SOStream: Self Organizing Density-Based Clustering over Data Stream*

Charlie Isaksson, Margaret H. Dunham, and Michael Hahsler

Department of Computer Science and Engineering,
Southern Methodist University, Dallas, Texas, USA
charlie.isaksson@tektronix.com, {mhd,mhahsler}@lyle.smu.edu

Abstract. In this paper we propose a data stream clustering algorithm, called Self Organizing density based clustering over data Stream (SOStream). This algorithm has several novel features. Instead of using a fixed, user defined similarity threshold or a static grid, SOStream detects structure within fast evolving data streams by automatically adapting the threshold for density-based clustering. It also employs a novel cluster updating strategy which is inspired by competitive learning techniques developed for Self Organizing Maps (SOMs). In addition, SOStream has built-in online functionality to support advanced stream clustering operations including merging and fading. This makes SOStream completely online with no separate offline components. Experiments performed on KDD Cup'99 and artificial datasets indicate that SOStream is an effective and superior algorithm in creating clusters of higher purity while having lower space and time requirements compared to previous stream clustering algorithms.

Keywords: Adaptive Threshold, Data Stream Clustering, Density-Based Clustering, Self Organizing Maps.

1 Introduction

Data stream mining has recently captured an enormous amount of attention. Stream mining can be defined as the process of finding complex structure within a large volume of data where the data evolves over time and arrives in an unbounded stream. A data stream is a sequence of continuously arriving data which imposes a single pass restriction where random access to the data is not feasible. Moreover, it is impractical to store all the arriving data. In this case, cluster features or synopses that typically include descriptive statistics for a cluster are used. In many cases, data stream algorithms have to observe space and time constraints. Stream clustering algorithms are used to group events based on similarity between features. Data arriving in streams often contain noise and outliers. Thus, data stream clustering should be able to detect, distinguish and filter this data prior to clustering.

Inspired by both DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [1] and SOM (Self Organizing Maps) [2], we propose a new data stream clustering algorithm, Self Organizing density-based clustering over data Stream (SOStream).

* This work was supported in part by the U.S. National Science Foundation NSF-IIS-0948893.

P. Perner (Ed.): MLDM 2012, LNAI 7376, pp. 264–278, 2012.

SOStream is a density-based clustering algorithm that can adapt its threshold to the data stream. It uses an exponential fading function to reduce the impact of old data whose relevance diminishes over time. SOStream has the following novel features:

- Setting a threshold manually for density-based clustering (similarity threshold, grid size, etc.) is difficult and if this parameter is set to an unsuitable value, then the algorithm will suffer from overfitting, while at the other extreme the clustering is unstable. SOStream addresses this problem by using a dynamically learned threshold value for each cluster based on the idea of building neighborhoods with a minimum number of points.
- SOStream employs a novel cluster updating strategy which is inspired by competitive learning techniques developed for Self Organizing Maps (SOMs) [2] and CURE (Clustering Using REpresentatives) [3]. CURE utilizes a unique shrinking strategy that encouraged us to implement the same methodology for SOStream. The micro-clusters that are formed after shrinking are used as a representative of the global cluster. The shrinking procedure also helps to correctly identify highly-overlapped clusters (See Figure 1). As a result, the clusters become less sensitive to outliers.
- All aspects of SOStream (including deletion, addition, merging, and fading of clusters) are performed online.

We conduct experiments using both a synthetic and the KDD Cup'99 datasets [4], and demonstrate that SOStream outperforms the state-of-the-art algorithms MR-Stream [5] and D-Stream [6] without the use of an offline component. Moreover, SOStream dynamically adapts its similarity threshold. SOStream achieves better clustering quality in terms of cluster purity and utilizes less memory which is a key advantage for any data stream algorithms.

Throughout this paper, we use threshold and radius interchangeably to specifically refer to a value that is used to cluster a new point into suitable micro-cluster or to find the neighborhood of the winning micro-cluster.

The remainder of this paper is organized in the following manner: Section 2 surveys related work; Section 3 presents the SOStream framework; Section 4 presents results of experiments evaluating the performance of SOStream; and Section 5 concludes the paper.

2 Related Work

We first review the most important data stream clustering algorithms to highlight the novel features of SOStream.

E-Stream [7] starts empty and for every new point either a new cluster is created around the incoming data point or the point is mapped into one of the existing clusters based on a radius threshold. Any cluster not meeting a predefined density level is considered inactive and remains isolated until achieving a desired weight. Cluster weights decrease over time to reduce the influence of older data points. Clusters not active for a certain time period may be deleted from the data space. Also, for each step, two clusters may be merged because the overlap is sufficiently large (or the maximum cluster limit is

Table 1. Features of different data stream clustering algorithms

Algorithms	new cluster	remove	merge	fade	split
SOStream	✓	✓	✓	✓	x
E-Stream	✓	✓	✓	✓	✓
CluStream	✓	✓	offline	x	x
DenStream	✓	✓	offline	✓	x
OpticsStream	✓	✓	offline	✓	x
HPStream	✓	✓	x	✓	x
WSTREAM	✓	✓	✓	✓	x
D-Stream	✓	✓	offline	✓	x
MR-Stream	✓	✓	offline	✓	x

reached) or one cluster may be split into two sub-clusters if internal data is too diverse. The data is summarized into an α-bin histogram, and the split is made if a deep valley between two significant peaks is found.

