E-Stream: Evolution-Based Technique for Stream Clustering

Komkrit Udommanetanakit, Thanawin Rakthanmanon, and Kitsana Waiyamai

Department of Computer Engineering, Faculty of Engineering
Kasetsart University, Bangkok 10900, Thailand
{fengtwr, fengknw}@ku.ac.th

Abstract. Data streams have recently attracted attention for their applicability to numerous domains including credit fraud detection, network intrusion detection, and click streams. Stream clustering is a technique that performs cluster analysis of data streams that is able to monitor the results in real time. A data stream is continuously generated sequences of data for which the characteristics of the data evolve over time. A good stream clustering algorithm should recognize such evolution and yield a cluster model that conforms to the current data. In this paper, we propose a new technique for stream clustering which supports five evolutions that are appearance, disappearance, self-evolution, merge and split.

1 Introduction

Stream clustering is a technique that performs cluster analysis of data streams that is able to produce results in real time. The ability to process data in a single pass and summarize it, while using limited memory, is crucial to stream clustering.

Several efficient stream clustering techniques have been presented recently, such as STREAM [9], CluStream [2], and HPStream [1]. STREAM is a k-median based algorithm that can achieve a constant factor approximation. CluStream divides the clustering process into an online and offline process. Data summarization is performed online, while clustering of the summarized data is performed offline. Experiments show that CluStream yields better cluster quality than STREAM. HPStream is the most recent stream clustering technique, which utilizes a fading concept, data representation, and dimension projection. HPStream achieves better clustering quality than the above algorithms.

Since the characteristics of the data evolve over time, various types of evolution should be defined and supported by the algorithm. Almost of existing algorithms support few types of evolution. The objective of the research reported here is to improve existing stream clustering algorithms by supporting 5 evolutions with a new suitable cluster representation and a distance function. Experimental results show that this technique yields better cluster quality than HPStream.

The remaining of the paper is organized as follows. Section 2 introduces basic concepts and definitions. Section 3 presents our stream clustering algorithm called *E-Stream*. Section 4 compares the performance of E-Stream and HPStream with respect to the synthetic dataset. Conclusions are discussed in Section 5.

R. Alhajj et al. (Eds.): ADMA 2007, LNAI 4632, pp. 605–615, 2007.

2 Basic Concepts and Definitions

In this section, we introduce some basic concepts and definitions that will be used subsequently.

The data stream consists of a set of multidimensional records $X_1 \ldots X_k \ldots$ arriving at time stamps $T_1 \ldots T_k \ldots$. Each data point X_i is a multidimensional record containing d dimensions, denote by $X_i = (x_i^1 \ldots x_i^d)$.

An isolated data point is a data point that is not a member of any clusters. Isolated data points remain in the system for cluster appearance computations.

An inactive cluster is a cluster that has a low weight. It can become an active cluster if its weight is increased.

An active Cluster is a cluster that can assemble incoming data if there is sufficient similarity score.

A cluster is a collection of data that has been memorized for processing in the system. It can be an isolated data point, an inactive cluster, or an active cluster.

Fading decreases weight of data over time. In a data stream that has evolving data; older data should have lesser weight. We decrease weight of every cluster over time to achieve a fast adaptive cluster model. Let λ be the decay rate and t be elapsed time, the fading function is

$$f(t) = 2^{-\lambda t} \tag{1}$$

Weight of a cluster is the number of data elements in a cluster. Weight is determined according to the fading function. Initially, each data element has a weight of 1. A cluster can be increased its weight by assembling incoming data points or merging with other clusters.

2.1 Fading Cluster Structure with Histogram: FCH

Each cluster is represented as a *Fading Cluster Structure (FCS)* [1] utilizing a α-bin histogram for each feature of the dataset. We called our cluster representation *Fading Cluster Structure with Histogram (FCH)*. Let T_i be the time when data point x_i is retrieved, and suppose t be the current time then $f(t-T_i)$ is the fading weight of data point x_i.

