Mining data streams with periodically changing distributions

ABSTRACT

Dynamic data streams are those whose underlying distribution changes over time. They occur in a number of application domains, and mining them is important for these applications. Coupled with the unboundedness and high arrival rates of data streams, the dynamism of the underlying distribution makes data mining challenging. In this paper, we focus on a large class of dynamic streams that exhibit periodicity in distribution changes. We propose a framework, called DMM, for mining this class of streams that includes a new change detection technique and a novel match-and-reuse approach. Once a distribution change is detected, we compare the new distribution with a set of historically observed distribution patterns and use the mining results from the past if a match is detected. Since, for two highly similar distributions, their mining results should also present high similarity, by matching and reusing existing mining results, the overall stream mining efficiency is improved while the accuracy is maintained. Our experimental results confirm this conjecture.

1. INTRODUCTION

Mining data streams for knowledge discovery, such as security protection [18], clustering and classification [2], and frequent pattern discovery [12], has become increasingly important. Within this context, an additional characteristic of the unbounded data streams is that the underlying distribution can show important changes over time, leading to *dynamic data streams*. Traditional data mining techniques that make multiple passes over data or that ignore distribution changes are not applicable to dynamic data streams.

In this paper, we address the problem of mining dynamic data streams. This problem is particularly challenging when one tries to balance accuracy with efficiency – highly accurate mining techniques are generally computationally expensive. Many change detection techniques for dynamic streams have been proposed in literature (e.g., [1, 12, 14, 15]). However, some of these techniques can only generate an alarm to indicate that a distribution change has occurred, without providing new mining results corresponding to the new distribution. Others follow the "detect-and-mine" approach where the distribution change is detected and mining is adjusted between changes, but many of these techniques are ad-hoc in that they are only suitable for one type of mining application, and they usually only perform well on certain types of streams.

For dynamic data streams, an interesting question is whether the distribution changes are entirely random and unpredictable, or whether it is possible for the distribution changes to follow certain patterns. If some regularity can be detected, this can be exploited in mining. An analysis of different applications and data streams reveals that there is a large class of data streams that exhibit periodic distribution changes; that is, a distribution that has occurred in the past generally reappears after a certain time period. Consider, for example, a scientific stream that records sea surface temperature [17]. There is periodicity to the distribution of collected data that may not be visible within one year, but is apparent over multiple years. Other examples of applications that may present periodically changing distributions include financial data streams such as stock price [24], scientific streams such as magnetic resonance imaging (MRI) readings [5], and network traffic.

1.1 Our proposal

Based on this observation, we propose a method for mining dynamic data streams where we store important (to be defined later) observed distribution patterns, and compare new detected changes with these patterns. For two substreams with the same (or highly similar) distributions, mining results such as a list of all frequent items/itemsets for frequent pattern discovery, and set of clusters/classes for clustering and classifications, should be the same (or highly similar). Therefore, if a distribution change is detected and if a match is found between the new distribution and an archived one, then it is possible to skip the re-mining process and directly output the mining results for the archived distribution as the results for the newly detected distribution. We call this the *match-and-reuse* strategy. Compared with the traditional detect-and-mine stream mining approaches, our match-and-reuse strategy is faster for periodically changing streams, since pattern matching is usually considerably less time consuming than mining.

The key issues that need to be resolved in our approach are the pattern selection, pattern representation, and matching. Since the historical patterns need to be stored along with their mining results, typical memory restriction in data stream systems make it infeasible to store every pattern that has ever occurred in the past. Hence, only important distributions, i.e., the ones that have a high probability to reoccur in the future, should be archived. Being able to determine these distributions is the first important issue. Second, each pattern needs to be stored as succinctly as possible - ideally by representing a distribution (pattern) using its density function. However, it is not possible to get an accurate density function for a random set of data elements. Hence, in data stream mining, the most popular approach is to represent the current distribution using a representative data set. Intuitively, the larger the size of the representative set, the closer is the distribution of this set to the real distribution. However, memory limitation forces the size of each representative set to be as small as possible. To find such a small representative set that can accurately reflect the true distribution of each pattern is a challenge. The third issue is that the matching procedure needs to be efficient for rapid data streams, with an accuracy high enough to meet the application requirements.

Similarity matching over time series has been extensively studied (e.g. [3, 10, 13, 16]). However, the distribution matching problem is far more complicated than time series matching in a number of ways. First, the sample patterns in time series matching are usually predefined. These well-chosen predefined patterns are very representative and usually noise-free. In our case, the patterns are directly extracted from the stream and may contain noise. Second, in time series data, each data element contains only one value. Streams generated from real-world are far more complex and have different forms. Many of the real-world stream elements have the form of relational tuples with multiple attributes. Hence, many of the time series matching techniques are too simple to be directly applicable on real-world streams. Third, a number of distance functions have been proposed for pattern matching. Each of these distance functions has its advantages and disadvantages. It is not clear which distance function would best fit a particular type of application and stream, nor is it known how to tune the distance threshold accordingly.

Our proposed match-and-reuse approach deals with these problems. For each important distribution, a set of representative sample data is extracted from the stream, which is continuously refined when new data arrive. Experimental results demonstrate that our distribution change detection technique using incrementally maintained representative set can achieve high accuracy. Experiments also reveal that, by reusing existing mining results on the newly detected distribution after a match is found, the overall efficiency of the mining application can be greatly improved without losing accuracy.

1.2 Contributions

In summary, the contributions of this paper are as follows:

- We propose a novel method for matching patterns and reusing mining results for mining a data stream with periodically changing distributions. Compared to previous work, this approach can reduce the overall mining time without affecting accuracy of the mining results.
- We develop a framework for detecting changes and generating mining results for dynamic data streams using the proposed match-and-reuse strategy. This framework is flexible. The detection process is independent of the mining process, and hence this framework can be used in most stream mining applications.
- We consider the issue of representing distributions of the stream. Existing techniques use the data set within a sliding window as the representative set for current distribution. This method is problematic, as will be discussed in more detail in Section 3.2. We propose a new technique to intelligently choose a small set of data as the representative set that can reflect the true distribution with high accuracy.

1.3 Paper organization

The rest of this paper is organized as follows. The problem is formalized in Section 2. We propose our stream mining framework in Section 3. In Section 4, we analyze and compare some of the popular distance functions. Experiments are presented in Section 5. Section 6 gives a brief review of the related work. We conclude the paper in Section 7.

2. PROBLEM DEFINITION

A data stream S is an unbounded sequence of elements $\langle s, t\rangle$, where s is the data element and t is a monotonically increasing timestamp indicating the arrival time of the element. s can have different forms, such as a relational tuple, or a set of items, depending on the underlying application that generates S. t can be either an explicit timestamp or an implicit timestamp [4]. In this paper, the exact assignment of timestamps is not important as long as they are monotonic. Each stream mining application may only consider a few attributes/items of the stream. The unconsidered attributes/items can be filtered before data are fed to the mining process.