CluStream [8] uses an online micro-clustering component to periodically store detailed summary statistics in a fast data stream while an offline macro-clustering component uses the summary statistics in conjunction with other user input to provide the user with a quick understanding of the clusters whenever required.

DenStream [9] maintains two lists: one with potential micro-clusters and the other with outlier micro-clusters. As each new point arrives, an attempt is made to merge it into one of the nearest potential micro-clusters. If the radius of the resulting micro-cluster is larger than specified, the merge is omitted, and an attempt is made to merge the point with the nearest outlier micro-cluster. If this resulting radius is larger than specified, the merge is omitted and a new outlier micro-cluster is created centered at the point. If any of the outlier micro-clusters exceeds a specified weight, they are moved into the potential micro-clusters list.

OpticsStream [10] is an online visualization algorithm producing a map representing the clustering structure where each valley represents a cluster. It adds the ordering technique from OPTICS [11] (not suitable for data stream) on top of a density-based algorithm (such as DenStream) in order to manage the cluster dynamics.

HPStream [12] is an online algorithm that discovers well-defined clusters based on a different subset of the dimensions of d-dimensional data points. For each cluster a d-dimensional vector is maintained that indicates which of the dimensions are included for continuous assignment of incoming data points to an appropriate cluster. The algorithm first assigns the received streaming data point to each of the existing clusters, computes the radii, selects the dimensions with the smallest radii, and creates a d-dimensional vector for each cluster. Next, the Manhattan distance is computed between the incoming data point and the centroid of each existing cluster. The winner is found by returning the largest average distance along the included dimensions, and the radius is computed for the winning cluster and compared to the winning distance. Then, either a new cluster is created centered at incoming data point or the incoming data point is added to the winning cluster. Clusters are removed if the number of clusters exceeds the user defined threshold or if they contain zero dimensions.

WSTREAM [13] is density-based and discovers cluster structure by maintaining a list of rectangular windows that are incrementally adjusted over time. Each window moves based on the centroid of the cluster which is incrementally recomputed whenever new data points are inserted. It incrementally contracts or expands based on the approximated kernel density and the user defined bandwidth matrix. If two windows overlap, the proportion of intersecting data points to the remaining points in each window is computed, and, upon meeting a user defined threshold, the windows are merged. Periodically, the weights of the stored windows are checked, and a window is removed if its weight is less than the defined minimum threshold (the window is considered to be an outlier).

D-Stream [6] is density-based and works on the basis of a time step model which starts by initializing an empty hash table grid list. An online component reads the incoming raw data record, and this record is mapped to the grid list or inserted into the grid list if it does not exist. After the insertion, the characteristic vector (containing all the information about the grid) is updated. Thus, the online component partitions the data into many corresponding density grids forming grid clusters while the offline component dynamically adjusts the clusters every gap time. The gap is the key decision factor for inspecting each grid and adjusting the cluster. If the grid is empty or receives no new value for a long period of time then it is removed.

MR-Stream [5] finds clusters at versatile granularities by recursively partitioning the data space into well-defined cells by using a tree data structure. MR-Stream facilitates both online and offline components. Table 1 summarizes features of these stream clustering algorithms and shows those possessed by our new SOStream algorithm. Although not contained in the SOStream algorithm presented in this paper, we have developed a splitting function which is easily incorporated into the SOStream algorithm presented later in this paper. This will be reported in subsequent publications.

3 SOStream Framework

A key issue for clustering stream data is the online constraint, which imposes a single pass restriction over the incoming data points. Although many previously proposed stream clustering algorithms have an offline component, this is neither desirable nor necessary. In this section we introduce Self Organizing density-based clustering over data Stream (SOStream) and highlight some of its novel features.

3.1 SOStream Overview

We assume that the data stream consists of a sequence of d-dimensional input vectors where $v(t)$ is used to indicate the input vector at time t, where $t = (1, 2, 3...)$. For every time step t, SOStream is represented by a set of micro-clusters $M(t) = \{ N_1, N_2, ..., N_k \}$, where for each cluster a tuple with three elements $N_i = (n_i, r_i, C_i)$ is stored. n_i is the number of data points assigned to N_i, r_i is the cluster's radius and C_i is the centroid. The tuple is a form of synopsis or cluster feature (CF) vector. Cluster feature vectors were introduced in the non-data stream clustering algorithm BIRCH [14].