FC1(t) is a vector of weighted sumation of data feature values at time t. The j^{th} dimension is

$$FC1^j(t) = \sum_{i=1}^{N} f(t - T_i) \cdot (x_i^j) \tag{2}$$

FC2(t) is the weighted sum of squares of each data feature at time t. The j^{th} dimension is,

$$FC2^j(t) = \sum_{i=1}^{N} f(t - T_i) \cdot (x_i^j)^2 \tag{3}$$

W(t) is a sum of all weights of data points in the cluster at time *t*,

$$W(t) = \sum_{i=1}^{N} f(t - T_i) \tag{4}$$

H(t) is a α-bin histogram of data values. For the j^{th} feature at time *t*, the elements of H^j *are*

$$H_l^j(t) = \sum_{i=1}^{N} f(t - T_i) \cdot (x_i^j) \cdot (y_{il}^j) \tag{5}$$

Where

$$y_{il} = \begin{cases} 1 & if \quad l \cdot b + left \le x_i \le (l+1) \cdot b + left \\ 0 & otherwise \end{cases} \tag{6}$$

$$left = \min(x_i^j) \tag{7}$$

$$right = \max(x_i^j) \tag{8}$$

$$b = \frac{left + right}{\alpha} \tag{9}$$

left is a minimum value, *right* is a maximum value in this cluster, *b* is a size of each bin, and y_{il} is a weigth of x_i in the l^{th} bin.

2.2 Histogram Management

We utilize a histogram of cluster data values to identify cluster splits. A α-bin histogram summarizes the distribution of cluster data for each dimension of each cluster. The range of each bin is calculated as the difference between the maximum and minimum feature values divided by α. When the maximum or minimum value changes, we calculate a new range and update the values in each range from the intersection between the new and old ranges. Each cluster has a histogram of feature values, but the histogram is utilized only for the split of active clusters. Only an active cluster can assemble an incoming data point.

Cluster split is based on the distribution of feature values as summarized by the cluster histogram [1]. If a statistically significant valley is found between two peaks in histogram values along any dimensions, the cluster is split. If more than one split valley occurs in the histogram values, the value with the minimum height relative to the surrounding peaks is chosen. When a cluster is split, the histogram in that dimension is split and the other dimensions are weighted from the split dimension. FC1, FC2 and W are recalculated from the new cluster histograms.

2.3 Distance Functions

Cluster-Point distance is a distance from a data point to the center of a cluster, ormalized by the standard deviation (radius) of the cluster data in each dimension.

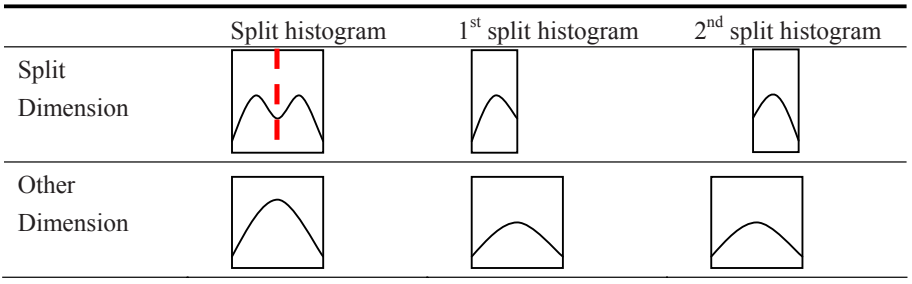

Fig. 1. Histogram management in a split dimensionop and other dimension

This function is used to find the closest active cluster for an incoming data point. A cluster with a larger radius yields a lower distance. Let *C* be an active cluster and *x* be a data point, the Cluster-Point distance is

$$dist(C,x)=\frac{1}{d}\cdot\sum_{j=1}^{d}\left|\frac{center_C^j - x^j}{radius_C^j}\right| \tag{10}$$

Cluster-Cluster distance is a difference between centers of two clusters. It is used to find the closest pair of cluster. If C_a and C_b are two clusters, the cluster-cluster distance is

$$dist(C_a,C_b)=\frac{1}{d}\sum_{j=1}^{d}\left|center_{C_a}^j - center_{C_b}^j\right| \tag{11}$$

3 The Algorithm

In this section, we first describe our idea. Then, we present the E-Stream algorithm composing of a set of sub-algorithms.

