})$, is the total count of elements containing both v_{11} and v_{12}.

The set $C = \{v_{ji}/all\ v_{ji}s\ are\ similar\}$ is a cluster of similar values of attribute A_j. For example, $\{GMC, KLE\}$ is a cluster as GMC and KLE are similar.

A data stream $D = \{T_1, T_2..., T_n\}$ is a stream of events. Two elements T_1 and T_2 are same if they represent same sets of values. Two values $v_{11} \in T_1$ and $v_{12} \in T_2$ are heavily similar if the number of $T_i = (v_{11}, v_{2i})$ and $T_j = (v_{12}, v_{2j})$ where $v_{2i} = v_{2j}$ and $i \neq j$, are not less than a minimum threshold value s_0 i.e. $sim(v_{11}, v_{12}) >= s_0$. The value of s_0 is decided by the user. The significance of s_0 is that it prevents the clustering of values together, which hardly have any transitions happening between them, which is quite often a situation in the real world. For example, a person changing joins a company not related to the type of study done at school. Such elements should be ignored so as to nullify their effect on the results. Such transitions seldom happen and are less in number. A

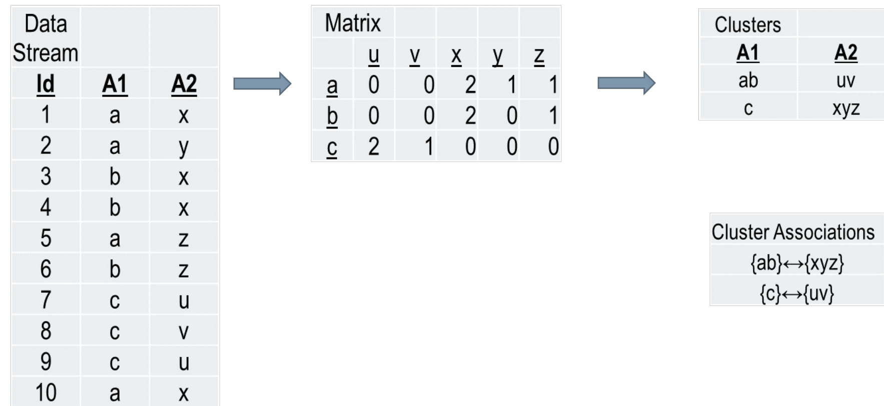


Figure 7.2: Intermediate summary data structure

minimum support threshold will restrain such values from clustering together.

Let C_{1i} and C_{2j} be clusters of values of attributes A_1 and A_2, respectively. The clusters C_{1i} and C_{2j} are associated with each other if there exists $v_{1a} \in C_{1i}$ and $v_{2b} \in C_{2j}$ such that $sim(v_{1a}, v_{2b}) >= s_0$, where $sim(v_{1a}, v_{2b})$ is the total number of elements containing both v_{1a} and v_{2b}.

7.3.2 Problem Statement

For two attributes A_1 and A_2, a set of values $V_1 = \{v_{11}, v_{12}, ...\}$ and $V_2 = \{v_{21}, v_{22}, ...\}$, a data stream $D = \{T_1, T_2, ...\}$ consisting of elements element $T = (v_{1i}, v_{2i})$, where $v_{1i} \in V_1$ and $v_{2i} \in V_2$, the problem is to generate a set of clusters of values for each attribute and association rules between two clusters, each belonging to attribute A_1 and attribute A_2.

7.4 The Approach

The intermediate summary data structure

The intermediate summary data structure presented in this chapter has two parts, a matrix MAT and two lists of clusters $CList_1$ and $CList_2$ for the attributes A_1 and A_2, respectively (Figure 7.2).

Matrix MAT

MAT is a matrix whose rows represent the values of V_1 and columns represent the values of V_2.The value of $MAT[v_i][v_j]$ represent the total number of data stream elements having both the values v_{1i} and v_{2j} in it.

List of clusters $CList_1$ **and** $CList_2$

$CList_1$ and $CList_2$ are sets of clusters of values of attributes A_1 and A_2, respectively.

