Chapter 6

MINING SENSOR DATA STREAMS

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY
charu@us.ibm.com

Abstract

In recent years, advances in hardware technology have facilitated new ways of collecting data continuously. One such application is that of sensor data, which may continuously monitor large amounts of data for storage and processing. In this paper, we will discuss the general issues which arise in mining large amounts of sensor data. In many cases, the data patterns may evolve continuously, as a result of which it is necessary to design the mining algorithms effectively in order to account for changes in underlying structure of the data stream. This makes the solutions of the underlying problems even more difficult from an algorithmic and computational point of view. In this chapter we will provide an overview of the problem of data stream mining and the unique challenges that data stream mining poses to different kinds of sensor applications.

Keywords: Data Streams, Sensor Data, Sensor Stream Mining

1. Introduction

In recent years, advances in sensor technology have lead to the ability to collect large amounts of data from sensors in an automated way. When the volume of the underlying data is very large, it leads to a number of computational and mining challenges:

- With increasing volume of the data, it is no longer possible to process the data efficiently by using multiple passes. Rather, one

C.C. Aggarwal (ed.), *Managing and Mining Sensor Data*, DOI 10.1007/978-1-4614-6309-2_6,

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 an inherent temporal component to the stream mining process. This is because the data may evolve over time. This behavior of data streams is referred to as *temporal locality*. 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 which is collected from sensors is often uncertain and error prone, as a result of which it is critical to be able to reduce the effects of the uncertainty in the mining process.

Another important characteristic of sensor data streams is that they are often mined in a distributed fashion. In some cases, intermediate sensor nodes may have limited processing power, and it may be desirable to perform in-network sensor processing for a variety of mining applications. In such cases, the application algorithms need to be designed with such criteria in mind [30, 60]. This chapter will provide an overview of the key challenges in sensor stream mining algorithms which arise from the unique setup in which these problems are encountered.

This chapter is organized as follows. In the next section, we will discuss the generic challenges which arise in the context of storage and processing of sensor data. The next section deals with several issues which arise in the context of data stream management. In section 3, we discuss several mining algorithms on the data stream model. Section 4 discusses various scientific applications of data streams. Section 5 discusses the research directions and conclusions.

2. Sensor Stream Mining Issues

Since 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. Therefore, new designs of data streaming systems are required for handling sensor data [23]. The challenging issues in sensor stream mining may arise during different phases including data collection, transmission, storage and processing. Some of the these key issues are as follows:

2.1 Data Uncertainty and Volume

Typical sensor mining applications collect large amounts of data, which is often subject to uncertainty and errors. This is because sensors often have errors in data collection and transmission. In many cases, where the battery runs out, the data may also be incomplete. Therefore, methods are required to store and process the uncertainty in the underlying data. A common technique is to perform *model driven data acquisition* [36], which explicitly models the uncertainty during the acquisition process. Furthermore, new methods must be constructed in order to explicitly use these uncertainty models during data processing. A detailed discussion of a variety of methods for modeling and mining uncertain data are proposed in [14].

A variety of techniques may be used in order to handle the large volume of sensor data. One method may be to simply lower the sampling rate for capturing or transmitting the data. This reduces the granularity of the data collection process. Many techniques have also been proposed [34, 35] in order to reduce or compress the volume of the data in the sensor network. Another approach is to discard parts of the data even after collection and transmission. 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 *loadshedding*. Since sensor generated data streams are generated by processes which are extraneous to the stream processing application, it is not possible to control the incoming stream rate. As a result, it is necessary for the system to have the ability to quickly adjust to varying incoming stream processing rates. One particular type of adaptivity is the ability to gracefully degrade performance via "load shedding" (dropping unprocessed tuples to reduce system load) when the demands placed on the system cannot be met in full given available resources. The loadshedding can be tailored to specific kinds of applications such as query processing or data mining. A discussion of several loadshedding techniques are provided in [4]. Finally, a method for reducing the data volume is that of sensor selection in which data from only a particular set of sensors is transmitted at a particular time, so that most of the essential information is retained. This is also useful for reducing the power requirements of transmission. We will discuss this method in the next subsection.

2.2 Power Issues in Sensor Collection and Transmission

Since sensor nodes have limited battery power, the issue of data collection and transmission is a challenging one for sensor networks. Sensor nodes have limited battery power, as a result of which it is necessary to collect and transmit the data on a limited basis. This means that it is often necessary to collect only a subset of the data for mining purposes. A classic technique which is often used in order to reduce the amount of collected data is *sensor selection*. In data driven sensor selection, goal-oriented techniques are used in order to reduce the amount of data collected for mining purposes [19, 2–1, 45, 61]. In most of these methods, the idea is to use the massive redundancy across the different sensors in order to reduce the total data which is collected. For example, in an environmental monitoring applications, two sensors which are located physically close together may often have very similar readings. In such cases, the correlations across the different sensors are leveraged in order to select a small number of sensors from which the data is collected. The values on the other sensors can be predicted from this small set. We note that this small set may vary over time, as different sensors may be inoperative at different times, or the correlations among the data may change. Methods for power-efficient and dynamic sensor selection are discussed in [1]. Another technique which is often used in order to reduce the power transmission costs is a method referred to as *in network* processing. We will discuss this technique in the next subsection.

2.3 In-Network Processing

Since sensor networks may use hundreds and thousands of nodes over a large area, and all the data from the different nodes may be need to be fused, this can incur significant communication costs, of all the raw data is directly transmitted to a central server for processing. While such a naive solution is easy to implement, its energy costs may be too large to make it practical for very large scale sensor networks which are distributed over wide regions. Since the cost of transmission is higher than computation, it is usually advantageous to organize the sensors into clusters. In such an environment, the data gathered by the sensors is *processed within the network* and only aggregated information is returned to the central location. This responsibility is typically provided to certain nodes in the network, which are referred to as *aggregators*. Numerous functions can be designed for such nodes, and the corresponding data sent may depend upon the relevant aggregate queries which are posed at the central server. Thus, the underlying assumption is that such em-

bedded devices in the network are smart enough to be able to compute functions of modest complexity. Such an approach results in a reduction of the transmission costs, both because of the smaller distances of transmission in a clustered environment, and also because only the aggregated data is transmitted (which has much lower volume than the raw data). It is possible to design different kinds of cluster hierarchies in order to optimize the transmission costs in the underlying network. A detailed discussion of the different aspects of in-network query processing may be found in [63, 78].

3. Stream Mining Algorithms

In this section, we will discuss the key stream mining problems and will discuss the challenges associated with each problem. We will also provide a broad overview of the different directions of research for these problems.