A substream $S_i = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, ..., \langle s_k, t_k\rangle\}$ of stream S is a finite set of elements that occurs in S, i.e., $\langle s_j, t_j\rangle \in S$, $j = 1, ..., k$. Similar to [15], we define the probability distribution of a substream S_i as the frequency distribution of all values within this substream, without considering their arrival time.

The following discussion and definitions assume the existence of two substreams S_A and S_B, with probability distributions P_A and P_B, respectively. As usual, we define the similarity of S_A and S_B as $1 - dist(S_A, S_B)$, where $dist(S_A, S_B)$ denotes the *distance* between S_A and S_B. As discussed in Section 1, in this work, we consider two substreams as two bags of elements observed during different time period. Therefore, we define the similarity between two substreams as the similarity of their distributions, i.e., $dist(S_A, S_B) = dist(P_A, P_B)$.

The distance between two substreams can be computed by a distance function. There are a number of distance functions that can be used. Specific functions for distribution matching in data streams have been proposed [15], but functions that have been defined for time series can also be used by appropriate conversions: the time axis and the attribute value axis in these functions should be replaced with value axis and probability value axis, respectively. Consider, for example, the computation of Euclidean distance over the two distributions P_A and P_B given in Figure 1, where $v(s)$ is the value domain of both substreams S_A and S_B, $p_A(v_i)$ and $p_B(v_i)$ are the probability of each $v_i \in v(s)$ occurring in S_A and S_B, respectively. Then the distance between P_A and P_B can be calculated as

$$dist_{euclidean}(P_A, P_B) = \sqrt{\sum_{i}^{n}(p_A(v_i) - p_B(v_i))^2}$$

where n is the total number of distinct values in $v(s)$.

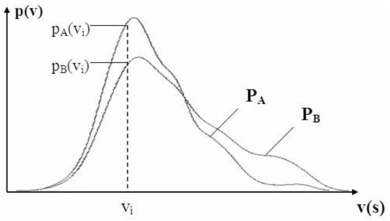

Figure 1: Distributions for substream S_A and S_B

Definition 1. Given S_A, S_B, P_A and P_B as defined above, if $dist(P_A, P_B) < \delta$, where δ is a predefined threshold referred to as *maximum matching distance*, then we say that the distribution of substreams S_A and S_B match each other, denoted as $P_A \rightleftharpoons_\delta P_B$.

The main idea of this paper (reusing existing mining results for periodically changing data streams) can now be formalized as follows. Let K be the currently running stream mining application over data stream S with periodically changing distribution. Let $\mathcal{P} = \{P_1, ..., P_m\}$ be a set of preserved important distributions in S for K, and $\mathcal{R} = \{R_1, ..., R_m\}$ be the corresponding mining results where R_i is the result obtained during the period that distribution P_i was in effect. Let P' be the most recently detected distribution, if $P' \rightleftharpoons_\delta P_i$, then the result of mining S during the period when P' is effective is R_i, i.e., $R(P') = R_i$. Otherwise, K is executed on P' to generate the result.

Due to the unboundedness of data streams and their high arrival rates, at any given time, only a portion of a stream can be monitored. Thus, window models are widely used in many stream techniques, and they are mainly of two types: time-based window and count-based window.

Time-based window is defined in terms of a time interval Δ. It contains a substream of S that arrives within time $\langle t, t+\Delta\rangle$. On the other hand, a count-based window contains a substream of S with fixed number of elements. The size of a window is the total number of data elements within it. The size of a count-based window is fixed, while the size of a time-based window can change, if the arrival rate of S is not stable.

Definition 2. Let W_A and W_B be two windows (either time-based or count-based) on stream S, containing substreams S_A and S_B, respectively. We define the *concatenation* of windows W_A and W_B, denoted as $W_A + W_B$, as the set union of the two substreams within them, i.e., $W_A + W_B = S_A \cup S_B$. Therefore, $\forall\langle s_i, t_i\rangle$ in $W_A + W_B$, $\langle s_i, t_i\rangle \in S_A$ or $\langle s_i, t_i\rangle \in S_B$.

Notice that although we do not consider the arrival time of the elements when calculating the distribution, we do not remove the timestamps attached to them. Therefore, two elements $\langle s_i, t_i\rangle$ and $\langle s_j, t_j\rangle$ with the same values, i.e., $s_i = s_j$, are not considered duplicates in $W_A + W_B$. This guarantees that the probability of a value that occurs in many elements will be calculated correctly. Thus, $|W_A+W_B| = |W_A|+|W_B|$ when W_A and W_B do not overlap.

Table 1 summarizes the main symbols used in this paper. Note that some of the symbols are introduced in later sections.

Table 1: Meanings of symbols used

Symbols	Meanings
S	data stream
s	data elements in S
$v(s)$	value of data element s
$p(v)$	probability of value v
$W_r(W'_r)$	reference window W_r - before update; W'_r - after update
W_t	observation window
S_r, S_t	substreams in W_r, W_t
P_r, P_t	distributions of S_r, S_t
δ	maximum matching distance
Δ	tumbling time interval
G_i	partition of a distribution
s_{G_i}	representative point of G_i
g_i^S, g_i^E	start and end point for each G_i
k	total number of partitions
$\mathcal{P}$	important distribution set

3. DMM FRAMEWORK

In this paper, we propose a framework called DMM (which stands for Detect, Match, and Mine) for mining data streams with periodically changing distributions. Briefly, DMM consists of four sequential procedures: representative set choosing, change detection, pattern matching, and (if pattern matching fails) stream mining.

Change detection procedure (Section 3.3) runs throughout the lifetime of a stream S to monitor its distribution changes. Once a new distribution is detected, a representative set is generated for this new distribution and is incrementally maintained during its lifespan (Section 3.2). This new distribution is matched against those that have been seen and archived earlier (Section 3.4). If a match is found between the new distribution and an archived one, DMM skips the re-mining process and outputs the mining results of the archived distribution, because two substreams with similar distributions should have similar data mining results. Consequently, processing time is greatly reduced. On the other hand, if pattern matching fails (i.e., no similar distribution has been seen previously), then the mining process is activated to generate new results for this new distribution. A set of "important" distributions that have already been observed is maintained. Since it may not be possible to store all observed distributions due to space restrictions, a heuristic approach is applied to determine "important" distributions for storage. When a new distribution is detected, this set is updated if appropriate.

Note that these four procedures are independent of each other. This provides considerable flexibility as it is possible to plug in any change detection and mining technique, and change the distance function for pattern matching at any point if the application requirements are altered.

