On Segment-Based Stream Modeling and its Applications*

Charu C. Aggarwal†

Abstract

The primary constraint in the effective mining of data streams is the large volume of data which must be processed in real time. In many cases, it is desirable to store a summary of the data stream segments in order to perform data mining tasks. Since density estimation provides a comprehensive overview of the probabilistic data distribution of a stream segment, it is a natural choice for this purpose. A direct use of density distributions can however turn out to be an inefficient storage and processing mechanism in practice. In this paper, we introduce the concept of *cluster histograms*, which provides an efficient way to estimate and summarize the most important data distribution profiles over different stream segments. These profiles can be constructed in a supervised or unsupervised way depending upon the nature of the underlying application. The profiles can also be used for change detection, anomaly detection, segmental nearest neighbor search, or supervised stream segment classification. The flexibility of the tasks which can be performed from the cluster histogram framework follows from its generality in storing the *historical density profile* of the data stream. As a result, this method provides a holistic framework for density based mining of data streams. We discuss and test the application of the cluster histogram framework to a variety of interesting data mining applications such as speaker recognition and intrusion detection.

1 Introduction

A number of stream mining problems have been researched extensively in recent years [1, 2, 3, 4, 5, 6, 8] because of advances in hardware technology which have made such data feasible. While data streams may contain insightful information for a variety of applications, it is often difficult to decipher it in real time because of the speed of the data stream. In many cases, the insights are not available from individual data records, but in particular segments of the data stream. The use of stream summarization of such segments can help leverage the underlying information for mining purposes. A stream summary must be representable in (an efficient) compact form and yet it must be detailed enough so as to provide the ability to re-construct the original data distribution. As long as such a re-construction is possible, the representation can be utilized in a fairly general and holistic way for a variety of data mining problems. We will propose a statistical characterization of sliding windows over the stream segments called *cluster histograms*. We will show that this representation is a surrogate for storing the most relevant density profiles of different segments of the data stream. Since density estimation provides a comprehensive overview of the probabilistic data distribution, it implies that cluster histograms can be leveraged for a variety of very detailed data mining tasks.

In many real data sets, the data space is only sparse populated because of the skew in the underlying data. In such cases, it is space and time efficient to summarize the density distribution only at the dense regions in the data. While density based methods [10] are often used to construct clusters, the reverse technique of constructing density estimates from clusters is rarely used in real applications. The latter is possible only under the implicit assumption that a large number of data points are available for a fine-grained clustering process. In this respect, data streams provide the ideal test-bed for such an approach. While the stream clustering problem has been discussed in [2], our focus in this paper is the construction of histogram based summaries and their application in conjunction with cluster generation. This provides a comprehensive framework for summarization based mining of data streams. For example, the framework can be used to monitor the stream continuously to find when a significant statistical change has occurred, find an appropriate classification of unlabeled stream segments, anomalous stream segments, nearest stream segments, or frequently occurring temporal data

*Research was sponsored by the US Army Research laboratory and the UK ministry of Defense under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies of the US Government, the US Army Research Laboratory, the UK Ministry of Defense, or the UK Government. The US and UK governments are authorized to reproduce and distribute reprints for Government purposes.

†IBM T. J. Watson Research Center, charu@us.ibm.com

distributions. We note that the above-mentioned tasks are only representatives of solvable applications. In general, the detailed density-based summarization of the cluster histogram approach ensures that we can eventually extend this method to a much wider horizon of data mining tasks.

This paper is organized as follows. In section 2, we will discuss the basic concepts underlying the cluster histogram framework. In section 3, we will discuss how the cluster histograms are maintained dynamically. In section 4, we will illustrate how to use the cluster histogram approach for a number of unsupervised applications. In section 5, we will show how to use the technique for classification in the presence of class labels. The empirical results are presented in section 6. Section 7 contains the conclusions and summary.

2 The Cluster Histogram Framework

The most direct way of storing the data distribution is that of kernel density estimation [9]. While density estimation has been adapted to large databases [12], its storage and use for real-time stream mining environments presents a number of different challenges. This is because of the inherent inefficiency of kernel density estimation which estimates the density distributions over all regions of the data including the (majority of) sparse ones. This can rapidly become untenable for non-uniform data distributions of even modestly high dimensionality. In many practical applications, most of the regions in the data are empty, and it is inefficient to maintain the density distributions at every point. At the same time, the dense and sparse regions can rapidly change in an evolving stream. Furthermore, our aim in the paper is to use the stream segment characterization not just for change detection, but also for a variety of other tasks such as segment classification and anomaly detection. In order to achieve this goal for fast data streams, it is important to be able to efficiently update and store the corresponding data distributions. To this effect, the cluster histogram framework turns out to be a practical, efficient and scalable approach. In order to construct cluster histograms, we build upon some machinery for micro-clustering which was developed in [2]. We assume that the dimensionality of the data stream is d. The definition of a micro-cluster is as follows:

DEFINITION 1. *A micro-cluster for a set of d-dimensional points $\mathcal{C} = \{X_{i_1} \ldots X_{i_n}\}$ with time stamps $T_{i_1} \ldots T_{i_n}$ is defined as the $(2 \cdot d + 2)$ tuple $(\overline{CF2^x}(\mathcal{C}), \overline{CF1^x}(\mathcal{C}), CF^t(\mathcal{C}), n(\mathcal{C}))$, wherein $\overline{CF2^x}(\mathcal{C})$ and $\overline{CF1^x}(\mathcal{C})$ each correspond to a vector of d entries. The definition of each of these entries is as follows:*

- *For each dimension, the sum of the squares of the data values is maintained in $\overline{CF2^x}(\mathcal{C})$. Thus, $\overline{CF2^x}(\mathcal{C})$ contains d values. The p-th entry of $\overline{CF2^x}(\mathcal{C})$ is equal to $\sum_{j=1}^{n} (x_{i_j}^p)^2$.*
- *For each dimension, the sum of the data values is maintained in $\overline{CF1^x}(\mathcal{C})$. The p-th entry of $\overline{CF1^x}(\mathcal{C})$ is equal to $\sum_{j=1}^{n} x_{i_j}^p$.*
- *The last update time (which is $max\{T_{i_1} \ldots T_{i_n}\}$) is maintained in $\overline{CF}^t(\mathcal{C})$.*
- *The number of data points is maintained in $n(\mathcal{C})$.*