3.1 Data Stream Clustering

Clustering is a widely studied problem in the data mining literature. However, it is more difficult to adapt arbitrary clustering algorithms to data streams because of one-pass constraints on the data set. An interesting adaptation of the k-means algorithm has been discussed in [43] which uses a partitioning based approach on the entire data set. This approach uses an adaptation of a k-means technique in order to create clusters over the entire data stream. However, in practical applications, it is often desirable to be able to examine clusters over user-specified time-horizons. For example, an analyst may desire to examine the behavior of the clusters in the data stream over the past one week, the past one month, or the past year. In such cases, it is desirable to store *intermediate cluster statistics*, so that it is possible to leverage these in order to examine the behavior of the underlying data.

One such technique is micro-clustering [10], in which we use cluster feature vectors [81] in order to perform stream clustering. The cluster feature vectors keep track of the first-order and second-order moments of the underlying data in order to perform the clustering. These features satisfy the following critical properties which are relevant to the stream clustering process:

- **Additivity Property:** The statistics such as the first- or second-order moments can be maintained as a simple addition of statistics over data points. This is critical in being able to maintain the statistics efficiently over a fast data stream. Furthermore, additivity also implies subtractivity; thus, it is possible to obtain the

statistics over a particular time horizon, by subtracting out the statistics at the beginning of the horizon from the statistics at the end of the horizon.

- **Computational Convenience:** The first and second order statistics can be used to compute a vast array of cluster parameters such as the cluster centroid and radius. This is useful in order to be able to compute important cluster characteristics in real time.

It has been shown in [10], that the micro-cluster technique is much more effective and versatile than the k-means based stream technique discussed in [43]. This broad technique has also been extended to a variety of other kinds of data. Some examples of such data are as follows:

- **High Dimensional Data:** The stream clustering method can also be extended to the concept of projected clustering [5]. A technique for high dimensional projected clustering of data streams is discussed in [11]. In this case, the same micro-cluster statistics are used for maintaining the characteristics of the clusters, except that we also maintain additional information which keeps track of the projected dimensions in each cluster. The projected dimensions can be used in conjunction with the cluster statistics to compute the projected distances which are required for intermediate computations. Another innovation proposed in [11] is the use of decay-based approach for clustering. The idea in the decay-based approach is relevant for the case of evolving data stream model, and is applicable not just to the high dimensional case, but any of the above variants of the micro-cluster model. In this approach, the weight of a data point is defined as $2^{-\lambda \cdot t}$, where t is the current time-instant. Thus, each data point has a half-life of $1/\lambda$, which is the time in which the weight of the data point reduces by a factor of 2. We note that the decay-based approach poses a challenge because the micro-cluster statistics are affected at each clock tick, even if no points arrive from the data stream. In order to deal with this problem, a *lazy approach* is applied to decay-based updates, in which we update the decay-behavior for a micro-cluster only if a data point is added to it. The idea is that as long as we keep track of the last time t_s at which the micro-cluster was updated, we only need to multiply the micro-cluster statistics by $2^{-\lambda(t_c - t_s)}$, where t_c is the current time instant. After multiply the decay statistics by this factor, it is possible to add the micro-cluster statistics of the current data point. This approach can be used since the statistics of each micro-cluster decay by the same factor in each track, and it is therefore possible to implicitly keep track of the decayed values,

as long as a data point is not added to the micro-cluster. In the latter case, the statistics need to be updated explicitly, while other counts can still be maintained implicitly.

- **Uncertain Data:** In many cases, such as in sensor networks, the underlying data may be noisy and uncertain. In such cases, it may be desirable to incorporate the uncertainty into the clustering process. In order to do so, the micro-cluster statistics are appended with the information about the underlying uncertainty in the data. This information can be used in order to make more robust clustering computations. The advantages of using the uncertainty into the clustering process are illustrated in [7].
- **Text and Categorical Data:** A closely related problem is that of text and categorical data. The main difference with the quantitative domain is the nature of the statistics which are stored for clustering purposes. In this case, we maintain the counts of the frequencies of the discrete attributes in each cluster. Furthermore, we also maintain the inter-attribute correlation counts which may be required in a variety of applications. In [12], an efficient algorithm has been proposed for clustering text and categorical data streams. This algorithm also allows for a decay-based approach as in [11]. Text and categorical streams often arise in the context of social sensors such as micro-blogging sites, or other tagging or event detection scenarios.

In addition, a number of density-based methods [25, 28] have also been proposed for the problem of stream clustering.

In the context of sensor networks, the stream data is often available only in a *distributed setting*, in which large volumes of data are collected separately at the different sensors. A natural approach for clustering such data is to transmit all of the data to a centralized server. The clustering can then be performed at the centralized server in order to determine the final results. Unfortunately, such an approach is extremely expensive in terms of its communication costs. Therefore, it is important to design a method which can reduce the communication costs among the different processors. A method proposed in [32] performs local clustering at each node, and merges these different clusters into a single global clustering at low communication cost. Two different methods are proposed in this work. The first method determines the cluster centers by using a *furthest point algorithm*, on the current set of data points at the local site. In the furthest point algorithm, the center of a cluster is picked as a furthest point to the current set of centers. For any incoming data point, it is assigned to its closest center, as long

the distance is within a certain factor of an optimally computed radius. Otherwise, a re-clustering is forced by applying the furthest point algorithm on current set of points. After the application of the furthest point algorithm, the centers are transmitted to the central server, which then computes a global clustering from these local centers over the different nodes. These global centers can ten be transmitted to the local nodes if desired. One attractive feature of the method is that an approximation bound is proposed on the quality of the clustering. A second method for distributed clustering proposed in [32] is the *parallel guessing algorithm*. Another method for distributed sensor stream clustering which reduces the dimensionality and communication cost by maintaining an online discretization may be found in [68].

3.2 Data Stream Classification

The problem of classification is perhaps one of the most widely studied in the context of data stream mining. 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 [39], whereas others (such as the methods in [13, 48]) are more effective in accounting for the evolution of the underlying data stream. The broad methods which are studied for classification in the data stream scenario are as follows:

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 approximations. The VFDT method splits a tree using the current best attribute, taking into consideration the fact that the number of examples used are sufficient to preserve the Hoeffding bound in a way that the output is similar to that of a conventional learner. The key question during the construction of the decision tree is the choice of attributes to be used for splits. Approximate ties are broken using a user-specified threshold of acceptable error-measure for the output. It can be shown that for any small value of δ, a particular choice of the split variable is the correct choice with probability at least $1 - \delta$, if a sufficient number of stream records have been processed. This number has been shown in [39] to increase at a relatively modest rate of $\ln(1/\delta)$. This bound can then be extended to the entire decision tree, so as to quantify the probability that the same decision tree as a conventional learner is cre-

ated. The VFDT method has also been extended to the case of evolving data streams. This framework is referred to as CVFDT [48], and it runs VFDT over fixed sliding windows in order to always have the most updated classifier. Jin and Agrawal [50] have extended the VFDT algorithm in order to process numerical attributes and reduce the sample size which is calculated using the Hoeffding bound. Since this approach reduces the sample size, it improves efficiency and space requirements for a given level of accuracy.

