CHAPTER-1

INTRODUCTION

Rapid advances in data collection and storage technology have enabled organizations to accumulate vast amount of data. Simple transactions of everyday life such as using a credit card, a phone or browsing the web lead to automated data storage. Similarly, advances in information technology have lead to large flows of data across IP networks. In many cases, these large volumes of data can be mined for interesting and relevant information in a wide variety of applications. When the volume of the underlying data is very large, it leads to a number of computational and mining challenges. Traditional data analysis tools and techniques cannot be used because of massive size of a data set and hence there is need to develop new methods. Data mining is a technology that blends traditional data analysis methods with sophisticated algorithms for processing large volumes of data. Currently, a large class of data-intensive applications, in which data is in the form of continuous streams, has been widely recognized. Not only is the size of the data for these applications unbounded, but the data arrives in a highly bursty mode.

1.1 Data Stream Model

In the data stream model, some or all of the input data that are to be operated on are not available for random access from disk or memory, but rather arrive as one or more continuous data streams. Data streams differ from the conventional stored relation model in several ways [12]:

1. The data elements in the stream arrive online.
2. The system has no control over the order in which data elements arrive to be processed.
3. Data streams are potentially unbounded in size.
4. Once an element from a data stream has been processed it is either discarded or archived.
5. Individual data items have to be processed extremely fast to match the stream rate.
6. Only limited storage is available for processing or capturing and archiving data at stream rate.

As data stream is continuous flow of data, volume of data to be processed is enormous. With increasing volume of the data, it is no longer possible to process the data efficiently by using multiple passes. Rather, one can process a data item at most once. This leads to constraints on the implementation of the underlying algorithms. Therefore, stream mining algorithms typically need to be designed so that the algorithms work with one pass of the data.

In most cases, there is time component associated to the stream mining process. This is because the data may evolve over time. Therefore, a straightforward adaptation of
one-pass mining algorithms may not be an effective solution to the task. Stream mining algorithms need to be carefully designed with a clear focus on the evolution of the underlying data.

Data streams are simulated as sequences of simple tuples. There are two types of streams [2]. The basic model of data streams is an arrivals-only one. Here, the stream consists of a quantity of tuples, or items, which describe the input. Typically each tuple is a simple, small object, which might indicate, for example, the identity of a particular object of interest, and a weight or value associated with this arrival. In a network, the observation of a packet could be interpreted as a tuple indicating the intended destination of the packet, and the size of the packet payload in bytes. For another application, the same packet could be interpreted as a tuple whose identity is the concatenation of the source and destination of the packet, with a weight of 1, indicating that it is a single packet. A richer model allows departures: This captures more general situations in which earlier updates might be revoked, or observations for which negative values are feasible. In either case, the assumption is that each tuple in the input stream must be processed as it is seen, and cannot be revisited later unless it is stored explicitly by the stream algorithm within its limited internal memory.

1.2 Theoretical Foundations

Research problems and challenges that have been arisen in mining data streams can be solved to some extent using well established statistical and computational approaches [3]. Here main focus is data and the task to be performed on data. When data is the key focus (data based technique) then, idea is to examine only a subset of the whole dataset or to transform the data vertically or horizontally to an approximate smaller size data representation. Sampling, load shedding and sketching techniques etc are some of the data based techniques. Here is an outline of the basics of these techniques with pointers to its applications in the context of data stream analysis.

**Sampling:** Sampling refers to the process of probabilistic choice of a data item to be processed or not. Sampling is an old statistical technique that has been used for a long time. Sampling methods are widely used for traditional database applications, and are extremely popular because of their broad applicability across a wide array of tasks in data streams[1]. A further advantage of sampling methods is that unlike many other synopsis construction methods, they maintain their inter-attribute correlations across samples of the data. It is also often possible to use probabilistic inequalities in order to bind the effectiveness of a variety of applications with sampling methods. However, a key problem in extending sampling methods to the data stream scenario is that one does not know the total number of data points to be sampled in advance. Rather, one must maintain the sample in a dynamic way over the entire course of the computation. Reservoir sampling is a method, which maintains such a sample dynamically.

**Classic Reservoir Sampling:** First, make a reservoir (array) of 1,000 elements and fill it with the first 1,000 elements in the stream. That way if we have exactly 1,000 elements, the algorithm works. This is the base case.