As these values change over time we include in the following description the time point where needed to identify the value at a particular time. Thus, $C_i(t)$ indicates the centroid for cluster N_i at time t. SOStream uses a centroid to describe the cluster as a d-dimensional vector. The number of clusters varies over time and depends upon the complexity of the input data. SOStream has built-in stream clustering operations to dynamically create, merge, and remove clusters in an online manner. In addition, an exponential fading function can be used to gradually reduce the impact of historical data.

SOStream uses competitive learning as introduced for SOMs where a winner influences its immediate neighborhood [2]. For each new input vector, $v(t)$, the winning cluster is determined by measuring the distance (e.g. Euclidean distance) between each existing cluster centroid and the current input vector:

$$N_{win}(t) = \underset{N_i \in M(t)}{\operatorname{argmin}} \{\, d(v(t), C_i), C_i \in N_i \,\} \tag{1}$$

If the winning cluster $N_{win}(t)$ is close enough (distance is below a dynamically determined threshold), then $v(t)$ is placed in that cluster otherwise a new cluster is created. Thus we assume a simple nearest neighbor algorithm is used. In our discussions we assume that the Euclidean distance metric is used for clustering, but any distance or similarity metric could be used. In addition, any other technique could be used to determine the winning cluster. The salient feature of SOStream is the weighted density concept described in the rest of this paper.

In the following subsections we examine the algorithm in more detail.

3.2 Density-Based Centroid

SOStream uses a centroid to identify each cluster. However the manner in which the centroid is calculated is not just a simple arithmetic mean applied to all points, $v(t)$, in the cluster. The way we calculate the centroid is inspired by the technique to update weights for the winning competitive node in a Kohonen Network [2]. In our case the winner is a cluster and the weights are associated with neighboring clusters. The centroid of a cluster is updated in several ways:

- When an input vector is added to a cluster the centroid is updated using a traditional arithmetic mean approach.
- Centroids of clusters sufficiently close to the winning clusters have their centroids modified to be closer to the winning cluster's centroid. This approach is used to aid in merging similar clusters and increasing separation between different clusters.
- Fading also adjusts the centroid values. This will be discussed later in the paper.

As the first of these techniques is straightforward, we concentrate on the second one.

As described earlier, the winning cluster is the one that is closest to the input vector. Updates are performed to the centroids of clusters that are within the neighborhood of the winning cluster. This brings clusters in the neighborhood of the winner closer to the incoming data in a similar way as the neighbors of a winning competitive node in a SOM have their weights adjusted to be closer to the winner.

We define the neighborhood of the winner based on the idea of a $MinPts$ distance given by a minimum number of neighboring objects [1,9]. This distance is found by computing the Euclidean distance from any existing clusters to the winning cluster. Then all the distances are ordered in ascending order and the maximum of the first MinPts minimum distinct distances is chosen and used to represent the radius of the winning cluster. Thus, every cluster whose distance from the winning cluster is less then the computed radius is considered to be a neighbor of the winning cluster. Note that the efficiency of this calculation can be improved using a min heap type data structure.

Motivated by Kohonen's work [2], we propose that the centroid C_i of each cluster N_i that is within the neighborhood of the winning cluster N_{win} is modified to resemble the winner:

$$C_i(t+1) = C_i(t) + \alpha\beta(C_{win}(t) - C_i(t)) \tag{2}$$

α is a scaling factor and β is a weight which represents the amount of influence of the winner on a cluster. We define β as

$$\beta = e^{-\frac{d(C_i, C_{win})}{2(r_{win}^2)}} \tag{3}$$

where r_{win} denotes the radius of the winner. The definition of β ensures that $0 < \beta \leq 1$.

Next we need to prove that updating a centroid moves the cluster closer to the winning cluster, i.e.

$$d(C_i(t+1), C_{win}(t)) \leq d(C_i(t), C_{win}(t))$$

By the definition of Euclidean distance we have

$$d(C_i(t), C_{win}(t)) = \\ \sqrt{(v_1 - z_1)^2 + (v_2 - z_2)^2 +, ..., +(v_n - z_n)^2}$$

where $C_i(t) = \langle v_1, v_2, ..., v_n \rangle$; $C_{win}(t) = \langle z_1, z_2, ..., z_n \rangle$ and $C_i(t+1) = \langle v'_1, v'_2, ..., v'_n \rangle$. If we can show that $0 < \alpha \leq 2$ is a necessary condition for $(v'_i - z_i)^2 \leq (v_i - z_i)^2$, then one can easily show that $d(C_i(t+1), C_{win}(t)) \leq d(C_i(t), C_{win}(t))$ when $0 < \alpha \leq 2$ by the definition of Euclidean distance.