7.5 The Algorithm

The algorithm presented in this chapter works in three steps. They are the *Update* step, the *Generate_Cluster* step, and the *Generate_Association* step.

The Update step

This step is executed when the sliding window slides across the data stream. It updates the matrix MAT when a new element enters and the oldest element leave the sliding window. It works in two steps, the *Add* step and the *Remove* step.

The *Add* step is performed when a new element enters the sliding window. When a new element (v_i, v_j) enters the sliding window, the *Add* step increases the value of $MAT[v_i][v_j]$ by one.

The *Remove* step is performed when the oldest element leaves the sliding window. For an element (v_i, v_j) leaving the sliding window, the *Remove* step decreases the value of $MAT[v_i][v_j]$ by one.

The *Generate_Cluster* **step**

This step is executed to generate clusters from the matrix MAT. First, it generates clusters for each row in MAT in the following way. For a row in MAT,

Matrix							Steps-->	
	u	v	x	y	z		A1	CL1↓
a	0	0	2	1	1		xyz	xyz
b	0	0	2	0	1		xz	
c	2	1	0	0	0		uv	uv
↓Steps								
A2	c	c	ab	a	ab			
CL2-->	c		ab				Cluster Associations	
							{ab}↔{xyz}	
							{c}↔{uv}	

Figure 7.3: $Generate| - Cluster$ step example

it clusters the values of the columns which are having the matrix value greater than or equal to s_0 and store them in $CList_1$. That is, for a row v_{1i} in matrix MAT, the Generate_Cluster step will form the cluster $\{v_{2s}/MAT[v_{1i}][v_{2s}] >= s_0\}$. Thereafter, the clusters in $CList_1$ having at least one common element are merged together. Clusters for the other attribute are obtained in the similar way.

The Generate step is demonstrated using a running example. The value of s_0 is set to 1. In the first row a, the cluster generated is $C_2 = \{x, y, x\}$ as $MAT[a][x]$, $MAT[a][y]$, and $MAT[a][x]$ are greater than or equal to s_0. In the second row b, the cluster generated is $C_2 = \{x, z\}$ and for the third row the cluster generated is $C_3 = \{u, v\}$. Since $C_0 \cap C_1 = \{x, z\}$, the clusters C_0 and C_1 are merged. Hence, $CL_1 = \{\{x, y, z\}, \{u, v\}\}$. The above algorithm can be executed by the user at any time by specifying the minimum threshold value s_0 (Figure 7.3).

The time complexity of the $Generate_Cluster$ step is $O(n^2)$.

The $Generate_Association$ step

This step generates associations between clusters of different attributes. Two clusters $C_1 \in CL_1$ and $C_2 \in CL_2$ are associated if $sim(v_1, v_2) >= s_0$, $v_1 \in C_1$ and $v_2 \in C_2$. The association between two clusters C_1 and C_2 is denoted as $C_1 \leftrightarrow C_2$ (Figure 7.4).

Matrix							Steps-->	
	u	v	x	y	z		A1	CL1↓
a	0	0	2	1	1		xyz	uvxyz
b	0	0	2	0	1			
c	2	1	1	0	0		uv	
↓Steps								
A2	c		ab					
CL2-->	abc						Cluster Associations	
							{abc}↔{uvxyz}	

Figure 7.4: *Generate_Association* step example

Parameters	Values
Number of elements in data stream	2000K
Number of attributes values per attribute	10

Table 7.1: Parameters of Datasets for Cluster Association Generation

The time complexity of the *Generate_Association* step is $O(n^2)$.

7.6 Experimental Analysis

Experiments were performed to check the efficiency of the proposed algorithm on two data sets, synthetic and a real data set. All experiments were performed on a system with 2.26GHz Intel Core i3 processor, 3 GB memory and Windows 7 operating system. The algorithms were implemented in C++ language and was compiled with GNU GCC compiler.

The synthetic data were generated using IBM Synthetic Data Generator [1][3]. The synthetic dataset parameters are mentioned in Table 7.1.