On Demand Classification: While most stream classification methods are focussed on a training stream, the on demand method is focussed on the case when both the training and the testing stream evolves over time. In the on demand classification method [13], we create class-specific micro-clusters from the underlying data. For an incoming record in the test stream, the class label of the closest micro-cluster is used in order to determine the class label of the test instance. In order to handle the problem of stream evolution, the micro-clusters from the specific time-horizon are used for the classification process. A key issue in this method is the choice of horizon which should be used in order to obtain the best classification accuracy. In order to determine the best horizon, a portion of the training stream is separated out and the accuracy is tested over this portion with different horizons. The optimal horizon is then used in order to classify the test instance.

Ensemble-based Classification: This technique [74] uses an ensemble of classification methods such as C4.5, RIPPER and naive Bayes in order to increase the accuracy of the predicted output. The broad idea is that a data stream may evolve over time, and a different classifier may work best for a given time period. Therefore, the use of an ensemble method provides robustness in the concept-drifting case.

Compression-based Methods: An interesting method for real-time classification of streaming sensor data with the use of compression techniques has been proposed in [57]. In this approach, time-series bitmaps, which can be updated in constant time are used as efficient classifiers. Because of the ability of be updated in constant time, these classifiers are very efficient in practice. The effectiveness of this approach has been illustrated on a number of insect-tracking data sets.

In the context of sensor networks, data streams may often have a significant level of errors and uncertainty. Data uncertainty brings a number of unique challenges with it in terms of the determination of the important features to be used for the classification process. In this context, a number of algorithms have been proposed for classification of uncertain data streams [14, 15]. In particular, the method discussed in

[15] constructs a density-based framework to summarize the uncertain data stream effectively and use it for classification purposes.

3.3 Frequent Pattern Mining

The problem of frequent pattern mining was first introduced in [16], and was extensively analyzed for the conventional case of disk resident data sets. In the case of data streams, one may wish to find the frequent itemsets either over a sliding window or the entire data stream [44, 53]. In the case of data streams, the problem of frequent pattern mining can be studied under several models:

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. Along this line, Manku and Motwani [64] proposed the first one lass algorithm called *Lossy Counting*, in order to find all frequent itemsets over a data stream. The algorithm allows false positives, but not false negatives. Thus, for a given support level s, the algorithm is guaranteed not to contain all frequent itemsets whose support is greater than $s - \epsilon$. Another interesting approach in [80] determines all the frequent patterns whose support is greater than s with probability at least $1 - \delta$, which the value of δ is as small as desired, as long as one is willing to add space and time complexity proportional to $\ln(1/\delta)$.Thus, 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. A method for determining the frequent patterns over a sliding window is discussed in [29]. The main assumption of this approach is that the number of frequent patterns are not very large, and therefore, it is possible to hold the transactions in each sliding window in main memory. The main focus of this approach is to determine closed frequent itemsets over the data stream. A new

mining algorithm called MOMENT is proposed, and the main idea is based on the fact that the boundary between closed frequent itemsets and frequent itemsets moves very slowly. A closed enumeration tree is developed in order to keep track of the boundary between closed frequent itemsets and the rest of the itemsets. Another method which is able to mine frequent itemsets over arbitrary time granularities is referred to as FPSTREAM [42]. This method is essentially an adaptation of the FP-Tree method to data streams.

Damped Window Model: We note that 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. 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. An algorithm proposed in [27] maintains a lattice for recording the potentially frequent itemsets and their counts. While the counts of each lattice may change upon the arrival of each transaction, a key observation is that it is sufficient to update the counts in a lazy way. 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. In other words, if t_c be the current transaction index, and the last time the count for the itemset was updated was at transaction index $t_s < t_c$, then we need to multiply the current counts of that itemset by $f^{t_s - t_c}$ before incrementing the count of this modified value. 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. We note that such a lazy approach is also applicable to other mining problems, where statistics are represented as the sum of decaying values. For example, in [11], a similar lazy approach is used in order to maintain decay-based micro-cluster statistics for a high dimensional projected stream clustering algorithm.

3.4 Change Detection in Data Streams

As discussed earlier, the patterns in a data stream may evolve over time. In many cases, it is desirable to track and analyze the nature of these changes over time. In [8, 37, 59], a number of methods have been discussed for change detection of data streams. In addition, data

stream evolution can also affect the behavior of the underlying data mining algorithms since the results can become stale over time. The broad algorithms for change diagnosis in data streams are as follows:

Velocity Density Estimation: In velocity density estimation [8], we compute the rate of change of data density of different points in the data stream over time. Depending upon the direction of density rate of change, one may identify regions of *dissolution*, *coagulation* and *shift*. Spatial profiles can also be constructed in order to determine the directions of shift in the underlying data. In addition, it is possible to use the velocity density concept in order to identify those combinations of dimensions which have a high level of evolution. Another technique for change quantification is discussed in [37], which uses methods for probability difference quantification in order to identify the changes in the underlying data. In [59], a method is discussed in order to determine statistical changes in the underlying data. Clustering [10] can be used in order to determine significant evolution in the underlying data. In [10], micro-clustering is used in order to determine significant clusters which have evolved in the underlying data.

A separate line of work is the determination of significant changes in the results of data mining algorithms because of evolution. For example in [10], it has been shown how to determine significant evolving clusters in the underlying data. In [13], a similar technique has been used to keep a refreshed classification model in the presence of evolving data. In this respect, micro-clustering provides an effective technique, since it provides a way to store intermediate statistics of the underlying data in the form of clusters. In [13], a micro-cluster based nearest neighbor classifier is used in order to classify evolving data streams. The key idea is to construct class-specific micro-clusters over a variety of time horizons, and then utilize the time horizon with the greatest accuracy in order to perform the classification process. The issue of stream evolution has been extended to many other problems such as synopsis construction and reservoir sampling [6]. We will discuss some of the synopsis construction methods later.

3.5 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. 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 [41]. Some key synopsis methods include those of sampling, wavelets, sketches and histograms. The key challenges which arise in the context of synopsis construction of data streams are as follows:

Broad Applicability: The synopsis structure is typically used as an intermediate representation, which is then leveraged for a variety of data mining and data management problems. Therefore, the synopsis structure should be c0onstructed in such a way that it has applicability across a wide range of problems.

One-pass constraint: As in all data stream algorithms, the one-pass constraint is critical to synopsis construction algorithms. We would like to design all synopsis construction algorithms in one pass, and this is not the case for most traditional methods. In fact, even simply methods such as sampling need to be re-designed in order to handle the one-pass constraint.

Time and Space Efficiency: Since data streams have a very large volume, it is essential to create the synopsis in a time- and space-efficient way. In this sense, some of the probabilistic techniques such as sketches are extremely effective for counting-based applications, since they require constant-space for provable probabilistic accuracy. In other words, the time- and space-efficiency depends only upon the accuracy of the approach rather than the length of the data stream.