3.1 Our Idea

The behavior of data in a data stream can evolve over time. We can classify this evolution into five categories: appearance, disappearance, self evolution, merge, and split. A clustering model can start from empty. In the beginning, incoming data are considered as isolated clusters. A cluster is formed when a sufficiently dense region appears. The cluster assembles similar data and increases its existence. When a set of clusters has been identified, incoming data must be assigned to a cluster based on similarity score, or the datum may be classified as an isolated. In every change of a cluster we check if any of the following evolutions occur and handle it.

Appearance: A new cluster can appear if there is a sufficiently dense group of data points in one area. Initially, such elements appear as a group of outliers, but (as more data appears in a neighborhood) they are recognized as a cluster.

Disappearance: Existing clusters can disappear because the existence of data is diminished over time. Clusters that contain only old data are faded and eventually disappear because they do not represent the presence of data.

Self-evolution: Data can change their behaviors, which cause size or position of a cluster to evolve. Evolution can be done faster if the data can fade.

Merge: A pair of clusters can be merged if their characteristics are very similar. Merged clusters must cover the behavior of the pair.

Split: A cluster can be split into two smaller clusters if the behavior inside the cluster is obviously separated.

3.2 E-Stream Algorithm

This section describes E-Stream in details. Following is the list of notations used in our pseudo-code.

- |FCH| = current number of clusters
- $FCH_i.W$ = weight of the i^{th} cluster
- $FCH_i.sd$ = standard deviation of the i^{th} cluster
- S = set of pair of the split cluster

E-Stream is the main algorithm. In line 1, the algorithm starts by retrieving a new data point. In line 2, it fades all clusters and deletes those having insufficient weight. Line 3 performs a histogram analysis and cluster split. Then line 4 checks for overlap clusters and merges them. Line 5 checks the number of clusters and merges the closest pairs if the cluster count exceeds the limit. Line 6 checks all clusters whether their status are active. Lines 7-10 find the closest cluster to the incoming data point. If the distance is less than *radius_factor* then the point is assigned to the cluster, otherwise it is an isolated data point. The flow of control then returns to the top of the algorithm and waits for a new data point.

```
Algorithm E-Stream
1   retrieve new data X_i
2       FadingAll
3       CheckSplit
4       MergeOverlapCluster
5       LimitMaximumCluster
6       FlagActiveCluster
7       (minDistance, index) ← FindClosestCluster
8       if minDistance < radius_factor
9         add x_i to FCH_index
10      else
11          create new FCH from X_i
12  waiting for new data
```

Fig. 2. E-Stream, stream clustering algorithm

FadingAll. The algorithm performs fading of all clusters and deletes the clusters whose weight is less than *remove_threshold*.

CheckSplit is used to verify the splitting criteria in each cluster using the histogram. If a splitting point is found in any cluster then it is split. And store the index pairs of split cluster in S.

CheckMerge is an algorithm for merging pairs of similar clusters. This algorithm checks every pair of clusters and computes the cluster-cluster distance. If the distance is less than *merge_threshold* and the merged pair is not in S then merge the pair.

LimitMaximumCluster is used to limit the number of clusters. This algorithm checks whether the number of clusters is not greater than *maximum_cluster* (an input parameter); if it exceeds then the closest pair of clusters is merged until the number of remaining clusters is less than or equal to the threshold.

FlagActiveCluster is used to check the current active cluster. If the weight of any cluster is greater or equal to active_*threshold* then it is flagged as an active cluster. Otherwise, the flag is cleared.