Next, we want to process the i'th element (starting with i = 1,001) such that at the end of processing that step, the 1,000 elements in our reservoir are randomly sampled amongst the i elements we have seen so far. Start with i = 1,001. With what probability
after the 1001'th step should element 1,001 (or any element for that matter) be in the set of 1,000 elements? The answer is easy: 1,000/1,001. So, generate a random number between 0 and 1, and if it is less than 1,000/1,001 we should take element 1,001. In other words, choose to add element 1,001 to reservoir with probability 1,000/1,001. If we choose to add it, then replace any element in the reservoir chosen randomly. This produces a 1,000/1,001 chance of selecting the 1,001'th element, but what about the 2nd element in the list? The 2nd element is definitely in the reservoir at step 1,000 and the probability of it getting removed is the probability of element 1,001 getting selected multiplied by the probability of #2 getting randomly chosen as the replacement candidate. That probability is 1,000/1,001 * 1/1,000 = 1/1,001. So, the probability that #2 survives this round is 1 - that or 1,000/1,001.

This can be extended for the i'th round - keep the i'th element with probability 1,000/i and if we choose to keep it, replace a random element from the reservoir. It is pretty easy to prove that this works for all values of i using induction. It obviously works for the i'th element based on the way the algorithm selects the i'th element with the correct probability outright. The probability any element before this step being in the reservoir is 1,000/ (i-1). The probability that they are removed is 1,000/i * 1/1,000 = 1/i. The probability that each element sticks around given that they are already in the reservoir is (i-1)/i and thus the elements' overall probability of being in the reservoir after i rounds is 1,000/ (i-1) * (i-1)/i = 1,000/i.

Algorithm 1.1 : classic reservoir sampling [9]

Inputs: r {reservoir size} 1: k = 0
2: for each tuple arriving from the input stream do
3: k = k + 1
4: if k ≤ r then
5: add the tuple to the reservoir
6: else
7: sample the tuple with the probability r/k and replace a randomly selected tuple in the reservoir with the sampled tuple
8: end if
9: end for

Load Shedding: As we know data streams are processes which create large volumes of incoming data, they lead to several challenges in both processing the data as well as applying traditional database operations. For example, when the incoming rate of the data streams is higher than that can be processed by the system, techniques are required in order to selectively pick data points from the stream, without losing accuracy. This
technique is known as load shedding. Load shedding refers to the process of dropping a sequence of data streams[3]. It has been used successfully in querying data streams.

**Sketching:** Many data stream problems cannot be solved with just a sample. Instead, data structure can be used which, in effect, include a contribution from the entire input, rather than just the items picked in the sample. For example, consider trying to count the number of distinct objects in a stream. It is easy to see that unless almost all items are included in the sample, then we cannot tell whether they are the same or distinct. Sketch is referred as a compact data structure which summarizes the stream for certain types of query. For classes of data stream queries where no exact data structure with the desired properties exists, one can often design an approximate data structure that maintains a small *synopsis* or *sketch* of the data rather than an exact representation, and therefore is able to keep computation per data element to a minimum [1]. Performing data reduction through synopsis data structures as an alternative to batch processing or sampling is a fruitful research area with particular relevance to the data stream computation model. Sketches use some properties of random sampling in order to perform counting tasks in data streams. Sketches are most useful when the domain size of a data stream is very large. In such cases, the number of possible distinct elements becomes very large, and it is no longer possible to track them in space-constrained scenarios. There are two broad classes of sketches: projection based and hash based. Sketching is the process of randomly project a subset of the features. Sketching has been applied in comparing different data streams and in aggregate queries in the data stream.

**Synopsis Construction in Data Streams:** The large volume of data streams poses unique space and time constraints on the computation process. Many query processing, database operations, and mining algorithms require efficient execution which can be difficult to achieve with a fast data stream[1]. Furthermore, since it is impossible to fit the entire data stream within the available space, the space efficiency of the approach is a major concern. In many cases, it may be acceptable to generate approximate solutions for many problems by summarizing the data in a time and space-efficient way. In recent years a number of synopsis structures have been developed, which can be used in conjunction with a variety of mining and query processing techniques. Some key synopsis methods include those of sampling, wavelets, sketches and histograms.