Next, we will define the cluster histogram, a statistical construct used to store summary information about the micro-clusters. The cluster histogram is defined over a set of clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ and window W. We note that the window size W may be expressed either terms of the time interval, or in terms of the number of data points. For the purpose of this paper, we define it over the number of data points. We assume that the data point which arrived in the window W belongs to one of the clusters denoted by $\mathcal{C}_1 \ldots \mathcal{C}_k$. Let $\mathcal{D}(W)$ denote the data in the past window W of the data stream. Then, the cluster histogram is defined as follows:

DEFINITION 2. *The cluster histogram $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$ for a set of k clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ and window of time W is defined as the k-dimensional vector $(f_1 \ldots f_k)$, where f_i is defined as follows:*

$$f_i = |\mathcal{D}(W) \cap \mathcal{C}_i| / |\mathcal{D}(W)| \tag{2.1}$$

Thus, the cluster histogram defines the relative frequencies of the data points in the clusters. We note that for a sufficiently high granularity of the clustering process, cluster histograms provide a good estimate of the density distribution in the data. In fact, cluster histograms can be used to estimate the density at a given data point. Let $\overline{X}(\mathcal{C}_i)$ and $h(\mathcal{C}_i)^2$ be the centroid and variance of cluster $\mathcal{C}_i$. Then, the density estimate $\eta(x)$ at any point x in the space can be constructed as follows:

$$\eta(x) = \sum_{i=1}^{k} \frac{f_i}{\sqrt{2 \cdot \pi} \cdot (h(\mathcal{C}_i) + h')} \cdot e^{-\frac{|(x - \overline{X(\mathcal{C}_i))^2}|}{2(\cdot h(\mathcal{C}_i)^2 + h'^2)}} \tag{2.2}$$

Here h' is an additive bandwidth which is defined in the same way as standard kernel density estimation, except that we use the number of clusters rather than the number of points to define h'. Specifically, the exact value of h' is defined by the Silverman's approximation rule [9]. According to this rule, for a data distribution with n points (clusters) and variance σ of the data points (cluster centroids), the value of h' is chosen to be $1.06 \cdot \sigma \cdot n^{-1/5}$. We note that $\lim_{n \to \infty} h' \to 0$. In effect, the density estimate is equal to the sum of Gaussian kernels which are centered at different clusters, and

have bandwidth which is defined by the variance of the data points in the cluster. We note that the relation defined in Equation 2.2 is similar to that of kernel density estimation, except that we use clusters rather than individual data points. For very fine grained clusters, such an approach provides a very good estimate to the process of kernel density estimation. This can be formally understood from the following observation:

LEMMA 2.1. *In the limiting case, when each cluster contains only one data point, the value $\eta(x)$ is equal to the standard kernel density estimate.*

Proof: The proof of the above lemma follows from the fact that the value of $h(\mathcal{C}_i)$ is zero for each cluster $\mathcal{C}_i$. On substituting the value of $h(\mathcal{C}_i) = 0$ in Equation 2.2, we obtain the same formula [9] used for standard kernel density estimation.

The above result also implies that clusters with low spatial variance (low value of $h(\mathcal{C}_i)$) provide an estimate which is closer to the true density estimate. Therefore, the use of fine grained micro-clusters is able to provide estimates which are quite close to the true estimate. On the other hand, for cluster histograms to be useful for summarization, each cluster must contain a statistically significant number of data points. In general, we would like clusters with low spatial variance but many data points. This is only possible in a massive data stream with a sufficiently large number of data points in each segment. Thus cluster histograms are specially suited to their use in a data stream environment. While the relationship between data density and cluster histograms is interesting from a conceptual point of view, it is more useful to work directly with cluster histograms, since they can be updated efficiently, and used for a variety of data mining tasks.

We note that the difference in probability density between two distributions $\eta_1(\cdot)$ and $\eta_2(\cdot)$ may be defined as follows:

$$Diff(\eta_1, \eta_2) = \int_x |(\eta_1(x) - \eta_2(x)| \mathrm{d}x \tag{2.3}$$

Since the area under each density distribution is one unit, the value of the above expression lies in the range $(0, 2)$. The discrete analogue of this difference in probability densities is defined by the difference in two cluster histograms. We define the distance between two cluster histograms as follows.

DEFINITION 3. *The distance $Dist(H_1(\cdot), H_2(\cdot))$ between two histograms $H_1(\cdot) = (f_1 \ldots f_k)$ and $H_2(\cdot) = (f'_1 \ldots f'_k)$ is defined as follows:*

$$Dist(H_1(\cdot), H_2(\cdot)) = \sum_{i=1}^{k} |f_i - f'_i| \tag{2.4}$$

Algorithm *ConstructHistogramProfiles*(HistWindow: W, Profile Threshold: δ, NumberOfClusters: k);
begin
Initialize $\{\mathcal{C}_1 \ldots \mathcal{C}_k\}$ using a k-means approach
on $InitNumber$ data points;
Initialize profile set $\mathcal{H}$ and active profile set $\mathcal{H}_{active}$ to {};
Initialize $\mathcal{CH}$ with k entries containing current centroids
along with indices, and current time stamp;
for the next data point $\overline{X}$ in the data stream **do**
begin
Determine closest micro-cluster $\mathcal{C}_i$ to $\overline{X}$;
Compute closest distance Δ from $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$
to any histogram in $\mathcal{H}_{active}$;
{ Denote corresponding histogram by $H_{closest}$; }
if $\Delta > \delta$ add $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$ to $\mathcal{H}$ and $\mathcal{H}_{active}$;
else increment the frequency count
of $H_{closest}$ by 1;
Store start/end pt. of cur. histogram in profile database;
Perform shift test on each centroid;
for each shifted centroid add it along with
corresponding time stamp to centroid history $\mathcal{CH}$;
Remove any histogram in $\mathcal{H}_{active}$
which contains at least $1/k$ fraction
of the points in shifted clusters;
end
end

Figure 1: Constructing Histogram Profiles

Thus, the distance between two cluster histograms is equal to the Manhattan distance between them. We note that since the area under each histogram is at most one unit, the value of the above distance always lies in the range $(0, 2)$. This value serves as an approximation to the difference between the probability densities of two stream segments for very fined grained clusters.