FindClosestCluster is used to find the distance and index of the closest active cluster for an incoming data point.

Algorithm *FadingAll*	**Algorithm** *CheckSplit*

for i ← 1 to |FCH|
  fading FCH_i
  if $FCH_i.W <$ *fade_threshold*
  delete FCH_i
	for i ← 1 to |FCH|
	  for j ← 1 to d
	  if FCH_{ij} have split point
	  split FCH_i
	  S ← S U {(i, |FCH|)}

Fig. 3. FadingAll and CheckSplit algorithms

Algorithm MergeOverlapCluster	**Algorithm** *LimitMaximumCluster*

for i ← 1 to |FCH|
  for j ← i + 1 to |FCH|
  overlap[i,j] ← dist(FCH_i,FCH_j)
  m ← *merge_threshold*
  if overlap[i,j] > m*(FCH_i.sd+FCH_j.sd)
  if (i, j) not in S
  merge(FCH_i, FCH_j)
	while |FCH| > maximum_cluster
	  for i ← 1 to |FCH|
	  for j ← i + 1 to |FCH|
	  dist[i,j] ← dist(FCH_i, FCH_j)
	  (first, second) ← $\text{argmin}_{(i,j)}$(dist[i,j])
	  merge(FCH_{first}, FCH_{second})

Fig. 4. MergeOverlapCluster and LimitMaximumCluster algorithms

Algorithm *FlagActiveCluster*	**Algorithm** *FindClosestCluster*

for i ← 1 to |FCH|
  if $FCH_i.W >=$ *active_threshold*
  flag FCH_i as active cluster
  else
  remove flag from FCH_i
	for i ← 1 to |FCH|
	  if FCH_i is active cluster
	  dist[i] ←dist(FCH_i, x_i)
	(minDistance, i) ← min(dist[i])
	return (minDistance, i)

Fig. 5. FlagActiveCluster and FindClosestClusterAlgorithms

4 Experimental Results

We tested the algorithm using a synthetic dataset consisting of two dimensions and 8,000 data points. This data changes the behavior of clusters over time. We can segment it into 8 intervals as follows

1. Initially, there are 4 clusters in a steady state. Data point from 1 to 1600.
2. The 5th cluster appears at position (15, 6). Data point from 1601 to 2600.
3. The 1st cluster disappears. Data point from 2601 to 3400.
4. The 4th cluster swells. Data point from 3401 to 4200
5. The 2nd and 5th cluster get closer. Data point from 4201 to 5000.
6. The 2nd and 5th are merged into a bigger cluster. Data point from 5001 to 5600.
7. A 6th cluster is split from the 3rd cluster. Data point from 5601 – 6400.
8. Every cluster is in a steady state again. Data point from 6401 – 8000.