**Mining compressed or approximate patterns:** To reduce the huge set of frequent patterns generated in data mining while maintain the high quality of patterns, recent studies have been focusing on mining a compressed or approximate set of frequent patterns. In general, pattern compression can be divided into two categories: lossless compression and lossy compression, in terms of the information that the result set contains, compared with the whole set of frequent patterns [10].

### 1.3 Stream Mining Algorithms

A number of algorithms have been proposed for extracting knowledge from streaming information. Some of the most popular techniques are clustering, frequent pattern mining, classification, and Outlier analysis techniques.

**Clustering:** The clustering problem is defined [11] as follows: for a given set of data points, one wishes to partition them into one or more groups of similar objects. The similarity of the objects with one another is typically defined with the use of some
distance measure or objective function. The clustering problem has been widely researched in the database, data mining and statistics communities because of its use in a wide range of applications. Recently, the clustering problem has also been studied in the context of the data stream environment. Some algorithms on clustering data streams assume that the clusters are to be computed over the entire data stream. Such methods simply view the data stream clustering problem as a variant of one-pass clustering algorithms. While such a task may be useful in many applications, a clustering problem needs to be defined carefully in the context of a data stream. This is because a data stream should be viewed as an infinite process consisting of data which continuously evolves with time. As a result, the underlying clusters may also change considerably with time. The nature of the clusters may vary with both the moment at which they are computed as well as the time horizon over which they are measured. For example, a user may wish to examine clusters occurring in the last month, last year, or last decade. Such clusters may be considerably different. Therefore, a data stream clustering algorithm must provide the flexibility to compute clusters over user-defined time periods in an interactive fashion. Since stream data naturally imposes a one-pass constraint on the design of the algorithms, it becomes more difficult to provide such flexibility in computing clusters over different kinds of time horizons using conventional algorithms.

**Frequent Itemset Mining:** Since the introduction of association rule mining, the frequent itemset mining (FIM) tasks have received a great deal of attention. Let’s assume we are given a set of items I. An itemset I belonging to S is some subset of items. A transaction is a couple T = (tid, I) where tid is the transaction identifier and I is an itemset. A transaction T = (tid, I) is said to support an itemset X, if $X \subseteq I$. A transaction database D is a set of transactions such that each transaction has a unique identifier. The cover of an itemset X in D consists of the set of transaction identifiers of transactions in D that support X: $\text{cover}(X,D) := \{ \text{tid} \mid (\text{tid},I) \in D, X \subseteq I \}$. The support of an itemset X in D is the number of transactions in the cover of X in D: $\text{support}(X;D) := |\text{cover}(X,D)|$. An itemset is called frequent in D if its support in D exceeds a given minimal support threshold $\sigma$ [4].

The goal is now to find all frequent itemsets, given a database and a minimal support threshold. If frequent itemsets are long, it simply becomes infeasible to mine the set of all frequent itemsets. In order to tackle this problem, several solutions have been proposed that only generate a representing subset of all frequent itemsets. Among these, the collections of all closed or maximal itemsets are the most popular. A frequent itemset I is called closed if it has no frequent superset with the same support, i.e., frequent itemset is called maximal if it has no superset that is frequent.

The collection of maximal frequent itemsets is a subset of the collection of closed frequent itemsets which is a subset of the collection of all frequent itemsets. Although all maximal itemsets characterize all frequent itemsets, the support of all their subsets is not available, while this might be necessary for some applications such as association rules. On the other hand, the closed frequent itemsets form a lossless representation of all frequent itemsets since the support of those itemsets that are not closed is uniquely determined by the closed frequent itemsets.

**Pattern Mining:** Patterns are mainly divided into three categories: frequent patterns, subfrequent patterns, and infrequent patterns [6]. The frequency of an itemset I over a time period $T$ is the number of transactions in $T$ in which $I$ occurs. The support of $I$ is the
frequency divided by the total number of transactions observed in $T$. Let the min-sup be $\rho = \varepsilon / \sigma$, where $\varepsilon$ is the maximum support error. $I$ is frequent if its support is no less than $\sigma$; it is sub-frequent if its support is less than $\sigma$ but no less than $\varepsilon$; otherwise, it is infrequent. Frequent and infrequent patterns have large number of commercial and scientific applications.