3.1 Windows model

We maintain two windows on a stream S: a count-based window W_r and a time-based window W_t. As mentioned in Section 1.1, since the distribution of the current stream cannot be directly extracted, we use a set of elements in this stream as a representative set to represent its distribution. The substream S_r in W_r represents the current distribution, and substream S_t in W_t records the set of data elements that have arrived in the last Δ time units. We call W_r the *reference window* and W_t the *observation window*.

The observation window W_t is implemented as a tumbling window. Every Δ time units, W_t "tumbles" so that all the elements in it are deleted and a new empty window is opened. We implement W_t as a tumbling window rather than the more common sliding window because of performance considerations. A time-based sliding window moves forward at each clock time (i.e., when time moves forward), and, all items in the window with a timestamp less than $t_{now} - t_{window-duration}$ are evicted. Since our change detection process is triggered every time W_t moves, using a sliding window would greatly impact the performance when the stream has high arrival rate.

The size of reference window W_r ($|W_r|$) and the time interval Δ of observation window W_t are pre-defined values. $|W_r|$ indicates the size of the representative set for each distribution. Hence, if many important distributions (we will precisely define important distributions in Section 3.5) are expected during the lifespan of S, then $|W_r|$ should be smaller. However, intuitively, the larger the representative set S_r, the closer its distribution is to the real distribution of S. For a given minimum accuracy requirement a, $|W_r|$ should satisfy

$$Prob(|dist(P_r, P_S)| \geq \delta) < e^{-a|W_r|}$$

where P_r and P_S are the distributions of substream S_r in W_r and the real current distribution in stream S, respectively. $|W_r|$ can be determined either manually or by running a set of training data at the beginning of S.

As mentioned above, the time interval Δ of W_t indicates the frequency of triggering the change detection procedure. Smaller Δ values would mean more frequent change detection, and thus, the number of delayed alarms will be fewer. On the other hand, smaller Δ values will reduce the efficiency of our change detection technique. Therefore, Δ value should be determined according to the minimum efficiency requirement of the relevant application.

3.2 Generating reference window

3.2.1 Motivation

Reference window W_r is used to store a set of data elements that represent current distribution of S. Change detection can be done by comparing the distribution of the substream in W_r and the distribution of the newly arrived data , i.e., the substream in W_t. This is discussed in detail shortly.

Statistically, the more samples collected (i.e., the larger $|W_r|$), the better the underlying distribution can be described. However, in reality, the size of W_r needs to be relatively small due to memory limitations. Furthermore, efficiency of distribution matching and change detection is related to the size of the sample sets. Therefore, for a mining application with high efficiency requirement, W_r cannot be too large. However, using a small data set to represent a distribution could be highly inaccurate, especially when the distribution is complicated.

In most of the existing work, representative set of the current distribution is set to be the substream in a window that starts when the distribution change is detected. This may be problematic since the new distribution may not be properly captured within this window (unless the window is very large). Furthermore, for slow distribution changes, it may be a while until the new distribution is stabilized. Hence, the substream captured immediately after the distribution change cannot reflect the real distribution. Therefore, this substream is not suitable to be the representative set of the current distribution in S.

Although the size of the representative set has to be limited due to memory and efficiency concerns, if, instead of blindly using the first group of data that arrived after a new distribution is detected, we can carefully choose a sample set with the same size from a large vault of samples, then the distribution of this selected set will be much closer to the true distribution. Based on this idea, we introduce the concept of *dynamic reference window* W_r for solving the problem of representing a (complicated) distribution using a small sample set accurately.

3.2.2 Merge and select process

The overview of generating the dynamic reference window W_r is as follows. We concatenate/merge windows W_r and W_t so that a larger substream with at most $|W_r| + |W_t|$ elements is available. We then carefully select $|W_r|$ elements from the concatenated window $W_r + W_t$ as the latest representative set, and replace the substream in W_r by this new set[1]. This merge-and-select process is triggered every time W_t tumbles. Hence, elements in W_r can be regarded as a "concentration" of all data from the current distribution that have been observed so far. In other words, this representative set is a careful selection from the large vault of observed elements since the beginning of the distribution. Thus, substream S_r in W_r is highly representative of the current distribution. This merge-and-select process is illustrated in Figure 2.

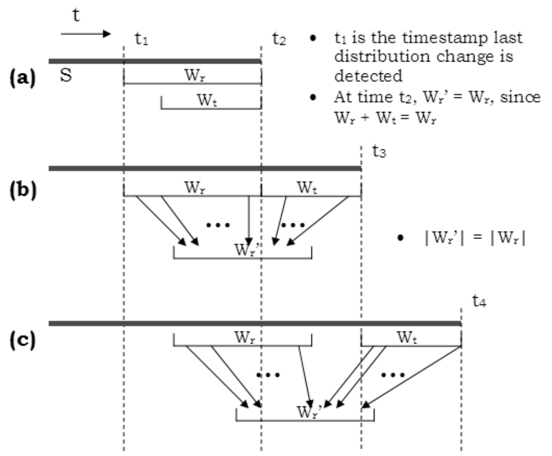
t
t1
t2
t1 is the timestamp last distribution change is detected
(a)
S
Wr
Wt
At time t2, Wr' = Wr, since Wr + Wt = Wr
t3
(b)
Wr
Wt
Wr'
|Wr'| = |Wr|
t4
(c)
Wr
Wt
Wr'

Figure 2: Reference window generation

Let a distribution change be detected at time t_1. Staring at t_1, W_r records a substream S_r that contains the first $|W_r|$ observed elements. Observation window W_t contains substream S_t with the newest $|W_t|$ elements (Figure 2a). At time t_2, W_t is full and ready to tumble forward. If at this point a distribution change is not detected, then we "merge" the data elements in W_t and W_r to form a data set with size $|W_r + W_t|$, denoted as $W_t + W_r$ as per Definition 2 (Figure 2b). We then select $|W_r|$ elements (will be discussed in Section 3.2.3) from $W_t + W_r$, so that the selected data set can better represent the current distribution (Figure 2c). In other words, for the substreams S_r and S'_r with distributions P_r and P'_r, respectively, P'_r is more similar to the true underlying distribution P of stream S (i.e.,

[1]In the rest of the paper, we use W_r and W'_r to denote the reference windows before and after merge-and-select process, respectively.

$dist(P'_r, P) \leq dist(P_r, P))$. This merge-and-select process continues until the next time a distribution change occurs.