3 Generating Histogram Profiles

The histogram profile generation process maintains a profile database of the different density profiles of the incoming data stream. The construction of the profile database requires us to simultaneously maintain shifting micro-cluster centroids together with their corresponding histogram profiles. This database of profiles is denoted by $\mathcal{H} = H_1 \ldots H_p$. The maintenance of such a database is a challenging task because some of the centroids may shift significantly over time because of the evolution of the data stream. As a result, some histogram profiles may also become stale, if measured with respect to the current set of centroids. We note that we do not wish to update and store away the centroids at each iteration, since this can be very costly from a computational standpoint. Rather, we would like to maintain an active centroid history $\mathcal{CH}$ which is updated only when the clusters have shifted significantly. This active centroid history is useful for tracking the clus-

ter behavior of previous histograms in the data and is maintained on disk. Each entry in $\mathcal{CH}$ contains the corresponding cluster centroid, its index (which can vary from one through the number of clusters), and the time stamp at which it was last updated. We also maintain an *active database* of profiles $\mathcal{H}_{active} \subseteq \mathcal{H}$. This active set corresponds to those histograms for which the current cluster centroids have not shifted significantly enough from their last update time in order to be considered different clusters. A histogram is deleted from the *active profile* database, if the shifted clusters contain a significant percentage of the data points in it. The histogram maintenance algorithm utilizes the following input parameters for the cluster generation process:
(1) The number of micro-clusters k which are stored by the algorithm. In general, a larger value of k is more desirable, since it provides a greater level of accuracy in histogram estimation. The value of k is defined by the main memory availability and computational limitations. **(2)** The minimum distance δ between two histogram profiles for them to be considered sufficiently different to be stored separately in the active database of profiles. Thus, for any pair of histogram profiles $H_i, H_j \in \mathcal{H}_{active}$, we must have $Dist(H_i, H_j) \geq \delta$. **(3)** The window size W used for the histogram profile generation process. This is an application dependent parameter which defines the horizon used for the data mining process. For the purpose of this description, we will describe the window size W in terms of the number of data points.

The process of histogram construction is illustrated in the pseudo-code of Figure 1. The first step is the initialization process in which we use a static k-means algorithm to construct the initial set of micro-clusters. In order to do so, we collect the first $InitNumber$ data points and apply the traditional k-means approach on it. This ensures that the initial set of micro-clusters are sufficiently robust for an effective clustering process. The overall process of histogram profile generation constructs micro-clusters and histograms simultaneously, while generating and deleting the profiles along with the process of updating micro-cluster statistics. The profile databases $\mathcal{H}$ and $\mathcal{H}_{active}$ contain a set of histograms along with the following information for each histogram:
(1) The frequency counts of each micro-cluster. (2) We maintain the starting and end points of the histogram segments in the data stream. In some applications, it may be desirable to know the position of that segment during the data stream arrival process. (3) We maintain a count of the number of times that a histogram profile is encountered. This can be used for a variety of tasks such as finding the frequent histogram profiles in the data stream. (4) In addition, we maintain a centroid history $\mathcal{CH}$ on disk which is useful for tracking the shifting centroids of the cluster histograms. This information is necessary to check whether some centroids have drifted significantly enough for a histogram to be considered stale. In such cases, the stale histograms are no longer useful for direct similarity comparision with current histograms, but are useful from a historical and mining standpoint.

The micro-cluster update process works with a nearest neighbor approach in which incoming data points are added to their closest micro-cluster. The corresponding micro-cluster statistics are updated whenever a data point is added to a new cluster. The next step is to construct a histogram from the previous window of length W. For this purpose, we maintain the history of assignments of points to micro-clusters *only* for the past window of length W. This is used to compute the number of points in each micro-cluster. Once the histogram H is constructed, we determine if it is sufficiently distinct from the current active database of profiles in order for it to be inserted as a distinct histogram in the profile database. If this minimum distance of H to any histogram in active database is larger than the threshold δ, then we add the histogram H to the profile database.

The centroids of the micro-clusters may shift over time because of changes in the data distribution. In some cases, this shift may be significant enough that the corresponding histogram profiles may no longer remain relevant. In such cases, it is desirable to delete those profiles from the histogram database $\mathcal{H}$. Therefore, we design a *shift test* for each micro-cluster in the profile database. This shift test is implemented by finding the number of standard deviations by which the centroid of a micro-cluster is distant from the corresponding centroid associated with the histogram at the time that it was added to the profile database. Let this number of standard deviations be denoted by s_i for the micro-cluster i. If the value of s_i is greater than a threshold[1] t for any micro-cluster i, then we need to update its centroid in the corresponding centroid history $\mathcal{CH}$. Those histograms which contain a significant percentage of the data points from shifted micro-clusters may be considered stale. Specifically, the fraction of data points from shifted micro-clusters must be larger than the fraction $1/k$ of the total points in that histogram. The reason for using this approach is that many micro-clusters may not contain a significant number of data points for a particular histogram. Stale

[1] We used a threshold $t = 1$, which corresponds to the fact that a centroid was never allowed to deviate more than the radius of that cluster.

histograms can be removed from the *active profile database* $\mathcal{H}_{active}$. However, these histograms are still maintained in the profile database $\mathcal{H}$, since they can be used in conjunction with the centroid history $\mathcal{CH}$ in order to perform density characterization.

The overall process of histogram construction and maintenance is illustrated in Figure 1. We note that in the pseudo-code of Figure 1, we have performed the insertion and deletion step whenever a new stream data point arrives. In practice, it is not necessary to do this. This is because the histograms and centroids do not change significantly upon the addition of a single data point. Therefore, we batch the insertion and deletion steps over the user specified windows of length W. This ensures that successive histograms do not have significant overlap in data points.

4 Unsupervised Applications

In this section, we discuss some unsupervised applications of cluster histograms, such as change detection, outlier detection, and nearest neighbor search.

4.1 Change Detection In the change detection problem, we would like to find sudden changes in the data stream. In [3], these changes have been defined in terms of changes in the underlying data distribution. However, as discussed in [3], this method becomes rapidly impractical for higher dimensionalities because of an exponentially large number of grid points at which the density needs to be estimated. Therefore, the (real-time component of the) method discussed in [3] is only practical for 2- to 4-dimensional projections of the data. On the other hand, we will test our application on data sets with dimensionality greater than 30. Since cluster histograms encode an indirect representation of the data distribution, they can also be used for change detection over successive windows of the data stream. Let the base histogram in the immediately preceding window be denoted by H_1. We would like to determine whether the histogram H_2 in the current window has significantly changed from H_1. One way of determining whether a sufficiently high level of change has occurred is to calculate the distance $Dist(H_1, H_2)$ between the histograms H_1 and H_2. This is a simple and useful definition for many applications because it approximates the difference in density distribution between the two windows. If the distance is larger than a user-defined threshold δ_t, then it is assumed that a change has indeed taken place.