Data Stream Evolution: Since the stream evolves over time, a synopsis structure which is constructed from the overall behavior of the data stream is not quite as effective as one which uses recent history. Consequently, it is often more effective to create synopsis structures which either work with sliding windows, or use some decay-based approach in order to weight the data stream points.

One key characteristic of many of the above methods is that while they work effectively in the 1-dimensional case, they often lose their effectiveness in the multi-dimensional case either because of data sparsity or because of inter-attribute correlations. Next, we will discuss the broad classes of techniques which are used for synopsis construction in data streams. Each of these techniques have their own advantages in different scenarios, and we will take care to provide an overview of the different array of methods which are used for synopsis construction in data streams. The broad techniques which are used for synopsis construction in data streams are as follows:

Reservoir Sampling: Sampling methods are widely used for tradi-

tional database applications, and are extremely popular because of their broad applicability across a wide array of tasks in data streams. 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 bound 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. A method called reservoir sampling was first proposed in [72], which maintains such a sample dynamically. This technique was originally proposed in the context of one-pass access of data from magnetic-storage devices. However, the techniques also naturally extend to the data stream scenario.

Let us consider the case, where we wish to obtain an unbiased sample of size n from the data stream. In order to initialize the approach, we simply add the first n points from the stream to the reservoir. Subsequently, when the $(t+1)$th point is received, it is added to the reservoir with probability $n/(t+1)$. When the data point is added to the reservoir, it replaces a random point from the reservoir. It can be shown that this simple approach maintains the uniform sampling distribution from the data stream. We note that the uniform sampling approach may not be very effective in cases where the data stream evolves significantly. In such cases, one may either choose to generate the stream sample over a sliding window, or use a decay-based approach in order to bias the sample. An approach for sliding window computation over data streams is discussed in [20].

A second approach [6] uses biased decay functions in order to construct synopsis from data streams. It has been shown in [6] that the problem is extremely difficult for arbitrary decay functions. In such cases, there is no known solution to the problem. However, it is possible to design very simple algorithms for some important classes of decay functions. One of these classes of decay functions is the *exponential decay function*. The exponential decay function is extremely important because of its *memory less property*, which guarantees that the future treatment of a data point is independent of the past data points which have arrived. An interesting result is that by making simple implementation modifications to the algorithm of [72] in terms of modifying the probabilities of insertion and deletion, it is possible to construct a robust algorithm for this problem. It has been shown in [6] that the approach is quite

effective in practice, especially when there is significant evolution of the underlying data stream.

While sampling has several advantages in terms of simplicity and preservation of multi-dimensional correlations, it loses its effectiveness in the presence of data sparsity. For example, a query which contains very few data points is unlikely to be accurate with the use of a sampling approach. However, this is a general problem with most techniques which are effective at counting frequent elements, but are not quite as effective at counting rare or distinct elements in the data stream.

Sketches: 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 become 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*. We will discuss each of them in turn.

Projection based sketches are constructed on the broad idea of random projection [54]. The most well known projection-based sketch is the AMS sketch [17, 18], which we will discuss below. It has been shown in [54], that by by randomly sampling subspaces from multi-dimensional data, it is possible to compute ϵ-accurate projections of the data with high probability. This broad idea can easily be extended to the massive domain case, by viewing each distinct item as a dimension, and the counts on these items as the corresponding values. The main problem is that the vector for performing the projection cannot be maintained explicitly since the length of such a vector would be of the same size as the number of distinct elements. In fact, since the sketch-based method is most relevant in the distinct element scenario, such an approach defeats the purpose of keeping a synopsis structure in the first place.

Let us assume that the random projection is performed using k sketch vectors, and r_i^j represents the jth vector for the ith item in the domain being tracked. In order to achieve the goal of efficient synopsis construction, we store the random vectors implicitly in the form of a seed, and this can be used to dynamically generate the vector. The main idea discussed in [49] is that it is possible to generate random vectors with a seed of size $O(\log(N))$, provided that one is willing to work with the restriction that $r_i^j \in \{-1, +1\}$ should be 4-wise independent. The sketch is computed by adding r_i^j to the jth component of the sketch for the ith item. In the event that the incoming item has frequency f, we add the value $f \cdot r_i^j$. Let us assume that there are a total of k sketch components which are denoted by $(s_1 \ldots s_k)$. Some key properties of the pseudo-

random number generator approach and the sketch representation are as follows:

- A given component r_i^j can be generated in poly-logarithmic time from the seed. The time for generating the seed is poly-logarithmic in the domain size of the underlying data.
- A variety of approximate aggregate functions on the original data can be computed using the sketches.

Some example of functions which can be computed from the sketch components are as follows:

- **Dot Product of two streams:** If $(s_1 \ldots s_k)$ be the sketches from one stream, and $(t_1 \ldots t_k)$ be the sketches from the other stream, then $s_j cdott_j$ is a random variable whose expected value of the dot product.
- **Second Moment:** If $(s_1 \ldots s_k)$ be the sketch components for a data stream, it can be shown that the expected value of s_j^2 is the second moment. Furthermore, by using Chernoff bounds, it can be shown that by selecting the median of $O(\log(1/\delta)$ averages of $O(1/\epsilon^2)$ copies of $s_j]cdott_j$, it is possible to guarantee the accuracy of the approximation to within $1 \pm \epsilon$ with probability at least $1 - \delta$.
- **Frequent Items:** The frequency of the ith item in the data stream is computed by by multiplying the sketch component s_j by r_i^j. However, this estimation is accurate only for the case of frequent items, since the error is estimation is proportional to the overall frequency of the items in the data stream.

More details of computations which one can perform with the AMS sketch are discussed in [17, 18].

The second kind of sketch which is used for counting is the *count-min* sketch [31]. The count-min sketch is based upon the concept of hashing, and uses $k = \ln(1/\delta)$ pairwise-independent hash functions, which hash onto integers in the range $(0 \ldots e/\epsilon)$. For each incoming item, the k hash functions are applied and the frequency count is incremented by 1. In the event that the incoming item has frequency f, then the corresponding frequency count is incremented by f. Note that by hashing an item into the k cells, we are ensuring that we maintain an overestimate on the corresponding frequency. It can be shown that the minimum of these cells provides the ϵ-accurate estimate to the frequency with probability at least $1 - \delta$. It has been shown in [31] that the method can also be naturally extended to other problems such as finding the dot product

Table 6.1. An Example of Wavelet Coefficient Computation

Granularity (Order k)	**Averages** Φ values	**DWT Coefficients** ψ values
$k = 4$	(8, 6, 2, 3, 4, 6, 6, 5)	-
$k = 3$	(7, 2.5, 5, 5.5)	(1, -0.5,-1, 0.5)
$k = 2$	(4.75, 5.25)	(2.25, -0.25)
$k = 1$	(5)	(-0.25)