3.2.3 Selecting representative set

Ideally, we want to find the data set W'_r in $W_r + W_t$, such that its distribution is the closest to the true distribution of S, i.e., $dist(P'_r, P) = min(dist(\forall P^i_r, P))$, where P^i_r is the distribution of any substream in $W_r + W_t$ with size $|W_r|$. However, finding such a substream can be computationally expensive. For stream applications with high efficiency requirement, an approximate algorithm is required.

The goal for our approximation is to find a substream W'_r with size $|W_r|$ from $W_r + W_t$, such that:

$$dist(P'_r, P) \leq min(dist(P_t, P), dist(P_r, P)) \tag{1}$$

To achieve this goal, we propose a two-step sampling approach. We first estimate the density function of the distribution of data set in $W_r + W_t$. The density function is estimated using the popular kernel density estimation [19]:

$$\hat{f}_h(s) = \frac{1}{(|W_r + W_t|)h} \sum_{i=1}^{|W_r+W_t|} K(\frac{s - s_i}{h}) \tag{2}$$

where K is the *kernel function*, h is the smoothing parameter called *bandwidth*, and s_i is a data element in $W_r + W_t$. The intuition of kernel density estimation is to replace each data element in the set with a smoothed "bump", whose width is determined by h. The resulting density function is the sum of all the bumps. In this paper, we set K to be the standard Gaussian function with mean zero and variance 1. This setting is usually chosen when no pre-knowledge of the distribution is available [21]. Thus,

$$K(s) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}s^2} \tag{3}$$

Hence, equation (2) can be rewritten as

$$\hat{f}_h(s) = \frac{1}{(|W_r + W_t|)\sqrt{2\pi h^2}} \sum_{i=1}^{|W_r+W_t|} e^{-(s-s_i)^2/2h^2} \tag{4}$$

The bandwidth h is a value between 0 and 1. A large h setting is preferable if the distribution has a long lifespan over the stream, whereas a small h value is suitable for a short trend. If the lifespan of current distribution is unpredictable, then it is reasonable to start from a small setting (e.g., $h = 0.1$), and increase h each time W_r is updated until the next distribution change is detected.

Although the kernel density estimation approach can generate an estimated density function of the current distribution, the accuracy of this estimation is usually low. This is because kernel density estimation assumes that the underlying stream should have an approximately normal distribution. For complex streams, this assumption does not hold. Performing a match by simply calculating the distance of estimated density introduces a large error (as demonstrated in Sections 5.1 and 5.2). On the other hand, representative sets consist of elements directly extracted from the stream, and thus are more reliable. Hence, we only use kernel density estimation as a preliminary guidance for our sampling strategy.

In the rest of the discussion, we consider only single-dimensional streams, i.e., the data element s in the stream contains only one attribute. Our technique can be extended to N-dimensional streams by replacing each single-attribute element s with a N-dimensional vector $\bar{s} = (s_1, ..., s_N)$, but the complexity of the algorithm may increase considerably for streams with high dimensions.

We use Figure 3 to illustrate the idea of our two-step sampling approach. With the density function $\hat{f}_h(s)$, we are able to estimate the "shape" of the current distribution. x-axis is the value of the data s ($v(s)$) in $W_r + W_t$, and y-axis is the probability ($p(v)$) for all data values. Higher $p(v)$ values indicate that these values occur more frequent in the stream, and thus more elements s with value $v(s)$ should be selected into the new reference window W'_r, and vice versa. Based on this insight, in the first step, we partition the whole area under the density function $\hat{f}_h(s)$ into k disjoint groups $G_1, G_2, ..., G_k$, where k is a constant for each $\hat{f}_h(s)$ (we discuss k value selection shortly). The start and end value for each partition G_i is denoted as g^S_i and g^E_i, respectively. Each partition has the same area, i.e., $area(G_1) = ... = area(G_k) = \frac{1}{k}area(\hat{f}_h(s))$.

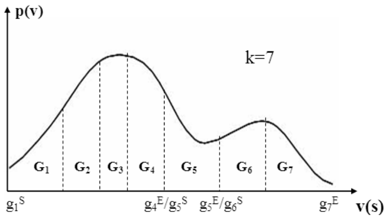

Figure 3: Example of two-step sampling

The second step is sampling from each partition. For two partitions G_i and G_j, if $g^E_i - g^S_i < g^E_j - g^S_j$ (i.e., if the value range for G_i is "narrower" than G_j), then the average probability of all values within partition G_i is greater than the average probability of values within partition G_j. If we select the same number of data elements with values within G_i and G_j, respectively, then the selected values within range $\langle g^S_j, g^E_j \rangle$ are more "sparse" than in $\langle g^S_i, g^E_i \rangle$. Hence, the shape of this distribution is better captured by the data set selected using this technique. We use S_{Gi} to represent the elements selected from each partition G_i. Since the resulting representative set is stored in W_r, the total number of data selected from each partition should be $|W_r|$, i.e., $\sum_{i=1}^{k} |S_{Gi}| = |W_r|$.

To partition $W_r + W_t$ into a number of groups G_i ($i = 1, ..., k$), we adopt a machine learning method [25] that is used for clustering real-valued attributes into disjoint subregions. We define a discretizer function Q for partitioning. For each partition G_i, every data element s with value $v(s) \in G_i$ maps (or rounds off) to a representative point s_{G_i} of G_i[2], such that:

$$Q(W_r + W_t) = \sum_{i=1}^{k} s_{G_i} 1_{G_i(s)}, s \in (W_r + W_t) \tag{5}$$

where $1_{G_i(s)}$ is the indicator function of G_i, i.e.,

$$1_{G_i(s)} = \begin{cases} 1 & \text{if } v(s) \in G_i; \\ 0 & \text{if } v(s) \notin G_i \end{cases}$$

In the optimal case, we want to find a discretizer function Q^O such that the error of mapping sample data to their representative point is minimized, i.e.,

$$err(Q^O) = lim_{k \to (|W_r+W_t|)} \sum_{i=1}^{k} \int_{G_i} |s - s_{G_i}|^2 \hat{f}_h(s) ds = 0 \tag{6}$$

Equation (6) implies that if the number of partitions is the same as the number of data elements in $W_r + W_t$, then each

[2]Note that s_{G_i} may not be a real data element observed in $W_r + W_t$, and therefore, it may not qualify to be a representative sample in W'_r.

data will have itself as its representative point, and hence the distribution of the selected representative set will have the same distribution as the data set in $W_r + W_t$. However, since the number of data elements we will have in the new reference window W'_r is fixed, we will always have $k \leq |W_r| < |W_r + W_t|$. Hence, this optimal goal will not be achieved in our approach.