While the above definition is useful because of its simplicity, it is often more interesting to find the change quantification directly in terms of the *probability* that that the change has occurred for reasons other than chance. Let the number of data points in each of the k clusters for the histogram H_1 be denoted by $n_1^1 \ldots n_k^1$, and the corresponding number of data points for histogram H_2 be denoted by $n_1^2 \ldots n_k^2$. We would like to determine whether the changes from histogram H_1 to H_2 have occurred by chance. Since the distribution of H_1 is used as the basis upon which the change detection process is executed, the process is sensitive to the number of data points in H_2, but not the number of data points in H_1. The larger the number of data points in H_2, the smaller the change required for it to be considered statistically significant. We note that the bin i of histogram H_1 contains a fraction $f_i^1 = n_i^1 / \sum_{i=1}^k n_i^1$ of the data points in it. The same quantity for histogram H_2 is denoted by f_i^2. We would like to test if the data points from H_2 could be considered to have been drawn from the distribution determined by H_1. therefore, we utilize a hypothesis test method. The null hypothesis is assumed to be the case that the data points from H_2 have drawn from the distribution determined by H_1. We note that the expected fraction of data points belonging to bin i (based on the correctness of the null hypothesis) is equal to f_i^1. Thus, the null hypothesis assumes that the count in bin i for histogram H_2 is equal to the sum of $\sum_{i=1}^k n_i^2$ i.i.d. random binary variables, each of which takes on the value of 1 with probability $f_i^1 = n_i^1 / \sum_{i=1}^k n_i^1$. The standard deviation of the average of these binary variables can be estimated to be[2] $\sigma_i = \sqrt{f_i^1 \cdot (1 - f_i^1)/n_i^2}$. Then, we determine the number of standard deviations by which the fraction f_i^2 is different from f_i^1. The corresponding z-number for bin i is defined as follows:

$$z(i) = ||f_i^2 - f_i^1||/\sigma_i \tag{4.5}$$

The value of $z(i)$ determines that the frequency in bin i for histogram H_2 differs from that of the basis histogram H_1 by chance. We note that it is not easy to combine the results from different bins, since the counts in the different bins are not independent from one another. Therefore, we compute the fraction of data points which belong to a bin i for which the corresponding frequencies were different at a probability level of significance[3] of 99.99% under the normal distribution assumption. This fraction is equal to the value of the change quantification, and directly represents the fraction of data points in the window whose distribution has changed for reasons other than chance.

[2] This is because the standard deviation of each variable is equal to $\sqrt{f_i^1 \cdot (1 - f_i^1)}$, and the corresponding average is estimated to be a normal distribution which scales inversely with the square-root of the number of data points in histogram H_2.

[3] This is achieved by finding the fraction of points for which each bin i satisfied $z(i) \geq 3$.

4.2 Outlier Detection and NN Search The above mentioned technique can be used for anomaly detection and nearest neighbor search as well. While outlier detection and nearest neighbor search are usually studied in terms of individual data points, we study them in terms of *stream segments*. This is because many anomalous events occurring in the stream may not show up in individual data points, but may be buried in the aggregate distributions of individual stream segments. A stream segment is defined to be a outlier or anomaly when its distribution is significantly different from the earlier segments that have been encountered. Since cluster histograms encode information about the data distribution, they can be used for anomaly detection by matching whether the profile of the current stream segment matches any of the previous stream segments in the histogram profile $\mathcal{H}$. For each histogram $H_i \in \mathcal{H}$, we calculate the distance of the histogram of the current stream segment with that each of the past segments $H_i \in \mathcal{H}$. We note that in order to match the current histogram with a histogram from the profile database, it is necessary to construct it with respect to the same set of clusters as available from the cluster history $\mathcal{CH}$. Therefore, for the current target, we find the last p changes in the centroid history. In each such period j, we compute the histogram HP_j for the current target window using the corresponding combination of centroids. Similarly, we determine the active histograms during each of the last p period of centroid change. Specifically, let the active histogram profile database during the jth period be denoted by $\mathcal{HA}_j$. A match is said to exist when the distance of HP_j to the segment $H_i \in \mathcal{HA}_j$ is less than a user-defined threshold δ. If such a match does not exist, then the current stream segment can be assumed to be an anomalous segment for the data stream. It is easy to see that this technique can also be used directly for nearest neighbor search. In fact, this approach is an extension of the nearest neighbor method for outlier detection [7].

The cluster histogram method can also be used for frequent segment recognition. This is because we store the frequency counts of the histograms in the profile database. Such histograms represent the frequently occurring distributions in the data stream. Frequent histograms are also useful for a wide variety of tasks such as the characterization of distinct segments in the data stream. For example, in the empirical section, we will illustrate a VOIP application in which cluster histograms encode the distinct density distributions of different speakers from a given data set. We will see that these frequent segments can often encode *signature characteristics* of application-specific behavior.

Algorithm *ConstructSupervisedProfiles*(Hist. Window: W, Profile Threshold: δ, NumberOfClusters: k);
begin
Initialize $\{\mathcal{C}_1 \ldots \mathcal{C}_k\}$ using a k-means approach on $InitNumber$ data points;
Initialize profile set $\mathcal{H}(L_i)$ for each class L_i to $\{\}$;
Compute micro-cluster statistics for each initial cluster;
for the next point $\overline{X}$ in the data stream **do**
  begin
  Determine closest micro-cluster $\mathcal{C}_i$ to $\overline{X}$;
  Add $\overline{X}$ to $\mathcal{C}_i$ and update micro-cluster statistics;
  Compute histogram profile $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$;
  Determine if previous window S of length W is associated with a particular class label denoted by $l \in \mathcal{L}$;
  if associated class label l exists then **begin**
    Compute closest distance Δ from $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$ to any histogram in $\mathcal{H}(l)$;
    if $\Delta > \delta$ add $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$ to $\mathcal{H}(l)$;
  end
  Perform shift test on each centroid for each histogram in $\mathcal{H}$ and remove histograms which is invalidated by the shift test;
  end
end

Figure 2: Constr. Supervsed Hist. Profiles

5 Supervised Applications

In the supervised application of the cluster histogram approach, we associate class labels with individual stream segments. This is different from standard data stream classification techniques [4], in which class labels are associated either with individual data points or the entire stream. However, in many real applications such as event detection and supervised trend detection, it is desirable to have the ability to classify particular stream segments as opposed to the entire data stream. Such an approach is significantly more flexible, since it is a more general model than either of the other two extreme approaches. For the purpose of training, we assume that a class label is associated with some of the segments in the data stream. These segments are then used to perform the classification process. Let us assume that the set of labels is drawn from the set $\mathcal{L} = \{L_1 \ldots L_m\}$. We also assume that the segments $S_1 \ldots S_r$ have the labels $l_{i_1}, \ldots l_{i_r}$ attached to them. We also assume that each of $l_{i_1} \ldots l_{i_r}$ are drawn from the set $\mathcal{L}$. In this case, we maintain different profile sets for each class label $L_i \in \mathcal{L}$. We assume that the histogram profile for the class L_i is denoted by $\mathcal{H}(L_i)$. As in the previous case, the profile generation method uses a window of length W. At the initialization of the method, we set each histogram profile $\mathcal{H}(L_i)$ to $\{\}$. Next, we start processing the data stream using a similar micro-clustering algorithm as discussed in the