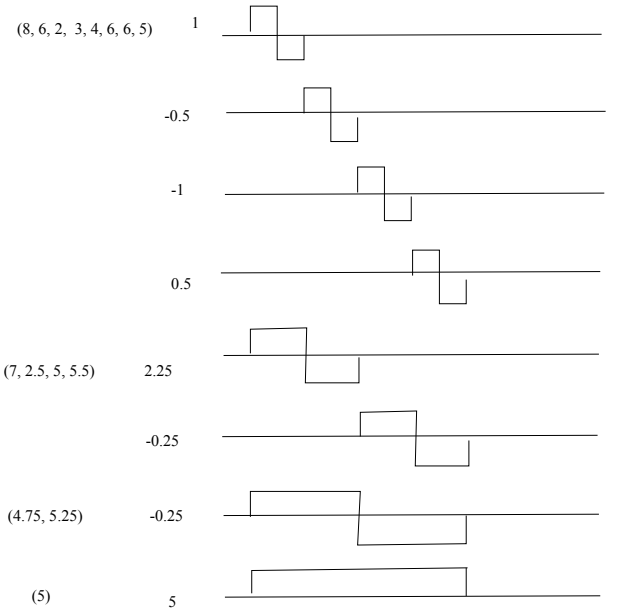

Figure 6.1. Illustration of the Wavelet Decomposition

or the second-order moments. The count-min sketch is typically more effective for problems such as frequency-estimation of individual items than the projection-based AMS sketch. However, the AMS sketch is more effective for problems such as second-moment estimation.

Wavelet Decomposition: Another widely known synopsis representation in data stream computation is that of the wavelet representation. One of the most widely used representations is the *Haar Wavelet*. We will discuss this technique in detail in this section.

This technique is particularly simple to implement, and is widely used in the literature for hierarchical decomposition and summarization. The basic idea in the wavelet technique is to create a decomposition of the data characteristics into a set of wavelet functions and basis functions. The property of the wavelet method is that the higher order coefficients

of the decomposition illustrate the broad trends in the data, whereas the more localized trends are captured by the lower order coefficients.

We assume for ease in description that the length q of the series is a power of 2. This is without loss of generality, because it is always possible to decompose a series into segments, each of which has a length that is a power of two. The Haar Wavelet decomposition defines 2^{k-1} coefficients of order k. Each of these 2^{k-1} coefficients corresponds to a contiguous portion of the time series of length $q/2^{k-1}$. The ith of these 2^{k-1} coefficients corresponds to the segment in the series starting from position $(i-1) \cdot q/2^{k-1} + 1$ to position $i * q/2^{k-1}$. Let us denote this coefficient by ψ_k^i and the corresponding time series segment by S_k^i. At the same time, let us define the average value of the first half of the S_k^i by a_k^i and the second half by b_k^i. Then, the value of ψ_k^i is given by $(a_k^i - b_k^i)/2$. More formally, if Φ_k^i denote the average value of the S_k^i, then the value of ψ_k^i can be defined recursively as follows:

$$\psi_k^i = (\Phi_{k+1}^{2 \cdot i-1} - \Phi_{k+1}^{2 \cdot i})/2 \tag{6.1}$$

The set of Haar coefficients is defined by the Ψ_k^i coefficients of order 1 to $\log_2(q)$. In addition, the global average Φ_1^1 is required for the purpose of perfect reconstruction. We note that the coefficients of different order provide an understanding of the major trends in the data at a particular level of granularity. For example, the coefficient ψ_k^i is half the quantity by which the first half of the segment S_k^i is larger than the second half of the same segment. Since larger values of k correspond to geometrically reducing segment sizes, one can obtain an understanding of the basic trends at different levels of granularity. We note that this definition of the Haar wavelet makes it very easy to compute by a sequence of averaging and differencing operations. In Table 6.1, we have illustrated how the wavelet coefficients are computed for the case of the sequence $(8, 6, 2, 3, 4, 6, 6, 5)$. This decomposition is illustrated in graphical form in Figure 6.1. We also note that each value can be represented as a sum of $\log_2(8) = 3$ linear decomposition components. In general, the entire decomposition may be represented as a tree of depth 3, which represents the hierarchical decomposition of the entire series. This is also referred to as the *error tree*. In Figure 6.2, we have illustrated the error tree for the wavelet decomposition illustrated in Table 6.1. The nodes in the tree contain the values of the wavelet coefficients, except for a special *super-root* node which contains the series average. This super-root node is not necessary if we are only considering the relative values in the series, or the series values have been normalized so that the average is already zero. We further note that the number of wavelet coefficients in this series is 8, which is also the length of the original series. The original

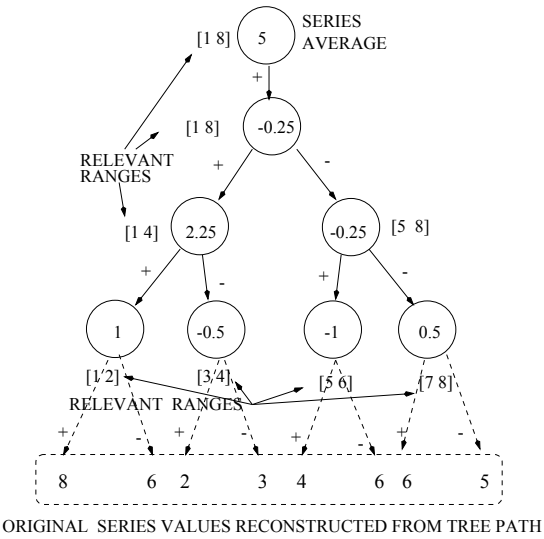

Figure 6.2. The Error Tree from the Wavelet Decomposition

series has been replicated just below the error-tree in Figure 6.2, and it can be reconstructed by adding or subtracting the values in the nodes along the path leading to that value. We note that each coefficient in a node should be added, if we use the left branch below it to reach to the series values. Otherwise, it should be subtracted. This natural decomposition means that an entire contiguous range along the series can be reconstructed by using only the portion of the error-tree which is relevant to it. Furthermore, we only need to retain those coefficients whose values are significantly large, and therefore affect the values of the underlying series. In general, we would like to minimize the reconstruction error by retaining only a fixed number of coefficients, as defined by the space constraints. While wavelet decomposition is easy to perform for multi-dimensional data sets, it is much more challenging for the case of data streams. This is because data streams impose a one-pass constraint on the wavelet construction process. A variety of one-pass algorithms for wavelet construction are discussed in [41].