These experiment were performed on the above data sets using the sliding window model approach to determine the number of clusters by changing the value of minimum threshold. The size of the sliding window was kept to 10K in both the cases

The experiment in figure 7.5 was performed on synthetic data set by varying

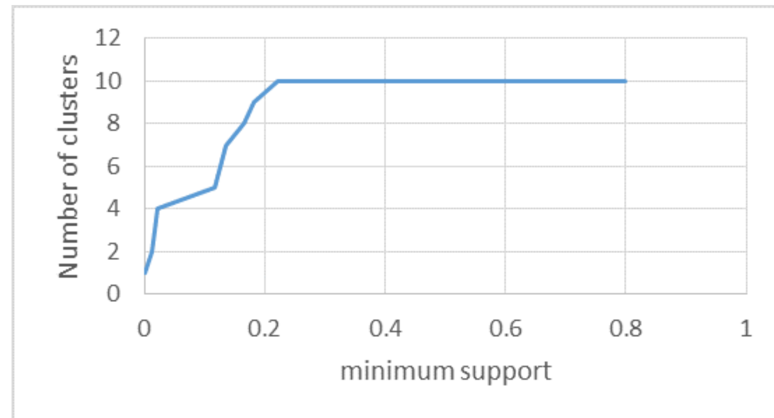


Figure 7.5: Number of clusters versus minimum support

the value of minimum support threshold from 0 to 1. This range is obtained by dividing the value of the minimum support threshold s_0 by the size of the sliding window. The clusters shown contain values of only one attribute. The number of clusters increases as the value of minimum support threshold increase. For lower values of minimum support threshold, the number of clusters is less because values that even have low similarity between them, but higher than the minimum support value, are grouped together. As the value of minimum support threshold increases more and more values become dissimilar as the similarity between the values begins to fall below the minimum support threshold.

The main objective of performing the above experiments was to check the accuracy, precision and recall of the algorithm. This was done by varying the value of minimum support. For every value of minimum support the clusters were generated for values of each attribute. Thereafter, associations between two clusters, each of different attribute were generated. The generated clusters and associations between them were compared with results obtained by applying apriori algorithm to the real data set to find the accuracy, precision and recall separately for the attributes and the associations. The results are as below.

The experiment in figure 7.6 was performed by varying the value of minimum support from 0 to 1. The precision value for the associations varied with upper

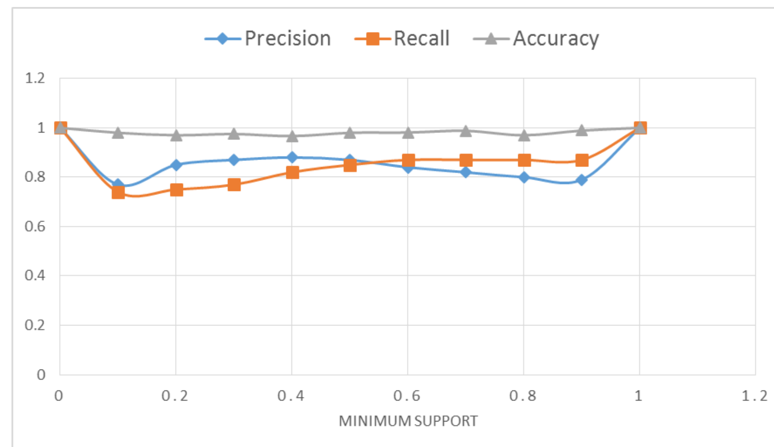


Figure 7.6: Precision, recall and accuracy of clusters

bound as 0.88 and lower bound as 0.77. The value recall value varied between 0.74 and 0.87. The recall value increased consistently with the value of minimum support and stayed stable after 0.6 in 87%, while the accuracy was always above 96%.

7.7 Conclusion

We proposed algorithms and an approach for analysis data of streams having its element as pair of values of two different attributes. The values of an attribute were clustered to generate groups of attribute values related to each other and then find association between clusters of values across different attributes. These values can be objects like company names, educational institutes, places, etc.

We have limited our study by considering only the attribute values. Data about the values themselves can be incorporated into the algorithms to generate better patterns.