Algorithm *ClassifyTeststream*(Hist. Window: W,
Histogram Profiles: $\mathcal{H}(L_1) \ldots \mathcal{H}(L_m)$, Clusters: $\mathcal{C}_1 \ldots \mathcal{C}_k$);
begin
for the next point $\overline{X}$ in the test stream **do**
begin
Determine closest micro-cluster $\mathcal{C}_i$ to $\overline{X}$;
Compute histogram profile $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$;
Compute closest q profiles to $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$
to any histogram in $\cup_{i=1}^{m} \mathcal{H}(L_i)$;
Compute the majority label $\mathcal{ML}$ in $\cup_{i=1}^{m} \mathcal{H}(L_i)$;
return($\mathcal{ML}$);
end
end

Figure 3: Classification Process

unsupervised version of the algorithm. However, in this case, we use the training segments to create *supervised histogram segment profiles* from the data stream. These supervised histogram profiles provide an understanding of the differences in density behavior of different classes of segments in the data stream.

As in the previous case, we use the data from the underlying stream to update the micro-clusters using a nearest neighbor assignment. After the micro-clusters have been updated, we determine if that segment is associated with a particular class label. A segment of length W is said to be associated with a particular class label if it intersects significantly with a segment S_i associated with class label l. An intersection is said to be significant when a majority of the records in the window W are drawn from the segment S_i. If such an association can be found, then we determine the closest histogram from $\mathcal{H}(l)$ to the segment S. If the corresponding closest distance Δ is greater than the user defined threshold δ, then we add the current profile H to $\mathcal{H}(l)$. While this approach is quite effective for cases in which stream segments are relatively contiguous by class, it may not work quite as effectively in cases in which the class labels are highly mixed in ordering. In such a case, a more effective approach is to keep track of the last W data points for each class, and use them to build the cluster histogram profiles. This requires us to keep track of the last W cluster assignments *for each class*. While this may result in slightly greater overhead because of the simultaneous maintenance of m different histogram profiles, it would continue to work well in cases in which the data records are mixed at the class label level.

As in the case for unsupervised histogram construction, we perform shift tests in each iteration to check if the centroids for the various histograms are valid at each stage of the algorithm. If the histograms are no longer valid, then they need to be deleted from the class specific profile database. The overall algorithm for histogram construction is illustrated in Figure 2.

We note that the testing portion of the classification process can be performed simultaneously with the training portion as long as the training and test data are received in continuous segments, which are tagged with their identity. This is because the histogram profiles which are created by the training process can be simultaneously used for testing, when a test segment if received. However, we present the testing process of the data stream separately for the purpose of conceptual clarity. The pseudo-code for the testing process is illustrated in Figure 3. The essential idea behind this process is to use a modified k-nearest neighbor classification algorithm over the histogram profiles. The first step is to construct the histogram profile $H(\mathcal{C}_1 \ldots \mathcal{C}_k, W)$ from the current window of test data points. This test histogram is determined by finding the assignment of test data points to their corresponding micro-clusters. Next, we determine the closest q histogram profiles from set $\mathcal{H}^u = \cup_{i=1}^{m} \mathcal{H}(L_i)$ to the test histogram. The majority class label out of this set of q histogram profiles is reported as the class label for that segment.

6 Experimental Results

The aim of the experimental results is to show that the approach can yield interesting insights for both supervised and unsupervised applications. The algorithms were tested for both effectiveness and efficiency. Unless otherwise mentioned, used a number of clusters $k = 40$, and window size $W = 900$.

6.1 Data Sets The results were tested over three different data sets, from two different domains. We will describe each of these data sets in some detail slightly later. The data sets along with their respective labels are as follows:

(1) KDD 1999 Network Intrusion Data: The Network Intrusion Detection data stream [2] was a temporal version of the KDD CUP 1999 data. It consists of a series of TCP connections, most of which are *normal* with occasional bursts of attacks at certain times.

(2) Six Speaker Speech Data Set (SpeakerVoc): The voice data sets were generated from six speakers using a VOIP system which constructed network packets in compressed G729 format. Each record contained 15 features corresponding to various characteristics of the speech such as the vocal tract model, pitch, and excitation. We note that this is a particularly difficult problem, since network packets are inherently noisy, and the compressed format contains an even smaller amount of distinguishing information. Each speaker created VOIP data for about 90 seconds of speech. We divided the

stream for each speaker into two equal parts. This resulted in 12 segments, which were concatenated with one another to create a contiguous data stream. The ordering of the segments corresponded to the first segment of the six speakers in a particular order followed by the second segment of the six speakers.

(3) Speech Silence Data Set (TalkSilence): The second voice data set is also a VOIP data set, but in this case one data set corresponds to voice, whereas the other corresponds to a period of silence. In this case, each data set corresponds to about 10 seconds of speech or silence.