**Infrequent Pattern mining:** Patterns that are rarely found in database (or data stream) are often considered to be uninteresting and are eliminated using the support measure. Such patterns are known as infrequent patterns [5]. An infrequent pattern is an itemset or a rule whose support is less than the min-sup threshold. Although a vast majority of infrequent patterns are uninteresting, some of them might be useful to the analysis, particularly those that correspond to negative correlations in data. Searching for infrequent or outliers in data stream is an important area of research in the world of data mining with numerous applications, including credit card fraud detection, discovery of criminal activities in electronic commerce, weather prediction, marketing and customer segmentation [7].

**Frequent Pattern mining:** Frequent pattern mining has been the focus of great interest among data mining researchers and practitioners. It is today widely accepted to be one of the key problems in the data mining fields. Frequent pattern mining is to find the item sets that appear frequently from plenty of data, an item set is any subset of the set of all items. Frequent pattern mining is the discovery of relationships or correlations between items in a dataset.

In the case of data streams, one may wish to find the frequent item sets either over a sliding window or the entire data stream. In the case of data streams, the problem of frequent pattern mining can be studied under several Models [1].

**Entire Data Stream Model:** In this model, the frequent patterns need to be mined over the entire data stream. Thus, the main difference from a conventional pattern mining algorithm is that the frequent patterns need to be mined in one pass over the entire data stream. Most frequent pattern mining algorithms require multiple passes in order to estimate the frequency of patterns of different sizes in the data. A natural method for frequent pattern counting is to use sketch-based algorithms in order to determine frequent patterns. Sketches are often used in order to determine heavy-hitters in data streams, and therefore, an extension of the methodology to the problem of finding frequent patterns is natural. This model does not allow false negatives, but may miss some of the frequent patterns. The main advantage of such a technique is that it is possible to provide a more concise set of frequent patterns at the expense of losing some of the patterns with some probability which is quite low for practical purposes.

**Sliding Window Model:** In many cases, the data stream may evolve over time, as a result of which it is desirable to determine all the frequent patterns over a particular sliding window [1]. The main assumption of this approach is that the numbers of frequent patterns are not very large, and therefore, it is possible to hold the transactions in each sliding window in main memory. Here window is divided into small basic windows and only store a synopsis and timestamp for each portion. When the time stamp of oldest window expires, its synopsis is removed, a fresh window is added to the front and the aggregate is incrementally recomputed. The design of sliding window is based on the fact that people are often interested in recent changes at a fine granularity, but long term changes at a coarse granularity.
**Damped Window Model:** Pure sliding windows are not the only way by which the evolution of data streams can be taken into account during the mining process. A second way is to introduce a decay factor into the computation [1]. Specifically, the weight of each transaction is multiplied by a factor of $f < 1$, when a new transaction arrives. The overall effect of such an approach is to create an exponential decay function on the arrivals in the data stream. Such a model is quite effective for evolving data stream, since recent transactions are counted more significantly during the mining process. Specifically, the decay factor is applied only to those itemsets whose counts are affected by the current transaction. However, the decay factor will have to be applied in a modified way by taking into account the last time that the itemset was touched by an update. This approach works because the counts of each itemset reduce by the same decay factor in each iteration, as long as a transaction count is not added to it. Such approach is also applicable to other mining problem, where statistics are represented as the sum of decaying values.

**Compact Representation of Frequent Itemsets:** Number of frequent itemsets produced from a data stream can be very large. It is useful to identify a small representative set of itemsets from which all other frequent itemsets can be derived. Two such representations are maximal and closed frequent itemsets[5].

**Maximal Frequent Itemsets:** A maximal frequent itemset is defined as a frequent itemset for which none of its immediate supersets are frequent. Maximal frequent itemsets effectively provide a compact representation of frequent itemsets. They form the smallest set of itemsets from which all frequent itemsets can be derived. Despite providing a compact representation, maximal frequent itemsets do not contain the support information of their subsets.

**Closed Frequent Itemsets:** Closed itemsets provide a minimal representation of itemsets. Here itemsets are represented without losing their support information. An itemset $X$ is closed if none of its immediate supersets has exactly the same support count as $X$. An itemset is a closed frequent itemset if it closed and its support is greater than or equal to minsup. Closed frequent itemsets are useful for removing some of the redundant association rules.