The study is limited to a data stream pertaining to two attributes only. It can be applied to data streams with multiple attributes. As there is no similar work done in this area a comparative study was not possible and experiments were performed to check the precision, recall and accuracy of the algorithm.

Chapter 8

Conclusion

Data stream mining is very interesting problem and can be applied to online transactional data mining, sentiment analysis of messages on social media, web-click pattern mining, sensor-data analysis,etc.

The process of mining patterns from data streams is challenging due to the inherent characteristics of data streams. The major ones being the unbounded size of the data stream and the inability to have multiple scans or revisit the entire history of the data stream.

In our study we have used sliding window to find patterns from data streams. The results of this processing were stored in intermediate summary data structure. The data from the intermediate summary data structure was used to generate patterns of interests. The patterns were mostly in the form of itemsets. The final results had an error component added to them as the patterns were generated from the summary data and not from the original data in the data streams.

Itemset mining generally requires multiple scans of the datasets. In data stream processing it is not possible to have multiple scans of the data. Hence, the major focus of this research work was on the development of single pass and incremental algorithms to mine patterns by extending itemset mining to data streams.

We began with partitioning the data stream into segments and generating fre-

quent itemsets for each segment. The frequent itemsets generated were stored in the intermediate summary data structure. This data was used to generate frequent itemsets for the entire or part of the data stream. This approach suffered from the problem of loosing itemset information for smaller segments of the data streams as the size of the data stream increased. The number of itemsets generated for each segment was large. This problem was addressed by proposing a single pass incremental algorithm which generated closed frequent itemsets for the data streams. This algorithm used a data structure called $PositionVector$ to speed up the process of searching itemsets stored in the intermediate summary data structure. Experiments showed that this algorithm outperformed some other algorithms like NewMoment[13] for lower values of minimum support. This algorithm faced the challenge of choosing the appropriate size of the $PositionVector$.

The later part of our study was more from application point of view. In this part we proposed a framework and an algorithm which used utility of an itemset to compress the size of the sliding window by replacing the itemsets in the sliding window, with high utilities, by pointers. These pointers pointed to the same itemsets but stored only once in the intermediate summary data structure. This algorithm promises good results for datasets having itemsets which are large in size and occur large number of times.

The process of pattern mining generates a large number of patterns out of which not all are of actual interest to the users. Users are generally interested in frequent patterns. Pattern generation can be restricted only to frequent patterns using a minimum support threshold value which is usually specified by the user. This value should be carefully specified in order to generate the most appropriate set of frequent patterns. Users have no sufficient clue about the data in the data stream to be able to specify the appropriate value of minimum support threshold. Hence we proposed an approach to dynamically generate the value of minimum support by calculating the mean and standard deviation of supports of itemsets

in the intermediate summary data structure. Users can specify their value of minimum support threshold using the value generated dynamically.

In the next part of the study we proposed an algorithm to cluster values of an attribute in a data stream. The data stream considered had pairs of values belonging to a same attribute as its elements. Two values were clustered together based on their occurrence as pairs in an element of the data stream. The same algorithm was then extended to two attributes to generate clusters of values for each attribute at the first instance and then generate association rules out of these clusters.

All the algorithms were implemented using C++ and experiments were performed on both synthetic and real data sets.

Itemset based pattern mining in data streams is an open research problem having high scope for further improvements by incorporating techniques like text mining, fuzzy-logic, etc.

Chapter 9

Future Work

The work done in our study can be extended in the future in the following directions- The FIMUST algorithm proposed in chapter 3 generates itemsets using the Apriori algorithm which requires multiple scans of the sliding window. These itemsets can be generated using single pass algorithms mentioned in chapter 4.

The algorithm proposed in chapter 4 uses static size for $PositionVector$ to increase its search efficiency. It would be worth exploring the effect of dynamic sized $PositionVectors$ on the efficiency. The algorithm also stores the non-frequent closed itemsets in the intermediate summary data structure. There is a scope to work upon an algorithm to generate frequent itemsets without storing the non-frequent itemsets.

The algorithm proposed in chapter 5 compresses the itemsets to reduces the size of the sliding window. Our study was only limited to identifying the amount of space saved using the method proposed. Exploring the ways of proper utilization and benefits of this methods, other than generating frequent itemsets, is proposed to be studied in the future.