Our proposed solution for partitioning $W_r + W_t$ is as follows. First, we calculate the start and end values g_i^S and g_i^E for each partition G_i. Using equations (4),(5),(6), we get the following equation (intermediate steps are omitted due to lack of space):

$$(\hat{f}(g^S) + \hat{f}(g^E))(g^E - g^S) = \frac{2}{k} \int_{W_r+W_t} \hat{f}(s)ds \qquad (7)$$

The solution of equation (7) yields (g_i^S, g_i^E) pairs for all partitions G_i $(i = 1, ..., k)$.

On the second step, we select a number of representative data from each partition. The number of representative data selected from partition G_i is determined by $k_i = round(|W_r|/k)$. If $\sum_1^k k_i \neq |W_r|$, then we add or remove several data from random groups to ensure $|W'_r| = |W_r|$.

Within each partition G_i, k_i data are randomly selected with the indicator function $1_{G_i(s)}$ as:

$$1_{G_i(s)} = \begin{cases} 1 & \text{if } g_i^S \leq v(s) \leq g_i^E; \\ 0 & \text{otherwise} \end{cases}$$

The random selection strategy is done to eliminate biases introduced through our kernel density estimation and partition process. The final substream S'_r in the updated reference window W'_r is the union of all data elements selected from all groups, i.e., $S'_r = S_{G_1} \cup S_{G_2} \cup ... \cup S_{G_k}$.

The number of total partitions k is a predefined constant value for each distribution. A higher k value implies a better quality of the representative set selection by reducing $err(Q)$. On the other hand, computation cost and memory consumption will also increase with higher k. Therefore, for a "smooth" distribution, i.e., fewer high density areas, or less "bumpy" shape, k can be kept smaller, which means that $W_r + W_t$ can be partitioned into fewer groups.

3.3 Change detection

A review of change detection techniques for dynamic streams indicates that a large portion of proposed techniques are tightly associated with a mining technique. Hence, those change detection approaches cannot be applied to other mining applications, and they may not even perform well if the mining techniques are changed. We propose an online change detection technique for our DMM framework that is not restricted to a specific stream processing application. This is useful since DMM is a general-purpose method.

As mentioned in Section 3.1, every time W_t tumbles, the change detection procedure is triggered. We compare the distributions of substreams in both W_r and W_t windows. By Definition 1, we evaluate $dist(P_r, P_t)$. If $dist(P_r, P_t) > \delta$, where δ is the predefined maximum matching distance, then a distribution change is flagged. The choice of distance functions used for calculating $dist(P_r, P_t)$ will be discussed in detail in Section 4.

3.4 Distribution matching

When a new distribution is detected at time t_1, we choose a set of data elements as the sample data set representing this new distribution. For the applications with high efficiency requirement, we directly use the substream in W_t from timestamp t_1 to $t_1 + \Delta$, so that the matching can start as soon as possible. However, as discussed in Section 3.2.1, the first set of data that arrive at the beginning of a new distribution may not capture the true distribution, especially when the distribution is complicated or the distribution change is slow. Therefore, if the accuracy is more important than efficiency, we observe the new distribution for a longer time (i.e., wait until W_t tumbles several times) and refine the representative set by using the same merging technique discussed in Section 3.2.

This new distribution is then matched with a set of important historical distributions that have been preserved. The representative sets of these distributions are sequences, and thus, we use the appropriate distance measure to check their similarity. If a match is found, then the preserved mining results for the stream with the matching distribution is output as the mining results for the new distribution. The justification is that for two highly similar distributions, their mining results should present high similarity as well. This way, the data mining time is dramatically reduced without reducing the quality of mining results.

The maximum matching distance δ is important: smaller δ implies a higher accuracy of the two matching distributions, while a larger δ increases the possibility of a new distribution to match a pattern in the preserved set leading to higher efficiency, since the time for matching distributions and reusing mining results are far less than the time for re-mining the new distribution. Therefore, the question of finding a balance between accuracy and efficiency for setting δ value arises. The impact of δ is empirically studied in Section 5.5. Our proposal for selecting δ is to initially start with $\delta = 0$ and increase it until the accuracy of matching reaches the minimum accuracy requirement of the application.

3.5 Choosing important distributions

For a dynamic data stream, there could be a large number of different distributions that are observed during the lifespan of the stream. Due to limited memory, it is infeasible to record all of these distributions along with their mining results. Furthermore, maintaining a large number of distributions could increase the time it takes to match a newly detected distribution. Hence, only important distributions, i.e., the ones that have a high probability to be observed again in the future, should be archived. We use $\mathcal{P}$ to denote the set of important distributions. In this section, we discuss how to determine the important distribution set $\mathcal{P}$.

We use the following heuristic rules to determine the importance of distributions:

1. Distributions that have occurred in the stream for more times are more important than the ones that have been observed fewer times. For each archived distribution P_i, we use a counter c_i to indicate the number of times P_i has occurred. Hence, for two distributions P_i and P_j, if $c_i > c_j$, then P_i is more important than P_j.
2. The longer a distribution lasts in the stream's lifespan, the more important it is. If t_1 is the timestamp a distribution P_i is detected and t_2 is the timestamp that the subsequent change is detected, we can denote P_i's lifespan as $T_i = t_2 - t_1$. Hence, if $T_i > T_j$, then P_i is more important than P_j.
3. The more distinctive a distribution is, the higher is the chance that it will be archived. A distribution that is similar to an existing distribution in $\mathcal{P}$ (but

not enough to be recognized as a match) is regarded as less important. When searching for a match of a new distribution P_i, we record the smallest distance $min_i(dist)$ between P_i and any of the archived distributions. Hence, for two distributions P_i and P_j, if $min_i(dist) > min_j(dist)$, then P_i is more important than P_j.

4. A distribution P_i that has mining results R_i with higher accuracy is more important than a distribution with less accurate mining results. Let $Acc(R_i)$ be the accuracy of the mining results for P_i, hence if $Acc(R_i) > Acc(R_j)$, then P_i is more important than P_j[3].

When a distribution change is detected, whether or not a match is found, the important distribution set $\mathcal{P}$ is updated. The distribution P_r that was in effect when a change was detected is evaluated to determine whether it should be included in $\mathcal{P}$. If P_r had been matched with pattern $P_i \in \mathcal{P}$, then P_r replaces P_i in $\mathcal{P}$ if its lifespan is longer than P_i's lifespan (rule 2). If P_r has no matching distribution in $\mathcal{P}$ and $\mathcal{P}$ has not reached its maximum memory allowance, we add P_r to $\mathcal{P}$. Otherwise, the distribution that is the least important is pruned from $\mathcal{P}$ according to rules 1-4.