Following fig 1.1 gives the relationship between closed, maximal and frequent sets.

![Fig. 1.1](image)

**Data Stream Classification:** The problem of classification is perhaps one of the most widely studied in the context of data stream mining. Classification problem aims to identify the characteristics that indicate the group to which each instance belongs. It can
be used both to understand the existing data and to predict how new instances will behave. The problem of classification is made more difficult by the evolution of the underlying data stream. Therefore, effective algorithms need to be designed in order to take temporal locality into account. The concept of stream evolution is sometimes referred to as concept drift in the stream classification literature. Some of these algorithms are designed to be purely one-pass adaptations of conventional classification algorithms, whereas others are more effective in accounting for the evolution of the underlying data stream. Most of the work on classification is based on Histograms, Wavelets, Decision trees etc. The broad methods which are studied for classifications in the data stream scenario are:

**VFDT Method:** The VFDT (Very Fast Decision Trees) method has been adapted to create decision trees which are similar to those constructed by a conventional learner with the use of sampling based approximation [1].

**On Demand Classification:** While most stream classification methods are focused on a training stream, the on demand method is focused on the case when both the training and the testing stream evolve over time [1].

**Outlier analysis:** One of the central tasks in managing, monitoring and mining data streams is that of outliers. Deviants (outliers) are based on one of the most fundamental statistical concept of standard deviation (or variance). Deviants are values whose removal from dataset leads to an improved compressed representation of the remaining items. Identifying deviants on a massive data stream using very small space at any instant is a challenge for researchers [7].

### 1.4 Data structure for analysis

Most popularly used data structures are trie tree, prefix trees and Frequency Pattern tree (FP Tree). Tree structure consists of one root labeled as “null”, a set of *item-prefix subtrees* as the children of the root, and a *frequent-item-header table*. Each node in the *item-prefix subtree* consists of three fields: *item-name*, *count*, and *node-link*, where *item-name* registers which item this node represents, *count* registers the number of transactions represented by the portion of the path reaching this node, and *node-link* links to the next node in the *FP-tree* carrying the same item-name, or null if there is none. Each entry in the *frequent-item-header table* consists of two fields, (1) *item-name* and (2) *head of node-link* (a pointer pointing to the first node in the tree carrying the *item-name*).

Following table 1.1 gives summary of stream data mining algorithms [8].

<table>
<thead>
<tr>
<th>Sr.No</th>
<th>Algorithm</th>
<th>Mining Task</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VFKM</td>
<td>K-Means</td>
<td>Sampling and reducing the number of passes at each step of the algorithm</td>
</tr>
<tr>
<td>2</td>
<td>VFDT</td>
<td>Decision Trees</td>
<td>Sampling and reducing the number of passes at each step of the algorithm</td>
</tr>
<tr>
<td>3</td>
<td>Approximate</td>
<td>Frequent</td>
<td>Incremental Pruning and update of itemsets with each block of</td>
</tr>
<tr>
<td>Frequency Counts</td>
<td>itemsets</td>
<td>transactions and time-sensitive patterns extension</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>4 FP-Stream</td>
<td>Frequent itemsets</td>
<td>Incremental Pruning and update of itemsets with each block of transactions and time-sensitive patterns extension</td>
<td></td>
</tr>
<tr>
<td>5 Concept-Drifting classification</td>
<td>Classification</td>
<td>Ensemble classifiers</td>
<td></td>
</tr>
<tr>
<td>6 AWSOM</td>
<td>Prediction</td>
<td>Incremental Wavelets</td>
<td></td>
</tr>
<tr>
<td>7 Approximate k-median</td>
<td>K-median</td>
<td>Sampling and reducing the number of passes at each step of the algorithm</td>
<td></td>
</tr>
<tr>
<td>8 GEMM</td>
<td>General Applied to decision tress and frequent itemsets</td>
<td>Sampling</td>
<td></td>
</tr>
<tr>
<td>9 CDM</td>
<td>Decision Trees, Bayesian Nets and clustering</td>
<td>Fourier spectrum representation of the results to save the limited bandwidth</td>
<td></td>
</tr>
<tr>
<td>10 ClusStream</td>
<td>Clustering</td>
<td>Online summarization and offline clustering</td>
<td></td>
</tr>
<tr>
<td>11 STREAM-LOCALSEARCH</td>
<td>Clustering</td>
<td>Sampling and incremental learning</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1 Mining Data Stream Algorithms