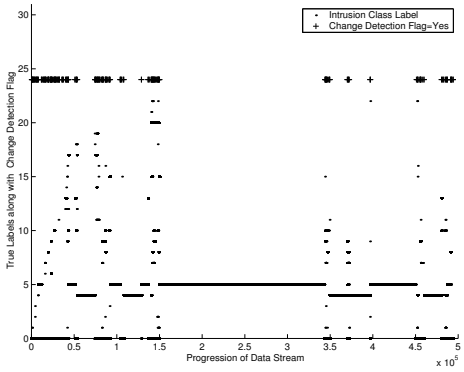

Figure 4: Intrusion Labels vs. Change Alarm

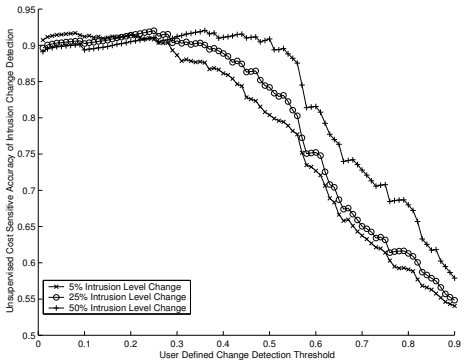

Figure 5: Unsup. Cost Sens. Intrusion Detection

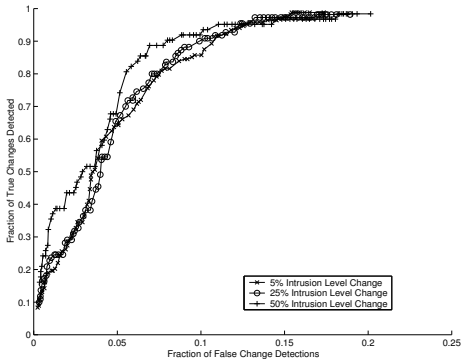

Figure 6: True Detection versus False Alarms

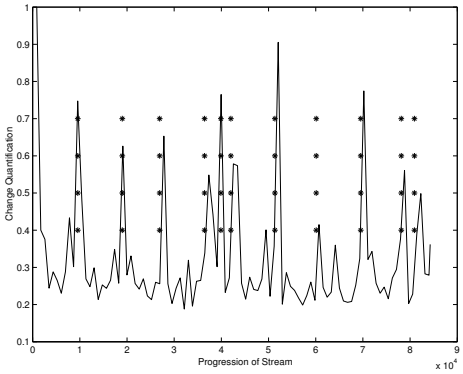

Figure 7: Change Quantification (Six Speaker)

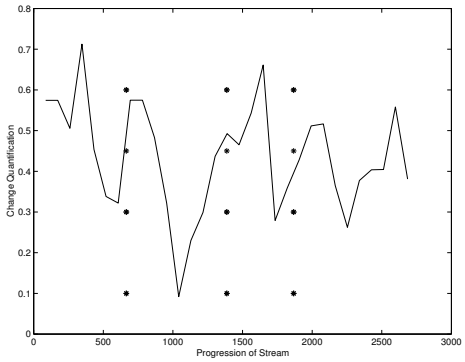

Figure 8: Change Quantification (*TalkSilence*)

6.2 Unsupervised Applications While the KDD Cup 1999 data set has been widely used for *supervised* intrusion detection, we will use it to test whether intrusion detection can be performed in an unsupervised way by using the change detection technique. Such a methodology is quite useful in applications where a significant amount of training data is not available a-priori. First, in order to provide a feel of the nature of the intrusions in the data, we have plotted the intrusions as a function of the progression of the network data stream in Figure 4. On the X-axis, we have illustrated the progression of the data stream, whereas the (numerical id of the) class labels is illustrated on the Y-axis. At the top of the same Figure, we have illustrated the points in the data stream at which the change detection parameter lies above the user defined threshold of 0.5 units. These points are each denoted by a '+'. It is clear that whenever there is a change in the intrusion label, a '+' appears at the corresponding

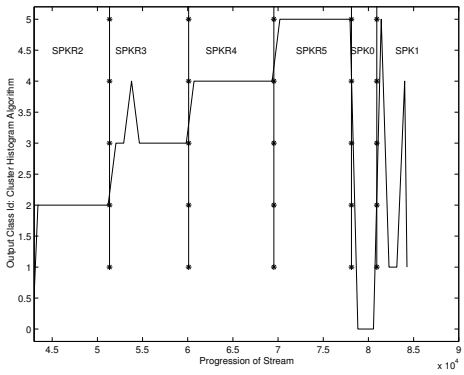

Figure 9: Sup. Classification (Six Speaker)

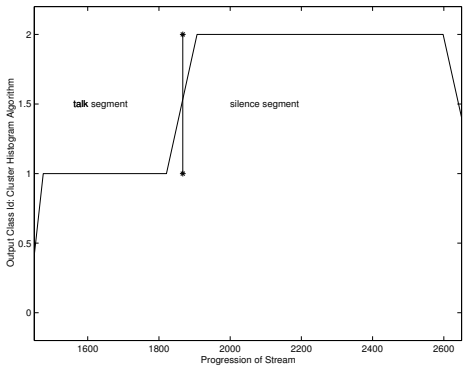

Figure 10: Sup. Classification (*TalkSilence*)

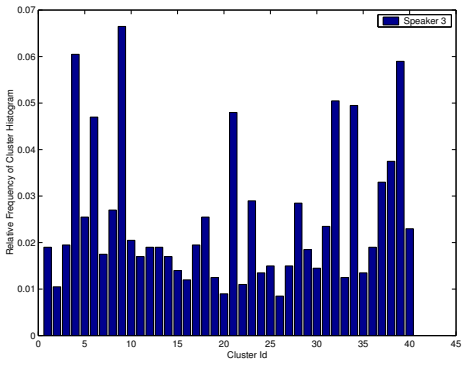

Figure 11: Example Histogram (Speaker 3)

point in the data stream progression. In some cases, this change may correspond to the beginning of an intrusion, and in other cases it corresponds to the end of the intrusion. The aim of the change detection approach is only to determine any significant shifts in the intrusion behavior of the underlying data. It is also important to note that the change points were computed without any knowledge of the class labels. This provides additional evidence about the effectiveness of the process.

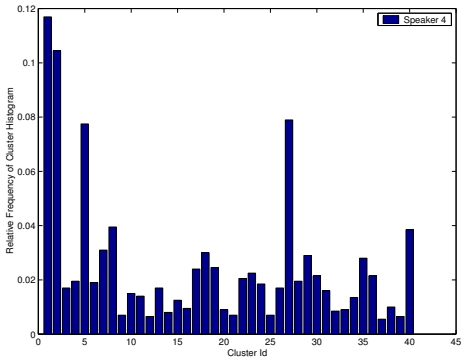

Figure 12: Example Histogram (Speaker 4)

The aim of Figure 4 is to give a qualitative feel of how intrusions are related to the change detection threshold. In order to test this relationship in a more rigorous way, we computed the *true change points* in the data in terms of the intrusion behavior. These true change points were determined using the intrusion labels. A given window was considered a change point if the percentage of non-overlapping intrusion types in the data varied by more than a certain percentage from those in the immediately preceding window. This was specifically computed by determining the relative frequency distribution over the different intrusion labels for the two windows. A window was considered a change point if the difference in the relative frequency distributions[4] from that of its preceding window was more than a certain threshold. Note that a higher change threshold only tries to detect drastic changes in intrusion behavior over two successive windows, and may sometimes ignore small or slowly developing (dying) attacks. We used three different thresholds corresponding to 5% change, 25% change, and 50% change points. We then used the unsupervised method (on only the feature attributes) to determine the change quantification over successive windows in the data. By using thresholds on this change quantification series, it is possible to predict particular change detection points of interest. In many cases, these change detection points may not be true change detection points, whereas in other cases true change detection points may be missed. We note that since true change detection points are much more rare than non-change points, it is more important to detect the true changes versus raising false change alarms at given points. Therefore, we used a cost sensitive method to measure the accuracy of the change. In this technique, each event was inversely weighted by its