4. DISTANCE FUNCTION SELECTION

Both the change detection and distribution matching phases of DMM rely on computing the distance between distributions. A number of distance functions have been proposed in literature for both distribution matching and time series pattern matching. Examples include L_1-norm (also known as *Manhattan distance*), L_2-norm (also known as *Euclidean distance*), Dynamic Time Wrapping (DTW) [20], Longest Common Subsequence (LCSS) [22], Edit Distance on Real sequence (EDR) [7], and Relativized Discrepancy (RD) [15]. For most of these distance functions, there is no proof of lower bound accuracy, nor is there a systematic comparison among them that would suggest the appropriate distance function for a given application when distribution matching (as defined in Definition 1) is performed.

L_1-norm and L_2-norm do not have the stretching ability, i.e., they require the two matching substreams to have the same length. Therefore, they are very inflexible and cannot be applied to many stream applications such as the applications using time-based sliding window model. This is because the length of a time-based window is not a constant; thus there is no guarantee that two time-based windows will always have the same number of data items. On the other hand, L_1-norm and L_2-norm are the only two distance functions in the list that are metric, and, therefore, computationally more efficient to compute. LCSS and EDR are robust to noise, making them suitable for mining applications that are not sensitive to noise. L_1-norm and L_2-norm are the most efficient distance functions and RD is the one with highest time complexity. There have been individual studies that have analyzed the accuracy of these techniques, but the results, over a number of data sets, are inconclusive. However, general patterns indicate that L_1-norm and L_2-norm are usually the least accurate, in particular when there is noise.

A proper distance function that can be plugged into DMM should be efficient (preferably linear or quadratic complexity), with the ability of stretching (i.e., the ability of calculating the distance between two substreams with different length). The noise tolerance ability may or may not be preferable depending on the nature of the stream. For example, for ECG streams, outliers/noise are critical information that must not be ignored, whereas in temperature readings, occasional noise is usually caused by sensor malfunction, and hence should be eliminated when detecting distribution changes or performing distribution matching.

According to these criteria, DTW, LCSS, EDR and RD are all reasonable distance functions to be used in the DMM framework. To evaluate the accuracy of our technique with these distance functions plugged in, we run a set of experiments using five synthetic data streams.

Each data stream contains 1,000,000 points with only one numerical attribute for each sample element. The distribution changes occur every 10,000 points. Hence, there are 99 actual changes in each stream. The arrival speed of the stream is stable, with one tuple per unit time. This is for the purpose of gaining control over the window's length, since a time-based sliding window will be equal to a count-based one when the stream speed is stable. However, keep in mind that our technique does not require the stream to have an even speed. Stream S_1 has a normal distribution, and S_2 is a mixture of normal distribution with some uniform noise. Streams S_3, S_4, and S_5 contain exponential, binomial and poisson distributions, respectively.

We set the size of W_r to 500 data items, and the time interval Δ for W_t to 200 time units. The maximum matching distance δ is set to 5% (impact of δ is studied in Section 5.5). We analyzed all data sets manually and drew the curves of the streams using Matlab to visually observe changes. We manually determined all the changes under the given threshold setting. This determines the ground truth. A mismatch indicates that the compared substreams do not have the same distribution. The recall (R) and precision (P) of our change-detection technique using each distance function are presented in Table 2.

Table 2: Distance function comparison

Stream	DTW		LCSS		EDR		RD	
	R	P	R	P	R	P	R	P
S_1	0.82	0.52	0.86	0.47	0.71	0.89	0.76	0.87
S_2	0.87	0.49	0.89	0.51	0.67	0.89	0.73	0.91
S_3	0.79	0.55	0.83	0.53	0.66	0.87	0.71	0.93
S_4	0.88	0.50	0.84	0.46	0.72	0.83	0.77	0.86
S_4	0.83	0.56	0.87	0.57	0.65	0.82	0.69	0.87

These results indicate that DTW and LCSS have high recall but very low precision, indicating large number of false-alarms. The precision of EDR is satisfying, but the recall is slightly lower than RD. RD has the best overall performance, with the trade-off of a higher computational cost.

5. EXPERIMENTS

In this section, we present a series of experiments using both synthetic and real data streams to evaluate the proposed DMM framework. Our experiments are carried out on a PC with 3GHz Pentium 4 processor and 1GB of RAM, running Windows XP. All algorithms are implemented in C.

[3]We use the mining results of a supervised learning task as the ground truth. $Acc(R_i)$ refers to the accuracy of R_i compared with the ground truth. If R_i is later reused for another distribution P_k that matches P_i, the accuracy of R_i over P_k is still calculated w.r.t to the ground truth. Therefore, the error introduced by the adopted mining technique and the error caused by the match are both taken into account.

5.1 Change detection evaluation

We compare our DMM approach with two other generic change detection techniques: the kernel density approach (KD) [1], and another distance function-based approach (DF) [15]. The gaussian kernel function used in KD is based on the authors' recommendation [1]. Several detection strategies using different statistics are discussed in [15]. According to their experiments, the ϕ statistics proposed in their paper has the overall best performance. Thus, in our experiments, we only implement DF using ϕ statistics.

We use the same test data streams $S_1 - S_5$ as in Section 4. The parameter settings are also the same, i.e., $|W_r| = 500$ data items, $\Delta = 200$ time units, and $\delta = 5\%$. Any changes reported after 200 time units (i.e., after the time W_t tumbles), from the beginning of the change is considered late. Since DF uses RD as its distance function, we also adopt RD in our change detection technique. The recall (R) and precision (P) of the three techniques are shown in Table 3. The number of on-time (O) and delayed (D) changes detected are presented in Table 4.

Table 3: Recall and precision comparison

Stream	DMM		DF		KD	
	R	P	R	P	R	P
S_1	0.76	0.87	0.66	0.83	0.72	0.82
S_2	0.73	0.91	0.62	0.84	0.61	0.82
S_3	0.71	0.93	0.54	0.80	0.38	0.50
S_4	0.77	0.86	0.57	0.77	0.49	0.59
S_5	0.69	0.87	0.52	0.78	0.42	0.60

Table 4: On-time and delayed comparison

Stream	DMM		KD		DF	
	O	D	O	D	O	D
S_1	59	16	49	16	54	17
S_2	55	17	43	18	46	14
S_3	53	17	35	18	24	14
S_4	60	16	37	19	28	21
S_5	50	18	39	12	25	17

These results demonstrate that our proposed change detection technique outperforms both DF and KD in terms of precision and recall. Furthermore, DF and KD suffer from a drop in recall, with KD's drop more significant, when the testing stream has a non-normal distribution. The performance decrease in KD is caused by its assumption that the underlying stream should have approximately normal distribution. When this assumption does not hold, performance suffers. The slight drop in DF's recall may be due to the bad representative set selection for complicated distributions.