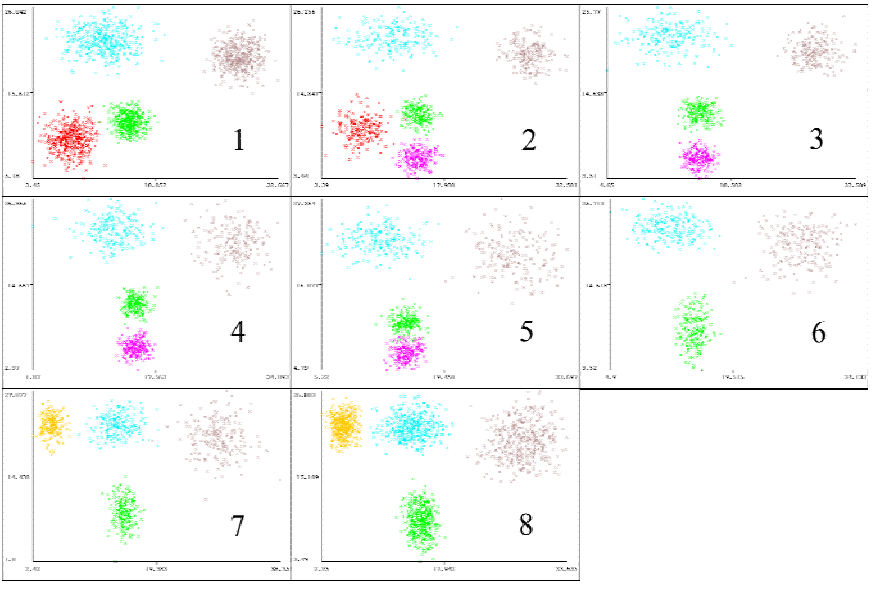

Fig. 6. The 8-Step evolution of the Synthetic Dataset

4.1 Efficiency Test

In this experiment, we set the parameters as in table 1. E-Stream allows the number of clusters to vary dynamically with the constraint of the maximum number of clusters, but requires a limit on the number of clusters. HPStream requires a fixed number of clusters. Since the synthetic dataset has at most 5 clusters in each interval, we used 5 as the cluster (group) count in HPStream, and 10 as the cluster limit in E-Stream. HPStream requires initial data for its initialization process before beginning stream clustering. We, therefore, set it to 100 points.

Table 1. Parameters of each algorithm

algorithm E-Stream	algorithm HPStream
maximum_cluster 10	num_cluster 5
stream_speed 100	stream_speed 100
decay_rate 0.1	decay_rate 0.1
radius_factor 3	radius_factor 3
remove_threshold 0.1	
merge_threshold 1.25	
active_threshold 5	

In the first interval, there are four clusters in a steady-state. Both algorithms yield clusters with little distortion, but E-Stream has a great number of clusters. Because at the beginning E-Stream does not have any active clusters, every incoming data point is considered as an isolated data point. As more data is accumulated, a cluster will appear. On the other hand, HPStream requires initial data (set to 100 points) for offline clustering, so HPStream exhibits better initial clustering than E-Stream.

In the second interval, a new cluster appears. HPStream still yields a better quality because it finds all clusters correctly, but E-Stream yields only little distortion.

In the third interval, an existing cluster disappears. E-Stream yields good clustering while HPStream tries to create a new cluster from existing cluster even though their do not have a significant difference, due to the fixed cluster-count constraint.

In the 4th interval, a cluster swells. In the 5th interval, two clusters are closer, an evolution that is supported by both algorithms. But, HPStream still tries to find five clusters.

In the 6th interval two close clusters are merged. Neither algorithms merge the two clusters in this interval even though their current behaviors are undistinguished.

In the 7th interval, a cluster splits. E-Stream can support this evolution when it receives enough data. But HPStream cannot detect this.

In the 8th interval there are four clusters in a steady state again. E-stream algorithm detects the previous merged case and identifies all clusters correctly within this interval. But HPStream still is confusing by the cluster behavior.

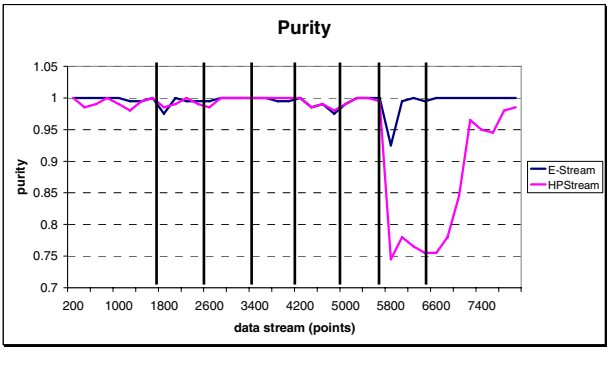

Fig. 7. Purity test between E-Stream and HPStream

In the purity test, E-Stream always has a purity greater than 0.9. But HPStream exhibits a big drop in the seventh interval (cluster split), because the algorithm cannot accommodate this evolution.