[4]The difference between two relative frequency distributions was defined as the sum of the absolute differences of the relative frequencies over different class labels.

frequency. This naturally leads to a greater importance for the change detection points as compared to the non-change points. In Figure 6, we have illustrated the cost sensitive accuracy of the method with different change detection thresholds. It is clear that the use of higher thresholds first leads to a slight increase in the accuracy and then leads to a reduction in accuracy. This is because increasing the change detection point upto 0.6 leads to a considerable reduction in the false detections without missing many true detections. Beyond a threshold of 0.6, a larger number of true change points are missed. Since the missing of true change detection points is penalized to a greater extent, thresholds which are larger than 0.6 lead to a rapid reduction in the cost sensitive accuracy. However, it is interesting to note that the cost sensitive accuracy is larger than 90% for all three cases. We note that the *supervised approach* from the winning entry in the KDD Cup 1999 contest shows an approximately 95.5% accuracy using the analogue[5] of our cost sensitive computation. While these results are not directly comparable[6], the effectiveness of an *unsupervised* change detection approach in detecting intrusions is quite interesting and applicable in real scenarios. In order to make the tradeoffs between between missing true change points and detecting false change points clear, we have plotted the fraction of true change misses versus the false detections in Figure 5. An interesting observation from these results is that even for a very small fraction (about 10%) of false detections, it is possible to detect most of the change points. This is an endorsement of the fact that even when class labels are not available, it is possible to detect the intrusions effectively using a change detection approach.

Next, we will discuss the results using the voice data sets. For the *SpeakerVoc* data set, we utilized the change detection method to determine whether the system could determine any changes in behavior when the voice stream changed from one speaker to another. As noted earlier, the 6-speaker data set was divided into 12 segments, such that each segment contained the voice data for one speaker. We calculated the change value over the entire stream and we have plotted this value with progression of the stream in Figure 7. An interesting observation from Figure 7 is that the change quantification peaks at various points in the data stream. In the same figure, we have also used an asterisk the positions on the stream which correspond to the true segmentation of the stream from one speaker to the next. *It is interesting to see that the change quantification peaks precisely coincide with the true moments of change.* This is because cluster histograms are a comprehensive snapshot of the density distribution of stream segments, and even small changes in the data can be detected using this method. As a result, the method can be used to effectively quantify change points in the data stream.

The same results were observed for the second data set containing voice and silence records. We segmented the stream into 4 alternating segments of voice and silence. The results are illustrated in Figure 8. In this case, the peaks are slightly less pronounced than in the case of the six speaker data set. This is because the data set had a much higher level of noise than the 6-speaker data set. In this case, we have illustrated the true change points in the data stream with asterisks. As in the previous case, the peaks in the change quantification correspond to the true change points in the data stream.

6.2.1 Outlier Detection Application For the network intrusion data set, the algorithm detected over 37 outlier segments in the course of stream progression. We note that each of the different kinds of network attacks appeared as one, two or three candidates in the list of outliers. Typically, these outliers were generated in the earliest stage of the occurrence of the intrusions. Of the 37 outliers, only 6 of them were generated from the normal records in the data, whereas 31 were generated from the intrusion attacks.

For the six speaker data set, a sequence of two to three outliers were generated at the beginning of (the first portion of) each speaker segment. A total of 19 outliers were generated. An interesting observation was that while each speaker portion was divided into two halves in the data stream, the outliers were mostly generated from the first half of each speaker segment. From the 19 outlier segments, a total of 16 were generated from the first set of segments of each speaker. This was because the new density distributions for each segment were encountered for the first time in each new speaker segment. Each time a new speaker was encountered, the corresponding density distributions were declared as outliers in the beginning. However, in later stages, the histograms for these speakers were recognized and no longer reported as outliers. Thus, the overall observation is that a new event (which has not occurred before in the stream) results in the detection of an outlier stream segment. This is because even when a small change occurs in the underlying data, it shows up in the stream distribution; and consequently also in the cluster histograms. A similar behavior was observed for the *TalkSilence* data set in which a total of 8 outliers

[5]We determined the cost sensitive accuracy using the binary classification version of the problem in which a given window was either normal or an intrusion.

[6]This is because the former classifies individual points whereas the latter detects changes in the intrusion trends.

were determined. Of these, 7 outliers were generated from the first portion of each stream segment. This shows that the outlier detection approach can effectively find new patterns which have not occurred before in the data stream.

6.3 Supervised Applications The aim is to determine events of interest associated with particular segments. The speech data sets were more useful for this purpose, since class labels were associated with individual stream segments (rather than individual data points). We noted earlier that each speaker of the *SpeakerVoc* data set was segmented into two portions. The first half of the stream contained the first half of each of the six speaker samples in a particular order, whereas the second half contained the remaining portion of each of the six speakers. Therefore, we used the first half of the data stream for training and the second half for testing. We used the same strategy with the voice-silence data set. We note that the obvious strategy for classification on individual data records is not practical because of the noisy nature of the voice domain. Each voice-packet corresponds to only about 10ms of speech, which could very well correspond to a silence gap. Therefore, the classification behavior may only be inferred from the *overall distribution* of the packets in the data. For example, for the case of the *SpeakerVoc* data set, the *record-level* classification accuracy[7] of a nearest neighbor classifier was only 23.1%, and for the *TalkSilence* data set (with two classes), the accuracy was 58.3%. These accuracies are not significantly better than those of a random classifier, and this illustrates the noise in the data at the record level.