Given $0 \leq \alpha \leq 2$ then $0 \leq \alpha\beta \leq 2$ provided that $0 < \beta \leq 1$, we have:

$$\begin{aligned}
&-2 \leq -\alpha\beta \leq 0 \\
&-1 \leq 1 - \alpha\beta \leq 1 \\
&|1 - \alpha\beta| \leq 1 \\
&|v_i - z_i||1 - \alpha\beta| \leq |v_i - z_i| \quad \text{where} \quad (|v_i - z_i| > 0) \\
&|v_i - z_i| \geq |(v_i - z_i)(1 - \alpha\beta)| \\
&= |v_i - \alpha\beta v_i - z_i + z_i\alpha\beta| \\
&= |\alpha\beta(z_i - v_i) + v_i - z_i|
\end{aligned}$$

Algorithm 1. SOStream(DS, α, $MinPts$)

```
1  $SOStream \leftarrow NULL$;
2  foreach $v_t \in DS$ do
3      $win \leftarrow minDist\{ v_t, M(t) \}$;
4      if $|M(t)| \geq MinPts$ then
5          $winN \leftarrow findNeighbors(win, MinPts)$;
6          if $d(v_t, win) \leq win.Radius$ then
7              updateCluster($win$, $v_t$, $\alpha$, $winN$);
8          else
9              newCluster($v_t$);
10         $overlap \leftarrow findOverlap(win, winN)$;
11         if $|overlap| > 0$ then
12             mergeClusters($win$, $overlap$);
13     else
14         newCluster($v_t$);
```

by Equation 2, $\alpha\beta(z_i - v_i) + v_i = v_i'$. Then we have:

$$\begin{aligned} &|v_i - z_i| \geq |v_i' - z_i|, \quad \text{which implies :} \\ &0 < \alpha \leq 2 \Rightarrow |v_i - z_i| \geq |v_i' - z_i| \\ &(v_i - z_i)^2 \geq (v_i' - z_i)^2 \end{aligned}$$

This shows that if $0 < \alpha\beta \leq 2$ then each dimension in the modified cluster centroid will move closer to the winner centroid.

3.3 SOStream Algorithm

We are finally ready to discuss the SOStream algorithm in more detail. Here we decompose SOStream into seven basic algorithms. Algorithm 1 is the main algorithm and performs its main loop (step 2) for each input data point in the original data stream (DS). When a new input vector is obtained, the winner cluster is identified, its neighbors are found and either clusters are merged, the winning cluster is updated, or a new cluster is created.

Algorithm 2 returns all the neighbors of the winning cluster as well as its computed radius (threshold) at line 8. If the size of the neighborhood satisfies MinPts then, Algorithm 3 is called to find clusters that overlap with the winner. For each overlapping cluster its distance is calculated to the winning cluster. Any clusters with a distance less than that of the merge-threshold will be merged with the winner. This process can be triggered at regular intervals or if there is any shortage of memory.

Algorithm 5 is called to update an existing cluster. If the neighborhood of the winning cluster does not have a sufficient number of nearest neighbors or the input data $v(t)$ does not lie within the radius of the winning cluster then, Algorithm 6 is called to create a new cluster and add it to the model $M(t) \leftarrow M(t) \cup \{ v(t) \}$. Over time if this cluster does not succeed in attracting enough neighbors, then it will fade and we can remove

Algorithm 2. findNeighbors(win, $MinPts$)

```
if $|M(t)| \geq MinPts$ then
    foreach $N_i \in M(t)$ do
        //Determine the distance from any
        //cluster $N_i$ to the $winner$.
        $winDistN \leftarrow winDistN \cup \{\ d(win, N_i)\ \}$;
    Sort $winDistN$ distances in ascending order;
    //kDist: represent the radius (threshold) of the winning cluster.
    $kDist \leftarrow$ winDistN[MinPts - 1];
    $win.setRadius(kDist)$;
    //Find the nearest neighbors for winner.
    foreach $d_i \in winDistN$ do
        if $d_i \leq kDist$ then
            $winNN \leftarrow winNN \cup \{\ N_i\ \}$;
    return $winNN$;
else
    return $\emptyset$;
```

Algorithm 3. findOverlap(win, $winN$)

```
overlap $\leftarrow \emptyset$;
foreach $N_i \in winN$ do
    if $(win.ID()! = N_i.ID())$ then
        if $d(win, N_i)$ - $(win.Radius + N_i.Radius) < 0$ then
            overlap $\leftarrow$ overlap $\cup \{\ N_i\ \}$;
return overlap;
```

it. Fading of cluster structure is used to discount the influence of historical data points. SOStream can adapt to changes in data over time, by using a decay decreasing function associated with each cluster:

$$f(t) = 2^{\lambda t} \tag{4}$$

where, λ define the rate of decay of the weight over time and $t = (t_c - t_0)$ where, t_c denote the current time and t_0 is the creation time of the cluster.

SOStream uses centroid clustering to represent the cluster center and does not store data points. The frequency count n determines the weight of each cluster. Aging is accomplished by reducing the count over time:

$$n_{i+1} = n_i 2^{\lambda t} \tag{5}$$

SOStream checks for clusters that are fading and removes any cluster that reaches a defined weight. See Algorithm 7. We do not explicitly show the fading function in Algorithm 1 as its use is optional. The fading function can be explicitly called or called at a regular time intervals which gives the flexibility and efficiency of using fading.