5.2 Distribution drifts vs. shifts

Distribution drifts refer to the slow and gradual changes in the stream, while distribution shifts refer to significant sudden changes. To study the performance of the DMM, DF, and KD change detection approaches over shifts and drifts, we modify streams S_1 and S_3 to generate four new streams. Streams S_6 and S_7 are the ones with normal distribution, while streams S_8 and S_9 follow exponential distribution. In S_6 and S_8 there are only distribution drifts, while in S_7 and S_9, only distribution shifts occur. The parameter settings are the same as those reported in the previous subsection. Table 5 displays the recall and precision of the three techniques over these four streams, and Table 6 shows the number of on-time and delayed detection.

These results suggest that, in general, distribution shifts are easier to detect than distribution drifts, because the changes are more significant. Our approach greatly outperforms KD and DF for detecting drifts. This is because, the process of kernel density estimation in KD and representative set selection in DF are performed immediately after a distribution change is detected. In the case of drifts that are slow by definition, a new distribution will take a long time to stabilize, and, thus, kernel density estimation and representative set used in KD and DF are highly unreliable. On the other hand, DMM keeps updating the representative set, and, hence, the next time a distribution change occurs, the reference window contains a substream that can accurately represent the current distribution for comparison.

Table 5: Recall and precision comparison for distribution drifts and shifts

Stream	DMM		DF		KD	
	R	P	R	P	R	P
S_6	0.67	0.85	0.53	0.67	0.56	0.80
S_7	0.77	0.93	0.69	0.94	0.73	0.92
S_8	0.65	0.86	0.42	0.71	0.31	0.71
S_9	0.70	0.88	0.55	0.81	0.51	0.62

Table 6: On-time and delayed comparison for distribution drifts and shifts

Stream	DMM		KD		DF	
	O	D	O	D	O	D
S_6	54	12	33	19	38	17
S_7	69	7	62	6	63	9
S_8	55	9	29	13	20	11
S_9	57	12	40	14	33	16

5.3 Distribution matching evaluation

To evaluate the performance of our distribution matching approach, we carried out a series of experiments on a real data set that exhibits periodically changing distributions. The data set we use is copyrighted data from the Tropical Atmosphere Ocean (TAO) project. These streams record the sea surface temperature from past decades. Detailed information about this project and the real-time data sets can be found in [17]. The distribution of this stream demonstrates both shifts and drifts. The data set contains 12,218 streams each with a length of 962. Each stream contains readings over different time period. Since these streams are too short, we sorted them by the time period these streams are recorded, and concatenate them to obtain one stream with 11,753,716 data points continuous over years. We analyzed Tao manually and drew the curves of the streams using Matlab to visually observe changes. There are a total of 3244 distribution changes in this stream. This determines the ground truth.

The arrival rate is set to be one tuple per unit time. Since TAO stream consists of some rapid distribution shifts, the efficiency requirement is high. Hence, we set the reference set size $|W_r|$ to 300 data items, and tumbling time interval Δ to 100 time units. Since this stream only contains numerical values, assuming each data element takes 1 byte, only 300 bytes are needed for archiving each distribution pattern. Even if there is no periodic distribution in the TAO stream, it only takes less than 1MB memory to store all the distributions. However, considering that the mining results will need to be stored along with each distribution, we limit the important distribution list $\mathcal{P}$ to 100 distributions.

The results of change detection are shown in Table 7, and the results for periodic distribution matching are shown in Table 8.

These results demonstrate that periodically changing dis-

Table 7: Change detection in TAO stream

Recall	Precision	On-time	Delay
0.72	0.89	1927	418

Table 8: Distribution matching in TAO stream

Total # of periodic distributions	Total matchings found	Avg. matchings per distribution
128	477	3.72

tributions do exist in real-world data streams, and our DMM of distribution change detection and matching performs well on real data sets.

5.4 Efficiency with and without DMM

To demonstrate that the proposed DMM framework can increase the overall efficiency for mining periodically changing streams, we apply a stream mining application on the TAO stream introduced in the previous section.

For a temperature monitoring stream, a popular mining task is to cluster the readings for further analysis and comparison. Hence, we adopt a popular decision tree-based clustering technique, VFDT [8], to cluster the temperature readings in TAO stream. VFDT is recognized as one of the best decision tree generation technique for dynamic data streams. Many other decision tree techniques cannot deal with the ever-changing distributions and cannot handle continuous numerical values. Every time a distribution changes, the existing decision tree may not be suitable for the new distribution in terms of both accuracy and efficiency. Hence, different decision trees are required for clustering data sets with different distributions. Rebuilding a decision tree can be time consuming. Many newly arrived data elements will be missed during the reconstruction process. However, if the decision trees for important distributions (i.e., distributions that may occur periodically) can be stored and reused, then the overall execution time for clustering TAO stream can be greatly reduced.

Table 9 verifies this insight. DMM reduces total execution time by 31.3% (including the time for updating reference window, change-detection, matching and mining).

Table 9: Run-time with and without DMM

Total time without DMM	Total time with DMM	Avg. time without DMM	Avg. time with DMM
1121927	771235	346	237

We did not conduct experiments to test the accuracy of the reused mining results. This is because the accuracy of the reused mining results is solely determined by the accuracy of the mining technique and the accuracy of our matching approach. Any existing mining technique can be plugged into our framework. Therefore, since our matching technique demonstrates a promising accuracy rate, the accuracy of the reused mining results should be close to the accuracy of the adopted mining technique. Testing the accuracy of reuse is not the major point of the paper; the development of the framework and the techniques that enable the reuse of (any) mining technique is the main focus.

5.5 Effect of maximum matching distance δ

As discussed in Section 3.4, the choice of the maximum matching distance δ can affect both the accuracy and efficiency of DMM. To study the effect of δ, we repeat the experiments of DMM on all data sets we introduced in previous sections using different δ settings. The empirical results are shown in Tables 10 – 13.

Table 10: Change detection in S_1 to S_5

Stream	$\delta = 15\%$		$\delta = 10\%$		$\delta = 5\%$		$\delta = 2\%$	
	R	P	R	P	R	P	R	P
S_1	0.63	1	0.71	0.92	0.76	0.87	0.86	0.74
S_2	0.60	1	0.68	0.96	0.73	0.91	0.93	0.79
S_3	0.57	0.99	0.69	0.92	0.71	0.93	0.88	0.84
S_4	0.54	1	0.71	0.94	0.77	0.86	0.95	0.81
S_5	0.55	1	0.70	0.97	0.69	0.87	0.86	0.77

Table 11: Change detection in S_6 to S_9

Stream	$\delta = 15\%$		$\delta = 10\%$		$\delta = 5\%$		$\delta = 2\%$	
	R	P	R	P	R	P	R	P
S_6	0.57	1	0.56	0.94	0.67	0.85	0.71	0.85
S_7	0.54	1	0.63	0.93	0.77	0.93	0.75	0.81
S_8	0.48	0.98	0.55	0.91	0.65	0.86	0.78	0.74
S_9	0.51	0.99	0.56	0.93	0.70	0.88	0.74	0.77

These results indicate that with a larger δ value, fewer distribution changes will be detected. Two distributions with 85% similarity will be considered the same with $\delta = 15\%$, whereas a distribution change will be detected if two distributions have less than 98% similarity with $\delta = 2\%$. Hence, fewer distribution changes exist in a stream with larger δ settings. More distribution matches are reported with a larger δ value, which may lead to a reduction of the overall mining time, since archived mining results can be used more frequently. However, the quality of the mining results may drop with increasing δ values, since the matches are less accurate.