1.5 Research Problem

As frequent pattern mining is an essential data mining task, developing efficient frequent mining techniques has been an important research direction in data mining. The general goal of this research is to develop algorithm to:

- Construct Synopsis of data stream of transactions
- Mine frequent itemsets
- Mine frequent patterns
- Mine infrequent patterns

As an inherent problem, frequent pattern mining over data stream has to deal with enormous data. Practically it is not possible to store the whole data so synopsis of data stream is constructed using sampling as summarization technique. Once the synopsis is constructed it is analyzed for mining frequent itemset. Mining association between these
frequent itemsets gives frequent patterns. Not only frequent patterns but infrequent patterns also have important applications like intrusion detection or outlier detection.

Here datasets are simulated as data stream of transactions. Data stream is offline and arrival only model of data stream is considered for analysis.

**Motivation:** Most of the research work on frequent pattern mining, adopt tree approach, which requires:

a. Ordering of items frequency wise, prior to updation of tree.

b. Tree is a recursive data structure which requires lots of space as well as computation time.

c. If numbers of distinct items over a data stream are less in number then use of tree structure is again a costly affair.

d. As data stream is continuous flow of data, volume of data to be processed is enormous. With increasing volume of the data, it is no longer possible to process the data efficiently by using multiple passes

This research work tries to make progress in giving some approximate solution for the above issues.

**Contributions:** Frequent pattern mining over data stream has to deal with enormous data. Practically it is not possible to store the whole data to overcome this problem many data summarization techniques are used. Here reservoir sampling method is modified to construct synopsis and count frequent item sets in the data stream.

To mine frequent items (number of distinct items are less in number) a local static data structure i.e. a double dimension array is used, which can generate patterns of length one and two. Using these patterns of length two longer patterns can be generated.

A dynamic FP-tree data structure for sequential frequent pattern mining is proposed. It is non recursive in nature. It does not require the list of items to be ordered. Items are added as and when they arrive.

FP tree is also used to mine infrequent patterns. Here the constraint is put on the support factor (deciding a support factor itself is a research problem in data stream so for experiments support factors are accepted from the user) and only those items are retrieved from the tree which satisfies this constraint. An infrequent patterns mining algorithm using dynamic FP-Tree is also proposed.

**The salient features of the research work presented here are:**

- A through literature review has been carried out.

- Data stream is continuous flow of data. Since it is not possible to analyze the whole stream. Hence a new algorithm based on reservoir sampling has been proposed to construct synopsis of data stream.

- Frequent itemset mining has number of scientific and commercial applications. A new algorithm for mining frequent itemsets over data stream has been proposed. This algorithm is also based on reservoir sampling.

- An algorithm is proposed to mine frequent itemsets over data stream which is based on trie tree. Here Trie tree is implemented using array. This algorithm works well if the numbers of distinct items over the stream are less in number as compared to existing trie tree algorithms.

- Once the frequent itemssets has been mined from the stream it is important get some association between these itemsets which results in frequent patterns. A new
data structure called Dynamic FP-Tree which is based on FP-tree has been proposed. This dynamic FP-tree gives good results in sequential pattern mining.

- Some applications like intrusion detection require mining of infrequent patterns over the stream. The above proposed dynamic FP-Tree is used to mine infrequent patterns over data stream.
- All the four proposed algorithms and data structure have been tested for their performance and efficiency.
- Computer programs are developed to test the performance and efficiency of the algorithms and data structure.
- Datasets consisting of transaction are simulated as data stream. Data stream used for analysis is offline. Here datasets are downloaded from http://fimi.cs.helsinki.fi/data/

### 1.6 Thesis Outline

The remainder of the thesis is structured as follows:

- Chapter 2 is based on synopsis construction over data stream. It may be impossible to store an entire data stream due to its tremendous volume. To discover knowledge or patterns from data streams, it is necessary to develop data stream summarization techniques. Section 2.2 is devoted to sketches which summarizes the entire data stream. Histograms are discussed in section 2.3. Section 2.4 introduces the concept of wavelets for hierarchical data decomposition and summarization. Section 2.5 deals with sampling methods for synopsis construction. In section 2.6 a new algorithm 2.4 is proposed. The proposed algorithm deals with construction of synopsis using reservoir sampling. Section 2.7 is dedicated to frequent itemset mining over transaction stream. In this section a new algorithm 2.5 is implemented to mine frequent itemsets over data stream using reservoir sampling. Section 2.8 gives concluding remarks on proposed work. Chapter concludes with references in section 2.9.