Algorithm 4 merges two clusters and set the new created cluster as the new winner and continue to test other clusters for merge. Based on experiments we expect the

Algorithm 4. mergeClusters(win, overlap)

```
foreach $N_i \in overlap$ do
    if $d(N_i, win) < mergeThreshold$ then
        //Equation 7 and 8 are used to merge two clusters.
        $merge(N_i, win)$;
```

Algorithm 5. updateCluster(win, v_t, α, $winN$)

```
//This method incrementally update the centroid of the winner.
win.updateCentroid($v_t$);
//Frequency-counter is incremented.
win.counter++;
//Modify the winning neighborhood to resemble winning cluster. The method
//$adjustCentroid$ is computed using equation 2.
winRadius ← win.Radius;
foreach $N_i \in winN$ do
    widthN ← $(winRadius)^2$;
    influence ← exp(- d($N_i$, $win$) / (2 widthN));
    $N_i$.adjustCentroid(win.getCentroid(), $\alpha$, influence);
```

number of clusters to be small (relative to the number of data points) as the merge/fade is happening online.

3.4 Online Merging

With data stream clustering, the creation of clusters in a quick online manner may result in many small micro-clusters. As a result, many earlier stream clustering algorithms created special offline components to perform a merging of similar micro-clusters into larger clusters. In SOStream, merging is efficiently performed online at each time step as an integral part of the algorithm by only considering the neighborhood of the winning cluster.

Centroid clustering is a well known clustering technique, where the centroid is the mean of all the points within the cluster. In data stream, incoming data points are incrementally clustered with the centroid of the nearest cluster. Over time the clusters change their original position and may result in overlapping with other clusters. As a result, clusters may be merged into one cluster. Recall that each cluster N_i has a radius r_i associated with it. Two clusters are said to overlap if the spheres in d-dimensional space defined by the radius of each cluster overlap. We merge clusters if they overlap with a distance that is less than the merge-threshold. Hence, the threshold value is a determining factor for the number of clusters. The impact of this threshold will be analyzed in section 4.

Merging procedure: Let S represent a set of clusters from the neighborhood of the wining cluster $S = \{ N_1, N_2, ..., N_k \}$. Two clusters in the neighborhood S are said to overlap if

$$d(C_i, C_j) - (r_i + r_j) < 0 \tag{6}$$

Algorithm 6. newCluster(v_t)

```
//Set $v_t$ as a centroid for the new cluster.
$N_{t+1} \leftarrow$ new_Cluster($v_t$);
//Set the frequency-count to 1.
$N_{t+1}$.counter $\leftarrow$ 1;
//Initialize the radius with 0. The radius gets computed only for winning clusters.
$N_{t+1}$.Radius $\leftarrow$ 0;
$M(t+1) \leftarrow M(t) \cup \{ N_{t+1} \}$;
```

Algorithm 7. fadingAll()

```
foreach $N_i \in M(t)$ do
    $N_i.fading(t, \lambda)$;
    if $N_i.counter < fadeThreshold$ then
        remove $N_i$;
```

These clusters will be merged only if the distance between the two centroids is less than or equal to the threshold value. By merging, a new cluster N_y is created by first, finding the weight of each cluster and then computing the weighted centroid for the new cluster. This is achieved by

$$N_y = (w_i a_i + w_j b_i)/(w_i + w_j) \tag{7}$$

where, w_i, w_j are the number of points within cluster N_i, N_j and a_i, b_i are the i^{th} dimension of the weighted centroids.

We compute the new cluster's radius r_y dynamically by selecting the larger of both sums

$$r_y = max\{ d(C_y, C_i) + r_i, d(C_y, C_j) + r_j) \}. \tag{8}$$

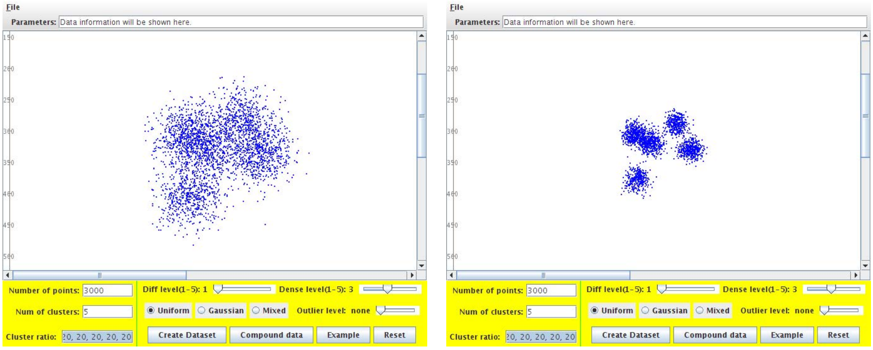
We choose the largest radius to avoid losing any data point within the clusters.