Table 12: Change detection in TAO

Stream	$\delta = 15\%$		$\delta = 10\%$		$\delta = 5\%$		$\delta = 2\%$	
	R	P	R	P	R	P	R	P
TAO	0.61	1	0.66	0.97	0.72	0.89	0.88	0.75

6. RELATED WORK

Detecting changes in data streams and adjusting stream mining models accordingly is a challenging issue and has only been recognized as an important one in the past few years. A number of techniques have been proposed for solving these problem [6, 12, 14, 23], but most of them are ad-hoc and cannot be generalized to different mining applications. These change detection technique usually only perform well on certain types of streams.

Aggarwal addressed the data stream change detection problem by providing a framework that uses velocity density estimation [1]. By estimating the rate at which the changes in the data density occur, the user will be able to analyze the changes in data over different time horizons. However, as mentioned in Section 3.2, without any pre-knowledge of the stream, density estimation can be highly inaccurate, leading to a large number of false and missed changes.

Kifer et al. present another approach to change detection that is not restricted to certain stream mining approaches [15]. Their idea is similar to ours in that two sliding windows are used over a data stream to capture the previous and current distributions. The distribution change is detected by calculating the distances between these two distributions using RD distance function. However, in their approach, a fixed window is used to maintain a reference to the original distribution (named 'baseline window'). The window contains the first p elements of the stream that occurred immediately after the last detected change. As mentioned in Section 3.2, the first small 'chunk' of data observed after the change point is usually not representative to the true distribution, especially when the distribution change is slow.

Similarity matching over distributions is a novel idea that has not been proposed previously. However, in time se-

Table 13: Distribution matching in TAO

δ value	Total # of periodic distr.	Total matchings found	Avg. matchings per distribution
15%	158	669	4.24
10%	136	541	3.98
5%	128	477	3.72
2%	76	208	2.74

ries analysis, similarity pattern matching is well studied. Time sequences can be regarded as simplified data streams, where each data s in stream S usually only contains a single value/field. Hence, some of the strategies for subsequence matching in time series [3, 9, 13, 16, 26] could contribute to distribution matching in data streams.

Ge and Zdonik proposed a technique for handling uncertain data in array databases using a similar idea of partitioning distribution [11]. Although a part of their idea is similar to ours, the focus of [11] is entirely different. They consider the already partitioned distribution as an available input. How to obtain the distribution of a large data set or how to partition efficiently is not discussed. These problems are not trivial and no existing work can be adopted directly. By contrast, our input is the raw data observed from a continuously arriving stream, and hence the problem is more complicated.

7. CONCLUSIONS

In this paper, we propose a framework, called DMM, for mining data streams with periodically changing distributions. In this framework, we incorporate a novel method for matching new distribution with historical patterns and reuse existing mining results. DMM is flexible, and can be applied to many types of data streams and mining applications. It is shown in the experiments that DMM can greatly reduce the overall mining time over dynamic streams with periodic distribution changes.

In order to detect and match distributions with high accuracy and efficiency, two tumbling windows are applied that contain two substreams representing the current and new distributions of the stream. An intelligent merge-and-select sampling approach is proposed that can dynamically update the reference window. The representative set generated by this approach is highly representative of the true distribution. Thus, the accuracy of our change detection and matching process is greatly improved.

Our work along these lines continue in a number of directions. One issue we are investigating is a method to automatically determine the number of partitions k for dynamic reference window selection rather than fixing it a priori. Another issue is automatically determining threshold δ and the size of window W_r. Finally, it may be interesting to design specific distance functions for matching distributions in special types of streams.

8. REFERENCES

[1] C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[2] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *Proc. 29th Int. Conf. on Very Large Data Bases*, 2003.

[3] R. Aggarwal, C. Faloutsos, and A. Swami. Efficient similarity searvh in sequence databases. 1993.

[4] B. Babock, S. Babu, M. Datar, R. Motiwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pages 1–16, 2002.

[5] H. Benoit-Cattin, B. Belaroussi, F. Bellet, and C. Odet. Simri: A versatile and interactive 3d mri simulator. *http://www-creatis.insa-lyon.fr/menu/ivolumique /segmentation/simri-hbc/*.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. Int. Colloquium on Automata, Languages, and Programming*, pages 693–703, 2002.

[7] L. Chen, M. Ozsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2005.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2000.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1994.

[10] L. Gao and X. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2002.

[11] T. Ge and S. Zdonik. Handling uncertain data in array database systems. In *Proc. 24th Int. Conf. on Data Engineering*, pages 1140–1149, 2008.

[12] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams and multiple time granularities. In *Proc. NSF Workshop on Next Generation Data Mining NGDM*, 2003.

[13] Guralnik and Srivastava. Event detection from time series data. In *Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 1999.

[14] R. Jin and G. Aggrawal. Efficient decision tree constructions on streaming data. In *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 571–576, 2003.

[15] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 180–191, 2004.

[16] X. Lian, L. Chen, J. Yu, G. Wang, and G. Yu. Similarity match over high speed time-series streams. In *Proc. 23rd Int. Conf. on Data Engineering*, 2007.

[17] Pacific Marine Environmental Laboratory. Tropical automosphere ocean project. *http://www.pmel.noaa.gov/tao/*.

[18] S. Muthukrishnan, R. Shah, and J. Vitter. Mining deviants in time series data streams. In *Proc. Int. Conf. on Scientific and Statistical Database Management*, 2004.

[19] E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, (3):1065–1076, 1962.

[20] S. Salvador and P. Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *Proc. KDD Workshop on Mining Temporal and Sequential Data*, 2004.

[21] B. Silverman. *Density Estimation*. Chapman & Hall, 1986.

[22] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. on Data Engineering*, 2002.

[23] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[24] H. Wu, B. Salzberg, and D. Zhang. Online event-driven subsequence matching over financial data streams. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2004.

[25] X. Wu. A bayesian discretizer for real-valued attributes. *The Computer Journal*, 1996.

[26] Zhu and Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 358–369, 2002.