Histograms: The technique of histogram construction is closely related to that of wavelets. In histograms the data is binned into a number of intervals along an attribute. For any given query, the counts from the bins can be utilized for query resolution. A simple representation of the histogram method would simply partition the data into equi-depth or equi-width intervals. The main inaccuracy with the use of histograms

is that the distribution of data points within a bucket is not retained, and is therefore assumed to be uniform. This causes inaccuracy because of extrapolation at the query boundaries. A natural choice is to use an equal number of counts in each bucket. This minimizes the error variation across different buckets. However, in the case of data streams, the boundaries to be used for equi-depth histogram construction are not known a-priori. We further note that the design of equi-depth buckets is exactly the problem of quantile estimation, since the equi-depth partitions define the quantiles in the data. Another choice of histogram construction is that of minimizing the variance of frequency variances of different values in the bucket. This ensures that the uniform distribution assumption is approximately held, when extrapolating the frequencies of the buckets at the two ends of a query. Such histograms are referred to as V-optimal histograms. Algorithms for V-optimal histogram construction are proposed in [51, 52]. A more detailed discussion of several algorithms for histogram construction may be found in [4].

3.6 Dimensionality Reduction and Forecasting in Data Streams

Because of the inherent temporal nature of data streams, the problems of dimensionality reduction and forecasting and particularly important. When there are a large number of simultaneous data stream, we can use the correlations between different data streams in order to make effective predictions [70, 75] on the future behavior of the data stream. In particular, the well known MUSCLES method [75] is useful in applying regression analysis to data streams. The regression analysis is helpful in predicting the future behavior of the data stream. A related technique is the SPIRIT algorithm, which explores the relationship between dimensionality reduction and forecasting in data streams. The primary idea is that a compact number of hidden variables can be used to comprehensively describe the data stream. This compact representation can also be used for effective forecasting of the data streams. A discussion of different dimensionality reduction and forecasting methods (including SPIRIT) is provided in [4].

3.7 Distributed Mining of Data Streams

In many instances, streams are generated at multiple distributed computing nodes. An example of such a case would be sensor networks in which the streams are generated at different sensor nodes. Analyzing and monitoring data in such environments requires data mining technology that requires optimization of a variety of criteria such as communication

costs across different nodes, as well as computational, memory or storage requirements at each node. There are several management and mining challenges in such cases. When the streams are collected with the use of sensors, one must take into account the limited storage, computational power, and battery life of sensor nodes. Furthermore, since the network may contain a very large number of sensor nodes, the effective aggregation of the streams becomes a considerable challenge. Furthermore, distributed streams also pose several challenges to mining problems, since one must integrate the results of the mining algorithms across different nodes. A detailed discussion of several distributed mining algorithms are provided in [4].

4. Sensor Applications of Stream Mining

Data streams have numerous applications in a variety of scientific scenarios. In this section, we will discuss different applications of data streams and how they tie in to the techniques discussed earlier.

4.1 Military Applications

Military applications typically collect large amounts of sensor data, for their use in a variety of applications such as the detection of events and anomalies in the data. Some classic examples of military applications are as follows:

4.1.1 Activity Monitoring. Military sensors are used for a variety of scenarios such as the detection of threats movements, sounds, or vibrations in the underlying data. For example, the movement of enemy tanks in a particular region may result in a particular combination of signals detected in the sound and activity sensors. Such monitoring may require the development of heterogeneous mining and fusion techniques [73], which can combine information from multiple sources in order to perform more effective mining. Such monitoring requires stream mining methods for the continuous detection of abnormalities, or for performing continuous queries in the underlying data [21, 22, 26, 66, 79].

4.1.2 Event Detection. This is related to the problem of activity monitoring, in that specific events are captured from the stream with the use of mining techniques. This requires the design of event detection algorithms from data streams. This typically requires the use of supervised learning algorithms in which the relationship of the events to the underlying stream attributes are learned from the training data. For example, such streams are quite common in social networks, in which

it is possible to determine the key events from the underlying social network from the patterns in the underlying text stream [11]. Other general methods for event detection in streams are discussed in [3, 65, 69, 76].

4.2 Cosmological Applications

In recent years, cosmological applications have created large volumes of data. The installation of large space stations, space telescopes and observatories result in large streams of data on different stars and clusters of galaxies. This data can be used in order to mine useful information about the behavior of different cosmological objects. Similarly, rovers and sensors on a planet or asteroid may send large amounts of image, video or audio data. In many cases, it may not be possible to manually monitor such data continuously. In such cases, it may be desirable to use stream mining techniques in order to detect the important underlying properties.

The amount of data received in a single day in such applications can often exceed several tera-bytes. These data sources are especially challenging since the underlying applications may be spatial in nature. In such cases, an attempt to compress the data using standard synopsis techniques may lose the structure of the underlying data. Furthermore, the data may often contain imprecision in measurements. Such imprecisions may result in the need for techniques which leverage the uncertainty information in the data in order to improve the accuracy of the underlying results.

4.3 Mobile Applications

Recently, new technologies have emerged which have allowed the construction of *wearable sensors* in the context of a variety of applications. For example, mobile phones carry a wide variety of sensors which can continuously transmit data that can be used for *social sensing* applications [62]. Similarly, wearable sensors have been designed for continuous monitoring in a wide variety of domains such as health-care [46, 71] or vehicular participatory sensing [47]. All vehicles which have been designed since the mid-nineties carry an *OBD Diagnostic System*, which collects a huge amount of information from the underlying vehicle operation. It is possible to use the information gleaned from on-board sensors in a vehicle in order to monitor the diagnostic health of the vehicle as well as driver characterization. Another well known method is the *VEDAS* system [55], and the most well known commercialized system is the *OnStar* system designed by General Motors. Such systems require quick analysis

of the underlying data in order to make diagnostic characterizations in real time. Effective event-detection algorithms are required in order to perform this task effectively.

The stock market often creates large volumes of data streams which need to be analyzed in real time in order to make quick decisions about actions to be taken. An example of such an approach is the MobiMine approach [56] which monitors the stock market with the use of a PDA. Such methods can be used for a wide variety of applications such as knowing human movement trends [24], social image search [77], animal trends [83] grocery bargain hunting [38], or more general methods for connecting with other entities in a given neighborhood [82].

4.4 Environmental and Weather Data

Many satellites and other scientific instruments collect environmental data such as cloud cover, wind speeds, humidity data and ocean currents. Such data can be used to make predictions about long- and short-term weather and climate changes. Such data can be especially massive if the number of parameters measured are very large. The challenge is to be able to combine these parameters in order to make timely and accurate predictions about weather driven events. This is another application of event detection techniques from massive streams of sensor data.

In particular, such methods have found numerous applications in prediction of long-term climate change [40, 58, 67]. For example, one can use various environmental parameters collected by sensors to predict changes in sea surface temperatures, indicators specific to global warming, or the onset of storms and hurricanes. A detailed discussion on the application of such methods for climate and weather prediction may be found in [40].

5. Conclusions and Research Directions

Data streams are a computational challenge to data mining problems because of the additional algorithmic constraints created by the large volume of data. In addition, the problem of temporal locality leads to a number of unique mining challenges in the data stream case. This chapter provides an overview to the generic issues in processing data streams, and the specific issues which arise with different mining algorithms.