4 Experiments

In this section we compare the performance of SOStream with two recent data stream clustering algorithms namely, MR-Stream and D-Stream. Our experiments were performed using synthetic datasets and the KDD Cup'99 dataset which was also used for evaluation in [8,9,6,5]. SOStream is implemented in C++ and the experiments are conducted on a machine with an Intel Centrino Duo 2.2 GHz processor and Linux Ubuntu 9.10 ($x86\_64$) as the operating system. For our test we selected the input parameters α, λ and $MinPts$ where SOStream provided the best results. In the following subsections we evaluate the ability of SOStream to detect clusters in evolving data streams and the resulting cluster quality.

4.1 Synthetic Data

In this test we generate a synthetic data stream to demonstrate SOStream's capacity of distinguishing overlapping clusters.



(a)

Parameters: Data information will be shown here.

Number of points: 3000

Num of clusters: 5

Cluster ratio: 20, 20, 20, 20, 20

Diff level(1-5): 1

Dense level(1-5): 3

● Uniform ○ Gaussian ○ Mixed

Outlier level: none

(b)

Parameters: Data information will be shown here.

Number of points: 3000

Num of clusters: 5

Cluster ratio: 20, 20, 20, 20, 20

Diff level(1-5): 1

Dense level(1-5): 3

● Uniform ○ Gaussian ○ Mixed

Outlier level: none

Fig. 1. (a) Data points of stream with 5 overlapping clusters and (b) Show SOStream capability to distinguish overlapped clusters. For visualizing cluster structure, we do not utilize Fading or Merging.

We used the dataset generator described in [15]. This generator is written in Java and we can control the density of the cluster, data size and noise (outlier) level. In Figure 1, the synthetic dataset has 3000 data points with no noise (outliers) added. It contains five convex-shaped clusters that overlap. It can be noted from Figure 1(b) that SOStream is able to detect the five clusters and also we can observe that the clusters are clearly separated. This is due to the use of CURE and SOM-like updating of centroids which is the most significant innovation of SOStream versus previous data stream clustering algorithms. Based on experiments we selected $\alpha = 0.1$ and $MinPts = 2$.

4.2 Real-World Dataset

Many recent streaming algorithms, such as [8,9,6,5] have been tested with KDD CUP'99 dataset to evaluate its performance. We use the same dataset to evaluate the clustering quality of SOStream algorithm. This dataset was developed with an effort to examine Network Intrusion Detection System in the Air Force base network [4]. It embeds realistic attacks into the normal network traffic. In this dataset there are a total of 42 available attributes out of which only 34 continuous attributes were considered. We compare SOStream with MR-Stream as it has been shown that MR-Stream outperformed D-Stream and CluStream. We are using the same clustering quality comparison method used by [5] which is an evaluation method that computes the average purity defined by [9,6]:

$$\text{purity} = \frac{\sum_{i=1}^{K} \frac{|N_i^d|}{|N_i|}}{K} 100\% \tag{9}$$

where K is the number of real clusters, $|N_i^d|$ is the number of points that dominate the cluster label within each cluster, and $|N_i|$ is the total number of points in each cluster.

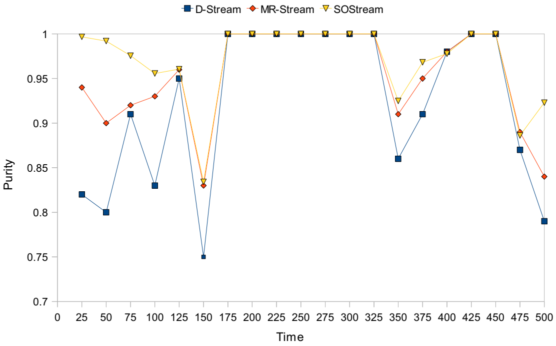


Fig. 2. SOStream clustering quality horizon = 1K, Stream speed = 1K. The quality Evaluation for MR-Stream and D-Stream is retrieved from [5].

Cluster structure fades with time which means some clusters may be deleted. This will effect the computation of purity. In order to efficiently compute the purity of the arriving data points, a predefined window (known as horizon) is used [9].

The dataset consists of 494,000 records. Using the same experimental setup as [5], the speed of the stream is set to 1000 points per second and the horizon is set to 1000 data points. There are approximately 500 time instances which are divided into intervals of 25 time instances where each instance shows an average purity value. Figure 2 shows the clustering quality comparison between SOStream, and the results of MR-Stream and D-Stream. These three algorithms all achieve an average purity of 1 between the time instance 196 and 320 since there is only one cluster appearing in one window. It can also be noticed from the graph that SOStream is able to maintain a better average purity than MR-Stream and D-Stream. The three approaches show the same pattern, this is related to the particular data set used and not to the different methods used. The overall average purity was above 95% for SOStream. In this experiment, the parameters used were again $\alpha = 0.1$, $\lambda = 0.1$ and $MinPts = 2$.