We first tested the cluster histogram algorithm on the *SpeakerVoc* data set. In Figure 9, we have illustrated the class label id returned by the algorithm with progression of the data stream. We note that we have illustrated only the second portion of the data stream, and have excluded the training portion of the data stream. Consequently, the numbers on the X-axis start off at high values representing the end of the training stream and the beginning of the test stream. The test stream contains 6 segments; one for each of the second portions of each speaker segment. We have also used vertical lines to separate out the 6 speakers in Figure 9. The *true* speaker ids occur in the order 2, 3, 4, 5, 0, 1. It is interesting to see that the class label ids almost exactly correspond to this order. The classification was performed at about 40 points in the data stream. Of these 40 points, only three points of mis-classification were observed in the entire data stream. Out of these three misclassification points, two points were observed for speaker 1 and one point was observed for speaker 3. This corresponds to a 92.5% accuracy. Thus, a fairly reliable distinguishing of speakers was achieved with the use of this approach. Similar results were obtained with the use of the voice-silence data sets. The results are illustrated in Figure 10. In this case, perfect classification could be achieved between the voice and silence data sets. Further insights may be obtained by examining the frequently occurring histograms in the various portion of the speech data. For example, the most frequently occurring histogram for speaker 3 (in the training data) is illustrated in Figure 11, and the most frequently occurring histogram for speaker 4 is illustrated in Figure 12. The clear difference between the two histogram distributions illustrates the difference in density distributions between these speakers. The other frequently occurring histograms for the same speaker were also very similar in shape to this base segment. During the classification process, these pre-stored (training) histograms were repeatedly selected by the classifier during the nearest neighbor classification process.

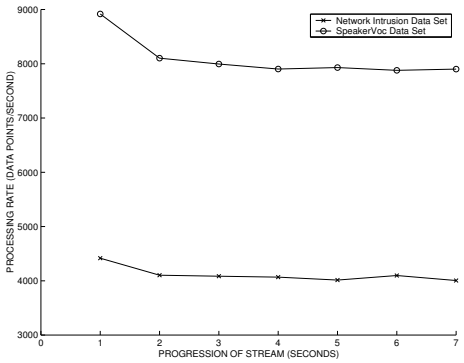

Figure 13: Proc. Rate (Stream Progression)

6.4 Efficiency Results All tests were performed on a 2.2 GHz processor with Windows XP operating system and 256 MB of main memory. In Figure 13, we have illustrated the rate of processing (in terms of data points per second with progression of the stream) of the *SpeakerVoc* and Network Intrusion streams. These results are illustrated with the use of 40 micro-clusters, and a window size of 900 data points. It is clear that in each case, several thousand data points are processed per second. The difference in processing times between the two data streams is because of the difference in dimensionality between the two data sets. The rate of processing is inversely related to the number of micro-clusters because of the larger number of distance

[7] Even a random classifier achieves an accuracy of about 18.1% on the *SpeakerVoc* data set.

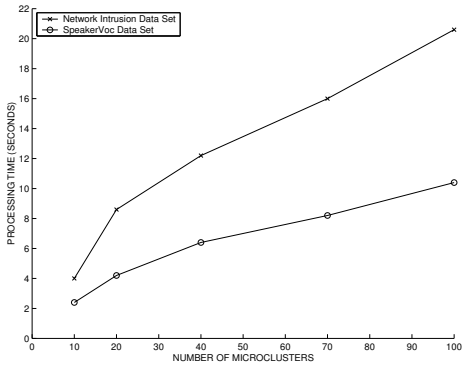

Figure 14: Processing Rate (Incr. Num. of Clusters)

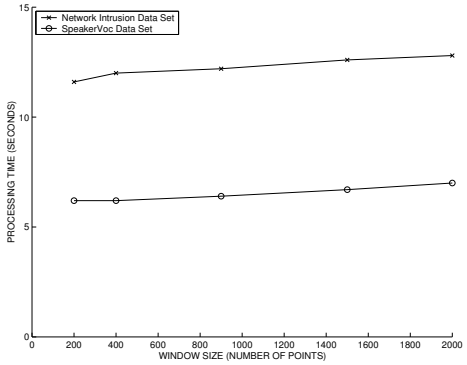

Figure 15: Processing Rate (Incr. Wind. Size)

calculations with greater number of micro-clusters. In Figure 14, we have illustrated the effect of the number of micro-clusters on the processing rate for a fixed window $W = 900$. We have illustrated the time required for processing $50,000$ data points of each stream for different numbers of micro-clusters. It is evident that the time required for processing increases linearly with the number of micro-clusters. Finally we tested the efficiency with varying window size W. In Figure 15, we have illustrated the processing rate with increasing window size. On the X-axis we have illustrated the window size, and the processing time for $50,000$ data points is illustrated on the Y-axis. An interesting observation is that the processing time is practically insensitive to the window size. This is because the majority of the time is spent in micro-clustering (which does not depend on window size) and only a small portion of the time is spent in histogram maintenance.

7 Discussion and Conclusions

In this paper, we discussed a method for profiling and mining stream segments using the technique of cluster histograms. The cluster histogram approach constructs a compact historical summary of the data stream, which can be used for a number of data mining tasks. As concrete examples, we illustrated methods for change detection over different segments, stream anomaly determination, and supervised event detection. While this paper has discussed a variety of useful applications, our aim was primarily to give a flavor of the versatility and effectiveness of the cluster histogram approach for different problems. The reason for this flexibility of the cluster histogram technique is that it encodes detailed density information about the data which can be leveraged for a wide variety of data mining tasks. In future work, we expect to further extend and study the power of cluster histograms in solving data mining problems over diverse application domains.

References

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

[2] C. C. Aggarwal, J. Han, J. Wang, and P. Yu, *A Framework for Clustering Evolving Data Streams*, in VLDB Conference Proceedings, (2003) pp. 81–92.

[3] C. C. Aggarwal, *A Framework for Diagnosing Changes in Evolving Data Streams*, in ACM SIGMOD Conference Proceedings, (2003) pp. 575–586.

[4] P. Domingos, and G. Hulten, *Mining High-Speed Data Streams*, in ACM KDD Conference Proceedings (2000), pp. 71–80.

[5] S. Guha, N. Koudas and K. Shim, *Data Streams and Histograms*, in ACM Symposium on Theory of Computing Proceedings (2001), pp. 471–475.

[6] D. Kifer, S. Ben-David, and J. Gehrke, *Detecting Change in Data Streams*, in VLDB Conference Proceedings (2004), pp. 180–191.

[7] E. Knorr, and R. Ng, *Algorithms for Mining Distance Based Outliers in Large Data Sets*, in VLDB Conference Proceedings (1998), pp. 392–403.

[8] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, *Streaming-Data Algorithms For High-Quality Clustering*, in IEEE ICDE Conference Proceedings (2002), pp. 685–696.

[9] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Monographs on Statistics and Applied Probability, Chapman and Hall, (1986).

[10] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander, *A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases*, in IEEE ICDE Conference Proceedings (1998), pp. 324–331.

[11] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, in ACM SIGMOD Conference Proceedings (1996), pp. 103–114.

[12] T. Zhang, R. Ramakrishnan, and M. Livny, *Fast Density Estimation Using CF-Kernel for Very Large Databases*, in ACM KDD Conference Proceedings (1999), pp. 312–316.