While considerable research has already been performed in the data stream area, there are numerous research directions which remain to be explored. Most research in the stream area is focussed on the one pass constraint, and generally does not deal effectively with the issue of temporal locality. In the stream case, temporal locality remains an

extremely important issue since the data may evolve considerably over time. Other important topics which need to be explored are as follows:

- Streams are often collected by devices such as sensors in which the data is often noisy and error-driven. Therefore, a key challenge is to effectively clean the data. This may involve either imputing or modeling the underlying uncertain data. This can be challenge, since any modeling needs to be done in real time, as large volumes of the data stream arrive.

- A related area of research is in using the modeled data for data mining tasks. Since the underlying data is uncertain, the uncertainty should be used in order to improve the quality of the underlying results. Some recent research addresses the issue of clustering uncertain data streams [7].

- Many recent applications such as privacy-preserving data mining have not been studied effectively in the context of data streams. It is often a challenge to perform privacy-transformations of continuously arriving data, since newly arriving data may compromise the integrity of earlier data. The data stream domain provides a number of unique challenges in the context of the privacy problem.

References

[1] Aggarwal C., Xie Y., Yu P. (2011) On Dynamic Data-driven Selection of Sensor Streams, *ACM KDD Conference*.

[2] Aggarwal C., Bar-Noy A., Shamoun S. (2011) On Sensor Selection in Linked Information Networks, *DCOSS Conference*.

[3] Abadi D., Madden S., Lindner W. (2005) REED: robust, efficient filtering and online event detection in sensor networks, *VLDB Conference*.

[4] Aggarwal C. (2007) Data Streams: Models and Algorithms, *Springer*.

[5] Aggarwal C., Procopiuc C, Wolf J. Yu P., Park J.-S. (1999) Fast Algorithms for Projected Clustering. *ACM SIGMOD Conference*.

[6] Aggarwal C. (2006) On Biased Reservoir Sampling in the presence of Stream Evolution. *VLDB Conference*.

[7] Aggarwal C., Yu P. (2008) A Framework for Clustering Uncertain Data Streams. *ICDE Conference*.

[8] Aggarwal C. (2003) A Framework for Diagnosing Changes in Evolving Data Streams. *ACM SIGMOD Conference*.

[9] Aggarwal C. (2002) An Intuitive Framework for understanding Changes in Evolving Data Streams. *IEEE ICDE Conference*.

[10] Aggarwal C., Han J., Wang J., Yu P (2003). A Framework for Clustering Evolving Data Streams. *VLDB Conference*.

[11] Aggarwal C., Han J., Wang J., Yu P (2004). A Framework for High Dimensional Projected Clustering of Data Streams. *VLDB Conference*.

[12] Aggarwal C., Yu P. (2006) A Framework for Clustering Massive Text and Categorical Data Streams. *SIAM Data Mining Conference*.

[13] Aggarwal C, Han J., Wang J., Yu P. (2004). On-Demand Classification of Data Streams. *ACM KDD Conference*.

[14] Aggarwal C. (2009). *Managing and Mining Sensor Data*, Springer.

[15] Aggarwal C., Yu P. (2007). On Density-based Transforms for Uncertain Data Mining, *ICDE Conference*, 2007.

[16] Agrawal R., Imielinski T., Swami A. (1993) Mining Association Rules between Sets of items in Large Databases. *ACM SIGMOD Conference*.

[17] Alon N., Gibbons P., Matias Y., Szegedy M. (1999) Tracking Joins and Self-Joins in Limited Storage. *ACM PODS Conference*.

[18] Alon N., Matias Y., Szegedy M. (1996) The Space Complexity of Approximating Frequency Moments. *The Space Complexity of Approximating Frequency Moments*, pp. 20–29.

[19] Arici T., Akgun T., Altunbasak Y. (2006) A prediction error-based hypothesis testing method for sensor data acquisition. *ACM Transactions on Sensor Networks (TOSN)*, Vol. 2, pp. 529–556.

[20] Babcock B., Datar M., Motwani R. (2002) Sampling from a Moving Window over Streaming Data. *SIAM Symposium on Discrete Algorithms (SODA)*.

[21] Babcock B., Olston C. (2003) Distributed top-k monitoring, *ACM SIGMOD Conference*.

[22] Babu S., Widom J. (2001) Continuous queries over data streams. *SIGMOD Record*, 30(3), pp. 109–120.

[23] Bonnet P, Gehrke J., Seshadri P. (2001) Towards sensor database systems, *International Conference on Mobile Data Management*.

[24] Calabrese F., Kloeckl K., Ratti C. (2007) Wikicity: Real-Time Urban Environments. *IEEE Pervasive Computing*, 6(3), pp. 52-53. http://senseable.mit.edu/wikicity/rome/

[25] Cao F., Ester M., Qian W., Zhou W. (2006) Density-based Clustering of an Evolving Data Stream with Noise. *SIAM Data Mining Conference*.

[26] Chandrasekaran S., Franklin M. (2002) Streaming queries over streaming data. *VLDB Conference*.

[27] Chang J. H., Lee W. S. (2003) Finding recent frequent itemsets adaptively over online data streams. *ACM KDD Conference*.

[28] Chen Y., Tu L. (2007) Density-based Clustering for Real Time Stream Data, *ACM KDD Conference*.

[29] Chi Y., Wang H., Yu P., Muntz R. (2004) Moment: Maintaining closed frequent itemsets over a stream sliding window. *ICDM Conference*.

[30] Cormode G., Garofalakis M. (2005) Sketching Streams Through the Net: Distributed Approximate Query Tracking. *VLDB Conference*.

[31] Cormode G., Muthukrishnan S. (2004) An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *LATIN*, pp. 29–38.

[32] Cormode, G., Muthukrishnan, S., Zhuang, W. (2007) Conquering the divide: Continuous clustering of distributed data streams. *ICDE Conference*.

[33] Datar M., Gionis A., Indyk P., Motwani R. (2002) Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813.

[34] Deligiannakis A., Kotidis Y., Roussopoulos N. (2004) Compressing Historical Information in Sensor Networks. *ACM SIGMOD Conference*.

[35] Deligiannakis A., Kotidis Y. (2005) Data Reduction Techniques in Sensor Networks. *IEEE Data Engineering Bulletin* 28(1): pp. 19–25.

[36] Deshpande A., Guestrin C., Madden S, Hellerstein J., Hong W. Model-driven data acquisition in sensor networks, *VLDB Conference*, 2004.

[37] Dasu T., Krishnan S., Venkatasubramaniam S., Yi K. (2005). An Information-Theoretic Approach to Detecting Changes in Multi-dimensional data Streams. *Duke University Technical Report CS-2005-06*.

[38] Deng L., Cox L. (2009) Livecompare: grocery bargain hunting through participatory sensing. *HotMobile*.