4.3 Parameter Analysis

One might consider a vector $(\alpha_1, ..., \alpha_n)$ in order to optimize the scaling factor. Because this paper focuses on a new stream clustering algorithm, we leave the optimization for future work. Table 2 shows how the parameter MinPts affects the behavior of SOStream and the average purity at different intervals of data points. For this test we used the KDD Cup'99 dataset with scaling factor, $\alpha = 0.1$. For Table 3 we changed the scaling factor parameter to $\alpha = 0.3$. On this particular test, SOStreams capability to cluster at a perfect purity level is evident for MinPts = 3. However, in order to convey and contrast SOStream improved performance against other stream clustering algorithms we used an average purity measure of different MinPts. To compare SOStream, we have derived the values for MR-Stream and D-Stream from [5]. In Table 4 the same dataset is used by MR-Stream, D-Stream and SOStream which allows us to determine the improvement

Table 2. Comparing average purity for different MinPts for $\alpha = 0.1$

	$\alpha = 0.1$			
Data Points	MinPts = 3	MinPts = 5	MinPts = 10	Mean
25000	0.983	0.990	0.921	0.965
75000	0.917	0.982	0.968	0.955
125000	0.907	0.973	1.000	0.960
175000	0.876	0.974	0.937	0.929
225000	0.876	0.974	0.937	0.929
275000	0.876	0.974	0.937	0.929
325000	0.876	0.974	0.937	0.929
375000	0.895	0.975	0.919	0.929
425000	0.907	0.975	0.963	0.949
475000	0.934	0.977	0.935	0.949
Mean	0.899	0.976	0.932	0.936

Table 3. Comparing average purity for different MinPts for $\alpha = 0.3$

	$\alpha = 0.3$			
Data Points	MinPts = 3	MinPts = 5	MinPts = 10	Mean
25000	0.999	0.938	0.914	0.950
75000	0.998	0.996	0.962	0.985
125000	0.998	0.997	0.890	0.961
175000	0.995	0.993	1.000	0.996
225000	0.995	0.993	1.000	0.996
275000	0.995	0.993	1.000	0.996
325000	0.995	0.993	1.000	0.996
375000	0.996	0.991	0.877	0.955
425000	0.996	0.992	0.941	0.977
475000	0.997	0.993	0.946	0.979
Mean	0.996	0.991	0.943	0.977

Table 4. Highlight the improvement SOStream compared to MR-Stream and D-Stream

Data Points	SOStream ($\alpha = 0.1$)	SOStream ($\alpha = 0.3$)	MR-Stream	Improvement to MR-Stream%	D-Stream	Improvement to D-Stream%
25000	0.965	0.950	0.94	2.592	0.82	15.027
75000	0.955	0.985	0.92	6.646	0.91	7.661
125000	0.960	0.961	0.96	0.000	0.95	1.182
175000	0.929	0.996	1	0	1	0
225000	0.929	0.996	1	0	1	0
275000	0.929	0.996	1	0	1	0
325000	0.929	0.996	1	0	1	0
375000	0.929	0.955	0.95	0.000	0.91	4.688
425000	0.949	0.977	1.00	-2.387	1.00	-2.387
475000	0.949	0.979	0.89	9.056	0.87	11.100
Mean	0.936	0.977	0.96	2.081	0.93	5.020

percentage between an average purity for SOStream and the results of MR-Stream and D-Stream. Over D-Stream, SOStream improves by an average of 5.0% and over MR-Stream it improves by 2.1%.

To test the merging threshold we used both quality evaluation and memory cost test on the same dataset with the matching parameters. We can observe from Figure 3 that the number of clusters residing in memory is low compared to the opposing algorithms. However, SOStream obtained a high purity (see Figure 2). This is due to the merging procedure that was presented earlier. From this study we have observed that a large merge threshold value causes the clusters to collapse and may result in only one cluster. On the other hand, a small threshold value will result in high memory cost.

4.4 Scalability and Complexity of SOStream

SOStream achieves high efficiency by storing the data structures in memory, where the updates of the stored synopses occur frequently in order to cope with the data stream. Using the same testing criteria as MR-Stream, we chose the high dimensional KDD CUP99 dataset. As we mentioned earlier, the data stream contains 494000 data points with 34 numerical attributes. We sample the memory cost every 25K records.

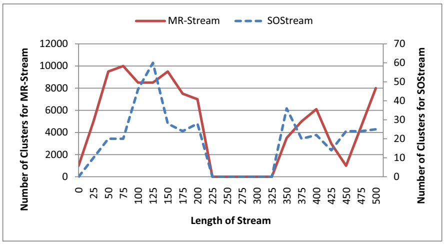


Fig. 3. SOStream memory cost over the length of the data stream. The Memory Evaluation for MR-Stream is retrieved from [5].