The algorithms in chapters 6 and 7 are applied to one and two attributes respectively. There is a scope for extending the study to multiple attributes. Incorporating added information about the values themselves shall yield patterns

with more knowledge.

List of Publications

1. Naik SB, Pawar JD (2012) Finding frequent item sets from data streams with supports estimated using trends. J Inf Oper Manage 3(1):153

2. Naik SB, Pawar JD (2013) An efficient incremental algorithm to mine closed frequent itemsets over data streams. In: Proceedings of the 19th International Conference on Management of Data, COMAD13, Mumbai, India, India. Computer Society of India, pp 117120

3. Naik SB, Pawar JD (2015) A quick algorithm for incremental mining closed frequent itemsets over data streams. In: Proceedings of the Second ACM IKDD Conference on Data Sciences, CoDS 15, New York, NY, USA. ACM, New York, pp 126127

4. Naik, S. B., Pawar, J. D. (2017, May). Clustering attribute values in transitional data streams. In Computing, Communication and Automation (ICCCA), 2017 International Conference on (pp. 58-62). IEEE.

5. Naik, S. B., Pawar, J. D. (2017, June). Mining association rules between values across attributes in data streams. In Computational Intelligence in Data Science (ICCIDS), 2017 International Conference on (pp. 1-6). IEEE.

6. Naik, S. B., Pawar, J. D. (2017, July). A single-pass algorithm for incremental mining patterns over data streams. In 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT) (pp. 565-569). IEEE.

7. Naik S.B., Pawar J.D. (2019) Frequent Itemsets in Data Streams Using Dynamically Generated Minimum Support. In: Kulkarni A., Satapathy S., Kang T., Kashan A. (eds) Proceedings of the 2nd International Conference on Data Engineering and Communication Technology. Advances in Intelligent Systems

and Computing, vol 828. Springer, Singapore

8. Naik S.B., Pawar J.D. (2019) Framework for High Utility Pattern Mining using Dynamically Generated Minimum Support Threshold. International Journal of Engineering and Technology(UAE)

Bibliography

[1] https://ibm-quest-synthetic-data-generator.soft112.com.

[2] www.kaggle.com.

[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[4] Nora Alkhamees and Maria Fasli. Event detection from social network streams using frequent pattern mining with dynamic support values. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 1670–1679. IEEE, 2016.

[5] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[6] Joong Hyuk Chang and Won Suk Lee. Decaying obsolete information in finding recent frequent itemsets over data streams. *IEICE transactions on information and systems*, 87(6):1588–1592, 2004.

[7] Joong Hyuk Chang and Won Suk Lee. Finding frequent itemsets over online data streams. *Information and Software Technology*, 48(7):606–618, 2006.

[8] Yu Cheng, Yusheng Xie, Zhengzhang Chen, Ankit Agrawal, Alok Choudhary, and Songtao Guo. Jobminer: A real-time system for mining job-related patterns from social media. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1450–1453. ACM, 2013.

[9] Yun Chi, Haixun Wang, Philip S Yu, and Richard R Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 59–66. IEEE, 2004.

[10] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S Yu. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212, 2003.

[11] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[12] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[13] Hua-Fu Li, Chin-Chuan Ho, and Suh-Yin Lee. Incremental updates of closed frequent itemsets over continuous data streams. *Expert Systems with Applications*, 36(2):2451–2458, 2009.

[14] Shankar B Naik and Jyoti D Pawar. Finding frequent item sets from data streams with supports estimated using trends. *Journal of Information and Operations Management*, 3(1):153, 2012.

[15] Matthijs Van Leeuwen and Arno Siebes. Streamkrimp: Detecting change in data streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 672–687. Springer, 2008.

[16] Huang Xu, Zhiwen Yu, Jingyuan Yang, Hui Xiong, and Hengshu Zhu. Talent circle detection in job transition networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 655–664. ACM, 2016.

[17] Xintian Yang, Amol Ghoting, Yiye Ruan, and Srinivasan Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 370–378. ACM, 2012.