[39] Domingos P., Hulten G. (2000) Mining High-Speed Data Streams. In *Proceedings of the ACM KDD Conference*.

[40] Garg A. et al (2011) Gopher: Global observation of Planetary Health and Ecosystem Resources. *IGARSS*, pp. 1449–1452.

[41] Garofalakis M., Gehrke J., Rastogi R. (2002) Querying and mining data streams: you only get one look (a tutorial). *SIGMOD Conference*.

[42] Gianella C., Han J., Pei J., Yan X., Yu P. (2002) Mining Frequent Patterns in Data Streams at Multiple Time Granularities. *NSF Workshop on Next Generation data Mining*.

[43] Guha S., Mishra N., Motwani R., O'Callaghan L. (2000). Clustering Data Streams. *IEEE FOCS Conference*.

[44] Giannella C., Han J., Pei J., Yan X., and Yu P. (2002) Mining Frequent Patterns in Data Streams at Multiple Time Granularities. *Proceedings of the NSF Workshop on Next Generation Data Mining*.

[45] Golovin D., Faulkner M., Krause A. (2010) Online distributed sensor selection. *IPSN Conference*.

[46] Holter N., Generelli J. (1949) Remote recording of physiologic data by radio. *Rocky Mountain Medical Journal*, pp. 747-751.

[47] Hull B., Bychkovsky V., Chen K., Goraczko M., Miu A., Shih E., Zhang Y., Balakrishnan H., Madden S. (2006) CarTel: A Distributed Mobile Sensor Computing System, *ACM SenSys*.

[48] Hulten G., Spencer L., Domingos P. (2001) Mining Time Changing Data Streams. *ACM KDD Conference*.

[49] Indyk P. (2000) Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation. *IEEE FOCS*.

[50] Jin R., Agrawal G. (2003) Efficient Decision Tree Construction on Streaming Data. *ACM KDD Conference*.

[51] Ioannidis Y., Poosala V. (1995) Balancing Histogram Optimality and Practicality for Query Set Size Estimation. *ACM SIGMOD Conference*.

[52] Jagadish H., Koudas N., Muthukrishnan S.,Poosala V.,Sevcik K., Suel T. (1998) Optimal Histograms with Quality Guarantees. *VLDB Conference*.

[53] Jin R., Agrawal G. (2005) An algorithm for in-core frequent itemset mining on streaming data. *ICDM Conference*.

[54] Johnson W., Lindenstrauss J. (1984) Extensions of Lipshitz mapping onto Hilbert Space. *Contemporary Mathematics*, Vol 26, pp. 189–206.

[55] Kargupta H. et al VEDAS: A Mobile and Distributed Data Stream Mining System for Vehicle Monitoring. *SDM Conference*, 2004.

[56] Kargupta H. et al MobiMine: Monitoring the stock market using a PDA, *ACM SIGKDD Explorations*, January 2002.

[57] Kasetty S., Stafford C., Walker G., Wang X., Keogh E. (2008) Real-Time Classification of Streaming Sensor Data, *ICTAI Conference*.

[58] Kawale J., Steinbach M., Kumar V. (2011) Discovering Dynamic Dipoles in Climate Data. *SDM Conference*.

[59] Kifer D., David S.-B., Gehrke J. (2004). Detecting Change in Data Streams. *VLDB Conference*, 2004.

[60] Kollios G., Byers J., Considine J., Hadjieleftheriou M., Li F. (2005) Robust Aggregation in Sensor Networks. *IEEE Data Engineering Bulletin*.

[61] Krause A., Guestrin C. (2007) Near-optimal observation selection using submodular functions. *AAAI Conference*.

[62] Krause A., Horvitz E., Kansal A., Zhao F. (2008) Toward Community Sensing. *IPSN*, pp. 481–492.

[63] Madden S., Franklin M., Hellerstein J. (2005) TinyDB: an acquisitional query processing system for sensor networks, *ACM Transactions on Database Systems*, 30(1), pp. 122–173.

[64] Manku G., Motwani R. (2002) Approximate Frequency Counts over Data Streams. *VLDB Conference*.

[65] Medioni G., Cohen I., Bremond F., Hogeng S., Nevatia R. (2001) Event Detection and Analysis from Video Streams, *IEEE TPAMI*, 23(8).

[66] Olston C., Jiang J., Widom J. (2003) Adaptive Filters for Continuous Queries over Distributed Data Streams. *SIGMOD Conference*.

[67] Race C., Steinbach M., Ganguly A., Semazzi F., Kumar V. (2010) A Knowledge Discovery Strategy for Relating Sea Surface Temperatures to Frequencies of Tropical Storms and Generating Predictions of Hurricanes Under 21st century Global Warming Scenarios, *CIDU*, pp. 204–212.

[68] Rodrigues P., Gama J., Lopes L. (2008) Clustering Distributed Sensor Data Streams, *PKDD Conference*.

[69] Sakaki T., Okazaki M., Matsuo Y. (2010) Earthquake shakes Twitter users: real-time event detection by social sensors, *WWW Conference*.

[70] Sakurai Y., Papadimitriou S., Faloutsos C. (2005). BRAID: Stream mining through group lag correlations. *ACM SIGMOD Conference*.

[71] Sung M., Marci C., Pentland A. (2005) Wearable Feedback Systems for Rehabilitation, *Journal of Neuroengineering and Rehabilitation*, 2:17.

[72] Vitter J. S. (1985) Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, Vol 11(1), pp. 37–57.

[73] Waltz E., Llinas J. (1990) Multi-Sensor Data Fusion, *Artech House Radar Library*.

[74] Wang H., Fan W., Yu P., Han J. (2003) Mining Concept-Drifting Data Streams using Ensemble Classifiers. *ACM KDD Conference*.

[75] Yi B.-K., Sidiropoulos N.D., Johnson T., Jagadish, H. V., Faloutsos C., Biliris A. (2000). Online data mining for co-evolving time sequences. *ICDE Conference*.

[76] Xue W., Luo Q., Chen L., Liu Y. (2006) Contour Map Matching for Event Detection in Sensor Networks, *ACM SIGMOD Conference*.

[77] Yan T., Kumar V., Ganesan D. (2010) Crowdsearch: Exploiting crowds for accurate real-time image search on mobile phones, *MobiSys*.

[78] Yao Y., Gehrke J. (2003) Query processing in sensors networks, *First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*.

[79] Yin J., Yang Q., Pan J. (2008) Sensor-based Abnormal Human Activity Detection, *IEEE TKDE*, 20(8), pp 1082–1090.

[80] Yu J. X., Chong Z., Lu H., Zhou A. (2004) False positive or false negative: Mining frequent itemsets from high speed transaction data streams. *VLDB Conference*.

[81] Zhang T., Ramakrishnan R., Livny M. (1996) BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Conference*.

[82] http;//www.wikitude.com

[83] http://www.movebank.org