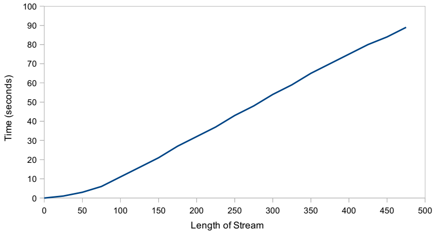


Fig. 4. SOStream execute time using high dimensional KDD CUP99 dataset with 34 numerical attributes. The sampling data rate is every 25K points.

To evaluate the memory usage of SOStream, we considered the total number of clusters in the memory. As can be seen from the Figure 3, our proposed algorithm demonstrated its low cost of memory by merging overlapping clusters which reduces the amount of space and time. In this experiment, the parameters used are $\alpha = 0.1$, MinPts = 2, fading and merging threshold = 0.1.

Figure 3 shows the memory utilized by SOStream and MR-Stream in terms of total number of clusters currently created. Our result indicates that SOStream utilizes less memory than MR-Stream. Also from Figure 3 we observe that the memory utilization profile is similar to our result. Between 0 and 200k data points, the memory utilization increases with different clusters which depreciate the clustering purity for all three algorithms. Between 200k and 350k data points, the data stream consisted mostly of one cluster, which explains why SOStream and MR-Stream consumed almost no memory. For both SOStream and MR-Stream the memory slightly increased between 350k and 400k data points and then, decreased around 450k.

Finally, we analyze the execution time and complexity of the SOStream. One appropriate data structure for Algorithm 2 is the min-heap data structure. The computation of the radius and neighbors of the winning micro-cluster takes $O(k\,log\,k)$. With n points, SOStream complexity is $O(nk\,log\,k)$, where k is the number of clusters. In the worst case $k = n$. In this case SOStream is $O(n^2\,log\,n)$. However, most data stream clustering algorithms make sure that k does not increase unbounded which reduces the more expensive operation. Other clustering operations such as remove, update and merge take $O(k)$. As shown in Figure 4, the algorithm shows that the execution time for clustering increases linearly with respect to time and number of data points.

5 Conclusion

In this paper, we proposed SOStream, which is an efficient streaming algorithm that is capable of distinguishing overlapping cluster in an online manner. The novel features of SOStream are the use of density based centroids, and an adaptive threshold. In addition, everything needed for stream clustering operations are included in a simple online algorithm. Our results show that SOStream outperformed MR-Stream and D-Stream in terms of purity and memory utilization. We are currently working on the development of a split clusters algorithm and creating outlier detection techniques based on SOStream.

References

1. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. of 2nd International Conference on Knowledge Discovery and, pp. 226–231 (1996)
2. Kohonen, T.: Self-Organized Formation of Topologically Correct Feature Maps. Biological Cybernetics 43, 59–69 (1982)
3. Guha, S., Rastogi, R., Shim, K.: Cure: an efficient clustering algorithm for large databases. SIGMOD Rec. 27, 73–84 (1998)
4. Hettich, S., Bay, S.D.: The UCI KDD Archive, University of California, Department of Information and Computer Science, Irvine, CA, USA (1999), `http://kdd.ics.uci.edu`
5. Wan, L., Ng, W.K., Dang, X.H., Yu, P.S., Zhang, K.: Density-based clustering of data streams at multiple resolutions. ACM Trans. Knowl. Discov. Data 3, 14:1–14:28 (2009)
6. Chen, Y., Tu, L.: Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2007, pp. 133–142. ACM, New York (2007)
7. Udommanetanakit, K., Rakthanmanon, T., Waiyamai, K.: E-Stream: Evolution-Based Technique for Stream Clustering. In: Alhajj, R., Gao, H., Li, X., Li, J., Zaïane, O.R. (eds.) ADMA 2007. LNCS (LNAI), vol. 4632, pp. 605–615. Springer, Heidelberg (2007)
8. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Databases, VLDB 2003, vol. 29, 81–92. VLDB Endowment (2003)
9. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: 2006 SIAM Conference on Data Mining, pp. 328–339 (2006)
10. Tasoulis, D.K., Ross, G., Adams, N.M.: Visualising the Cluster Structure of Data Streams. In: Berthold, M., Shawe-Taylor, J., Lavrač, N. (eds.) IDA 2007. LNCS, vol. 4723, pp. 81–92. Springer, Heidelberg (2007)
11. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. ACM SIGMOD Record 28(2), 49–60 (1999)
12. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for projected clustering of high dimensional data streams. In: Proceedings of the Thirtieth International Conference on Very Large Databases, VLDB 2004, vol. 30, pp. 852–863. VLDB Endowment (2004)
13. Tasoulis, D.K., Adams, N.M., Hand, D.J.: Unsupervised clustering in streaming data. In: Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops, pp. 638–642. IEEE Computer Society, Washington, DC (2006)
14. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. In: Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD), pp. 103–114 (1996)
15. Pei, Y., Zaïane, O.: A synthetic data generator for clustering and outlier analysis. Technical report, Computing Science Department, University of Alberta, Edmonton, Canada T6G 2E8 (2006)