In the F-measure test, E-Stream yields an average value much better than HPStream although, there are two intervals where E-Stream has a lower F-measure. HPStream yields better results due to an initial offline process to find the initial clusters. The second instance is the second interval (1601-2600), where E-Stream merged two clusters incorrectly.

From the efficiency test, we can say that HPStream cannot support the evolution of number of clusters because the algorithm constrains it. E-Stream can support all the evolutions in Section 3, even though some evolutions such as merge require a lot of data to detect.

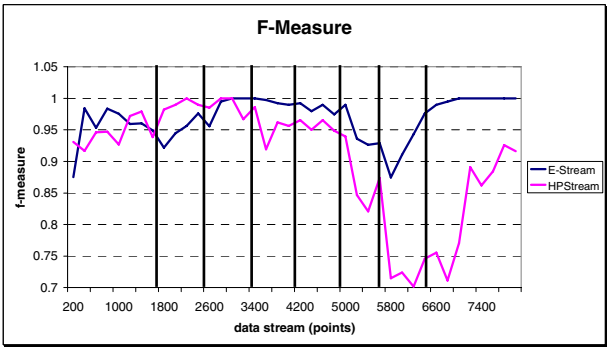

Fig. 8. F-Measure test between E-Stream and HPStream

4.2 Sensitivity with Number of Cluster (Input Parameters)

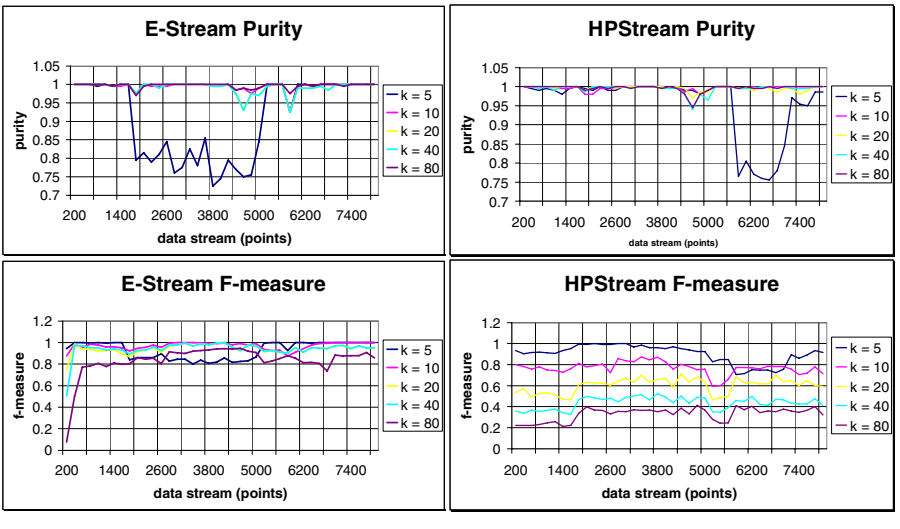

Fig. 9. Sensitivity with number of cluster (input parameter)

From the experiment, the Purity of both algorithms is not sensitive to this input parameter. But in F-measure terms, HPStream has a tendency to drop if the input number of clusters differs greatly from the actual number. E-Stream is not sensitive to this parameter since the number of clusters is not fixed. As long as the maximum number is not exceeded, E-Stream still yields good results.

4.3 Runtime with Number of Data

In this experiment we use a dataset consisting of 500,000 data points with two dimensions and five clusters.

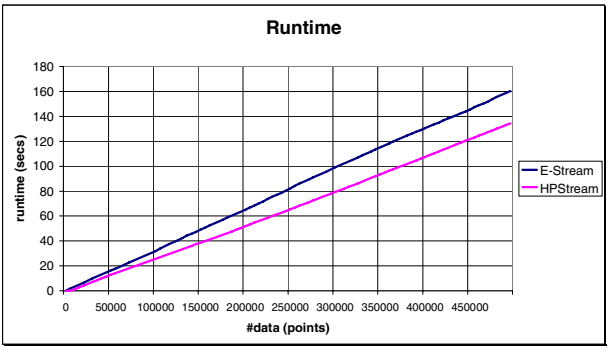

Fig. 10. Runtime with number of data (points)

Both algorithms exhibit linear runtime in number of data points, which is a constraint for stream clustering algorithms.