- Chapter 3 is devoted to frequent itemset mining. The frequent itemset problem is to find all items occurring more than a given fraction of the time. Many applications rely directly or indirectly on finding the frequent items, and implementations are in use in large scale industrial systems. Section 3.2 divides the frequent itemset algorithms into four categories which are discussed in detail in following sections. Section 3.3 is devoted to sketch algorithms which compute a summary that is linear transformation of the frequency vector. Tree based algorithms are discussed in section 3.4. Section 3.5 elaborates in detail counter based algorithm. A new algorithm 3.10 to mine frequent itemsets is proposed in section 3.6. This new algorithm mines frequent itemset using static implementation of trie tree. Section 3.7 discusses the system and optimization issues related to algorithm 3.10. Section 3.8 gives the concluding remarks and chapter concludes with references in section 3.9.

- Chapter 4 introduces the concept of frequent pattern mining. Frequent pattern mining is the discovery of relationships or correlations between items in a data stream either over a sliding window or the entire data stream. Section 4.2 elaborates on frequent itemset generation. Section 4.3 describes in detail Apriori algorithm. Section 4.4 is dedicated to Prefix trees, giving different types of prefix trees in frequent pattern mining. Section 4.5 deals with time sensitive pattern mining. Section 4.6 introduces a new data
structure called Dynamic FP-tree and a new algorithm 4.2 is proposed to mine frequent itemsets over sliding windows. Section 4.7 gives the experimental details of the proposed algorithm 4.2. Section 4.8 comments on the conclusion and chapter concludes with references in section 4.9.

Chapter 5 is based on Infrequent Patterns. Infrequent pattern mining is concerned with extracting “rare” or “unusual” patterns from streams of data. In the past, frequent pattern mining has been investigated in detail with little research being done in infrequent pattern mining. However, infrequent patterns are often more useful than frequent patterns as they provide information about events of interest (such as in network intrusion detection). Section 5.2 presents different techniques for Mining Interesting Infrequent Patterns followed by section 5.3 which is based on outlier detection. In section 5.4 a new Algorithm 5.3 is proposed to mine infrequent patterns using Dynamic FP-tree. Section 5.5 gives the performance details of the algorithm 5.3 and section 5.6 gives concluding remarks. References related to the chapter are given section 5.7.

In chapter 6 interesting applications of mining frequent patterns are discussed and some future directions for the research are presented.
1.7 References

[1] Data Streams: Models and algorithms edited by Charu C. Aggarwal
Springer- 2007 (Book)

Workshop on data stream analysis – San Leucio, Italy - March, 15-16, 2007

Mining Data Streams: A Review, SIGMOD Record, Vol. 34, No. 2, June 2005


[5] Pang-Ning Tan, Michael Steinbach, Vipinn Kumar: Introduction to data mining,
Pearson Education, Book

multiple time granularities”, In : H. Kargupta, A. Joshi, K. Sivakumar, eds. Next

Streams”, Proceedings of the 16th International Conference on Scientific and

[8] Mohamed Medhat Gaber, Shonali Krishnaswamy, Arkady Zaslavsky, Ubiquitous
Data Stream Mining, Current Research and future direction workshop proceedings
held in conjunction with PAKDD 2004, Sydney, Australia, May-26 -2004

Sampling over Data Streams, 19th International conference on scientific and
Statistical Data base Management (SSDBM- 2007), IEEE

[10] Dong Xin, Jiawei Han, Xifeng Yan, Hong Cheng: Mining Compressed Frequent-
Pattern Sets, Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

[11] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, Philip S. Yu: A Framework for
Clustering Evolving Data Streams, Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003

[12] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Jennifer Widom:
Models and Issues in Data Stream Systems, Proceedings of the twenty first ACM
SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems,
June- 3-5, 2002, Madison, Wisconsin.