4.4 Runtime as a Function of Clusters and of Dimensions

To test the runtime as a function of the number of clusters, we use two dimensions, 100,000 data points, and vary the number of data clusters from 5 to 25 in increments of 5 clusters.

For runtime with the number of dimensions test, we use 5 clusters, 100,000 data points, and vary the number of dimensions from 2 to 20.

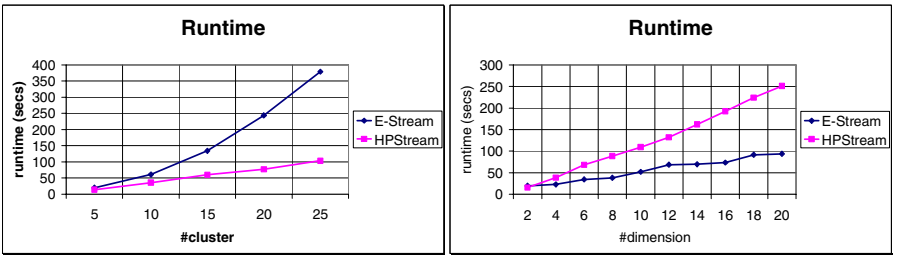

Fig. 11. Runtime with number of clusters and number of dimensions

The results of both experiments are summarized in Figure 15. HPStream exhibits linear runtime in both the number of clusters and the number of dimensions. E-Stream exhibits linear runtime in the number of dimensions but polynomial runtime in the number of clusters. This is due to the merging procedure, which requires $O(k^2)$ time in the number of clusters.

5 Conclusions

This paper proposed a new stream clustering technique called E-Stream which can support five cluster evolutions: appearance, disappearance, self-evolution, merge, and split. These evolutions can normally occur in an evolving data stream. This technique outperforms a well-known technique, HPStream. However, the runtime of the new approach is polynomial with respect to the number clusters.

Acknowledgment. Thanks to J. E. Brucker and P. Vateekul for their reading and comments of this paper.

References

1. Milenova, B.L., Campos, M.M.: Clustering Large Databases with Numeric and Nominial Values Using Orthogonal Projections. In: Proceedings of the 29th VLDB Conference (2003)
2. Aggarwal, C., Han, J., Wang, J., Yu, P.S.: A Framework for Projected Clustering of High Dimensional Data Streams. In: Proceeding of the 30th VLDB conference (2004)
3. Aggarwal, C., Han, J., Wang, J., Yu, P.S.: A Framework for Clustering Evolving Data Streams. In: Proceeding of the 29th VLDB conference (2003)
4. Barbara, D.: Requirements for Clustering Data Streams. In: SIGKDD Explorations (2002)
5. Gaber, M.M., Zaslavsky, A., Krishnaswmy, S.: Mining Data Streams: A Review. In: SIGMOD Record, vol. 34(2) (June 2005)
6. Oh, S., Kang, J., Byun, Y., Park, G., Byun, S.: Intrusion Detection based on Clustering a Data Stream. In: Proceedings of the 2005 Third ACIS International Conference on Software Engineering Research, Management and Applications (2005)
7. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering Data Streams: Theory and Practice. TKDE special issue on clustering 15 (2003)
8. Song, M., Wang, H.: Highly Efficient Incremental Estimation of Gaussian Mixture Models for Online Data Stream Clustering. In: SPIE Conference on Intelligent Computing: Theory And Application III (2005)
9. Zhang, T., Ramakhrisnan, R., Livny, M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. In: Proc. ACM SIGMOD Int. Conf. Management of Data (1996)